STAT 5244 – Unsupervised Learning

Homework 2

Name: Chuyang Su UNI: cs4570

1 Mixture Models

1.1 EM Algorithm Derivation

Since we model count-valued data, assume each observation $x_i = (x_{i1}, \ldots, x_{ip}) \in \mathbb{N}_0^p$ is generated from a finite mixture of *independent* Poisson distributions:

$$p(x_i;\, \pi, \lambda) = \sum_{k=1}^{K} \pi_k \prod_{j=1}^{p} \frac{e^{-\lambda_{kj}} \lambda_{kj}^{x_{ij}}}{x_{ij}!}, \quad \pi_k \geq 0,\ \sum_{k=1}^{K} \pi_k = 1,\ \lambda_{kj} > 0.$$

Here $\pi = (\pi_1, \ldots, \pi_K)$ are mixture weights and $\lambda_k = (\lambda_{k1}, \ldots, \lambda_{kp})$ are component-wise Poisson means.

Latent variables. Introduce latent indicators $z_{ik} \in \{0, 1\}$ with $\sum_{k=1}^{K} z_{ik} = 1$, where $z_{ik} = 1$ if x_i comes from component k. The complete-data likelihood is

$$L_c(\pi, \lambda) = \prod_{i=1}^{n} \prod_{k=1}^{K} \left[\pi_k \prod_{j=1}^{p} \frac{e^{-\lambda_{kj}} \lambda_{kj}^{x_{ij}}}{x_{ij}!} \right]^{z_{ik}}.$$

Taking logs and dropping constants independent of (π, λ) (i.e., $\log x_{ij}!$) gives the complete-data log-likelihood

$$\ell_c(\pi, \lambda) \propto \sum_{i=1}^{n} \sum_{k=1}^{K} z_{ik} \left[\log \pi_k + \sum_{j=1}^{p} \left(x_{ij} \log \lambda_{kj} - \lambda_{kj} \right) \right].$$

E-step Derivation. Since the latent indicators z_{ik} are unobserved, we take their conditional expectation under the current parameters. Define

$$\gamma_{ik} := \mathbb{E}[z_{ik} \mid x_i; \pi^{(t)}, \lambda^{(t)}] = P(z_{ik} = 1 \mid x_i; \pi^{(t)}, \lambda^{(t)}),$$

which represents the posterior probability that observation x_i belongs to component k.

Using Bayes' theorem,

$$P(z_{ik} = 1 \mid x_i; \pi^{(t)}, \lambda^{(t)}) = \frac{P(z_{ik} = 1; \pi^{(t)})\, P(x_i \mid z_{ik} = 1; \lambda^{(t)})}{P(x_i; \pi^{(t)}, \lambda^{(t)})}.$$

Each term can be expressed as:

$$P(z_{ik} = 1; \pi^{(t)}) = \pi_k^{(t)}, \quad P(x_i \mid z_{ik} = 1; \lambda^{(t)}) = \prod_{j=1}^{p} \frac{e^{-\lambda_{kj}^{(t)}} (\lambda_{kj}^{(t)})^{x_{ij}}}{x_{ij}!},$$

and

$$P(x_i; \pi^{(t)}, \lambda^{(t)}) = \sum_{\ell=1}^{K} \pi_\ell^{(t)} \prod_{j=1}^{p} \frac{e^{-\lambda_{\ell j}^{(t)}} (\lambda_{\ell j}^{(t)})^{x_{ij}}}{x_{ij}!}.$$

Substituting these expressions into Bayes' rule yields:

$$\gamma_{ik} = \frac{\pi_k^{(t)} \prod_{j=1}^{p} e^{-\lambda_{kj}^{(t)}} (\lambda_{kj}^{(t)})^{x_{ij}} / x_{ij}!}{\sum_{\ell=1}^{K} \pi_\ell^{(t)} \prod_{j=1}^{p} e^{-\lambda_{\ell j}^{(t)}} (\lambda_{\ell j}^{(t)})^{x_{ij}} / x_{ij}!}.$$

Since the term $\prod_{j=1}^{p} x_{ij}!$ does not depend on k, it cancels out between numerator and denominator. Therefore, the final expression for the responsibilities is:

$$\boxed{\gamma_{ik} = \frac{\pi_k^{(t)} \prod_{j=1}^{p} e^{-\lambda_{kj}^{(t)}} (\lambda_{kj}^{(t)})^{x_{ij}}}{\sum_{\ell=1}^{K} \pi_\ell^{(t)} \prod_{j=1}^{p} e^{-\lambda_{\ell j}^{(t)}} (\lambda_{\ell j}^{(t)})^{x_{ij}}}}, \quad i = 1, \ldots, n, \quad k = 1, \ldots, K.$$

M-step. We maximize

$$Q(\pi, \lambda) = \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma_{ik} \left[\log \pi_k + \sum_{j=1}^{p} \left(x_{ij} \log \lambda_{kj} - \lambda_{kj} \right) \right]$$

subject to $\pi_k \geq 0$, $\sum_{k=1}^{K} \pi_k = 1$, and $\lambda_{kj} > 0$.

Update for π_k. Introduce a Lagrange multiplier η for the simplex constraint:

$$\mathcal{L}(\pi, \eta) = \sum_{k=1}^{K} \left(\sum_{i=1}^{n} \gamma_{ik} \right) \log \pi_k + \eta \left(1 - \sum_{k=1}^{K} \pi_k \right).$$

Setting the partial derivatives to zero,

$$\frac{\partial \mathcal{L}}{\partial \pi_k} = \frac{\sum_{i=1}^{n} \gamma_{ik}}{\pi_k} - \eta = 0 \quad \Longrightarrow \quad \pi_k = \frac{\sum_{i=1}^{n} \gamma_{ik}}{\eta}.$$

Summing over k and using $\sum_{k=1}^{K} \pi_k = 1$ gives

$$1 = \sum_{k=1}^{K} \pi_k = \frac{1}{\eta} \sum_{k=1}^{K} \sum_{i=1}^{n} \gamma_{ik} = \frac{1}{\eta} \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma_{ik} = \frac{1}{\eta} \sum_{i=1}^{n} 1 = \frac{n}{\eta} \implies \eta = n.$$

Hence

$$\boxed{\pi_k^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} \gamma_{ik}}.$$

(Concavity: $\partial^2 \mathcal{L} / \partial \pi_k^2 = -(\sum_i \gamma_{ik}) / \pi_k^2 < 0$.)

Update for λ_{kj}. For each (k, j), the terms of Q that involve λ_{kj} are

$$Q_{kj}(\lambda_{kj}) = \sum_{i=1}^{n} \gamma_{ik}\big(x_{ij} \log \lambda_{kj} - \lambda_{kj}\big).$$

Differentiate and set to zero:

$$\frac{\partial Q_{kj}}{\partial \lambda_{kj}} = \sum_{i=1}^{n} \gamma_{ik} \left(\frac{x_{ij}}{\lambda_{kj}} - 1 \right) = 0 \quad \Longrightarrow \quad \lambda_{kj} = \frac{\sum_{i=1}^{n} \gamma_{ik} x_{ij}}{\sum_{i=1}^{n} \gamma_{ik}}.$$

(Concavity: $\partial^2 Q_{kj}/\partial \lambda_{kj}^2 = -\sum_i \gamma_{ik} x_{ij}/\lambda_{kj}^2 < 0$ when $\lambda_{kj} > 0$.) Therefore

$$\boxed{\lambda_{kj}^{(t+1)} = \frac{\sum_{i=1}^{n} \gamma_{ik} x_{ij}}{\sum_{i=1}^{n} \gamma_{ik}}}.$$

1.2 Interpretation and Comparison of Poisson and Gaussian Mixture Models.

Both the Poisson Mixture Model (PMM) and the Gaussian Mixture Model (GMM) successfully converged and achieved almost identical clustering performance on the author–chapter word-frequency data.

The PMM converged in 16 iterations with a clustering purity of **0.8989**, while the GMM converged in 14 iterations with a purity of **0.9001**. The learned author–cluster mappings were consistent across models: Cluster 0 corresponds to *Shakespeare*, Cluster 1 to *London*, and Clusters 2–3 to *Austen*, indicating that Austen's writing exhibits two stylistically distinct sub-modes.

Examining the estimated centroids revealed interpretable linguistic patterns. The Shakespeare cluster assigns high weights to archaic forms such as *thou*, *thy*, and *hath*, capturing the syntax of Early Modern English. London's centroid emphasizes neutral verbs like *was*, *had*, and *were*, representing narrative realism, while Austen's two clusters differ primarily in pronoun and modal-verb usage: one dominated by *she, her, would, could* (social dialogue tone), and the other by *my, our, only, such* (introspective narration).

By inspecting the soft responsibilities γ_{ik}, chapters with $\max_k \gamma_{ik} < 0.6$ were identified as *low-certainty chapters*. These typically occur near stylistic transitions or among authors with overlapping vocabularies. Such ambiguous sections highlight that soft clustering provides richer insights than hard assignments.

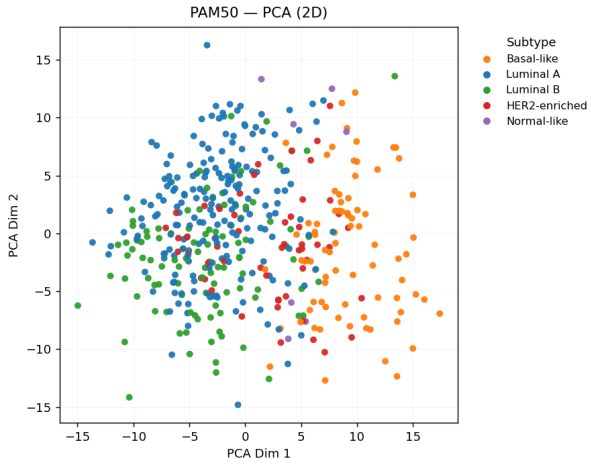
Overall, both mixture models achieve roughly 90% purity and uncover meaningful stylistic structures. While the GMM yields a marginally higher purity, the Poisson mixture remains theoretically more appropriate for discrete count data and offers comparable empirical performance.

2 Open-Ended Cluster Analysis - Breast Cancer gene expression data.

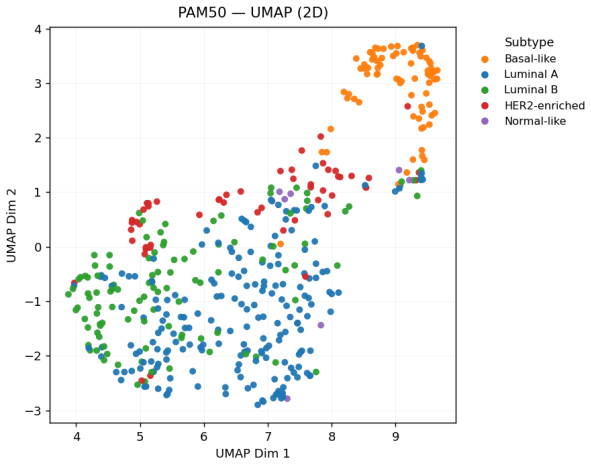
2.1 Apply clustering techniques to explore this data set.

To explore the internal structure of the BRCA gene-expression dataset, I first applied a two-step dimensionality reduction approach using **PCA** followed by **UMAP**.

Based on conclusions drawn in Homework 1, this strategy provides an effective balance between global and local structure preservation: PCA removes redundant noise variables while retaining the main variance directions, and UMAP further compresses the PCA-transformed space, producing a low-dimensional embedding that reveals local neighborhood structure and facilitates visual interpretation. Following the empirical results from Homework 1, I selected 30 principal components (the "elbow" point in the cumulative variance curve) for PCA, and adopted UMAP parameters $n\_neighbors = 10$, $min\_dist = 0.0$, and $n\_components = 2$ or 10, where the two-dimensional embedding was used for visualization and the ten-dimensional representation served as the feature space for clustering. The results are shown below:



(a) PAM50 subtypes visualized in PCA(2D) space.



(b) PAM50 subtypes visualized in UMAP(2D) space.

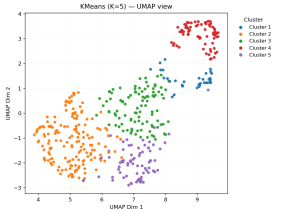
Subsequent clustering was performed using seven algorithms: **KMeans**, **Gaussian Mixture Model (GMM)**, **Spectral Clustering**, **DBSCAN**, and **Agglomerative Clustering** with three linkages (Ward, Average, and Complete). Given that the biological subtype reference (PAM50) contains five classes, all parametric clustering methods were set to $K = 5$, while DBSCAN automatically determined the number of clusters.

Among these, **Spectral Clustering** was treated as a special case. Because it internally constructs a similarity graph and performs Laplacian eigen-decomposition—conceptually similar to UMAP's graph-based embedding—it is inappropriate to apply Spectral Clustering directly in UMAP space. Doing so would perform a second spectral decomposition on an already nonlinear manifold, likely distorting the intrinsic geometry. Therefore, Spectral Clustering was conducted on the PCA(30D) features instead, ensuring consistency with its

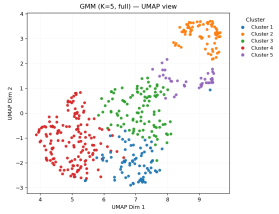
theoretical formulation.

For the Agglomerative Clustering, I compared only the **Ward**, **Average**, and **Complete** linkages, excluding **Single linkage** for three reasons: (1) it is highly sensitive to outliers and prone to chaining effects, which are amplified in high-dimensional continuous data; (2) its computational cost is high while its silhouette scores are unstable; and (3) the remaining three linkages already represent the major clustering behaviors.

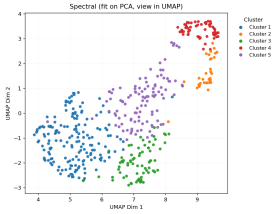
All the clustering results expect of hierachical clusterings is shown below:



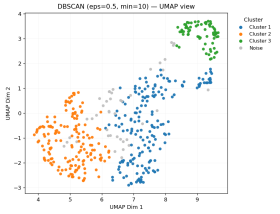
(a) K-Means



(b) Gaussian Mixture Model

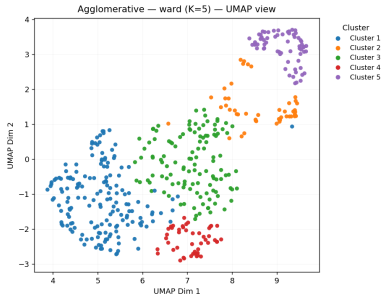


(c) Spectral Clustering in PCA(30)

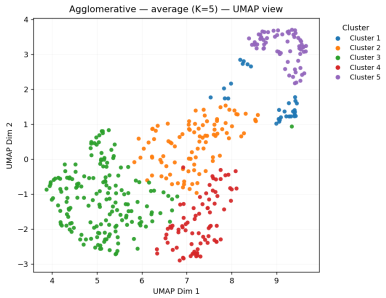


(d) DBSCAN

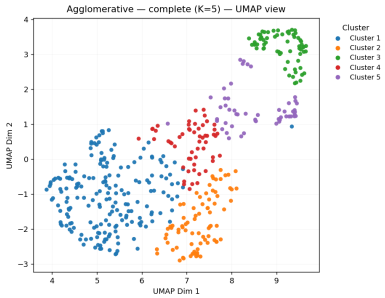
And three types of hierachical clusterings' results are:



(a) Ward



(b) Average



(c) Complete

Insights from the clustering results. Rather than directly comparing algorithmic performance, the focus here is on what each clustering method reveals about the *data manifold itself*. Across methods, two key structural patterns emerge. First, the dataset contains both compact and diffuse regions: **Basal-like** and **Normal-like** samples consistently form dense, isolated clusters, while **Luminal A** and **Luminal B** exhibit gradual overlap, suggesting a continuous transition rather than distinct boundaries. Second, the overall data distribution is non-spherical and heterogeneous in density, which explains why algorithms assuming isotropic clusters (KMeans, GMM, Ward) perform similarly and fail to separate overlapping subtypes.

DBSCAN highlights these density variations most clearly—it detects three main groups and labels the remaining sparse regions as noise. Although this underestimates the total number of subtypes, it accurately reflects the intrinsic density imbalance of the dataset: the rarest subtypes (**HER2-enriched**, **Normal-like**) are naturally absorbed as low-density regions. **Average linkage** produces smoother transitions that capture the gradual relationship between Luminal subtypes, while **Complete linkage** emphasizes inter-cluster separation. Taken together, these clustering outcomes indicate that the BRCA expression data contain

both discrete and continuous subtype structures—Basal-like being compact and distinct, whereas Luminal A/B lie on a continuum of transcriptional profiles—consistent with known biological heterogeneity among breast cancer subtypes.

2.2 Implement cluster validation techniques.

In this section, I evaluate the clustering results obtained in 2(a) through systematic parameter tuning and validation. For algorithms that require specifying the number of clusters K (*KMeans*, *GMM*, *Spectral Clustering*, and *Agglomerative Clustering* with three linkages), I performed a grid search over $K = 2, \ldots, 9$ and computed three complementary validation criteria:

- **Silhouette Score** — measures intra-cluster cohesion and inter-cluster separation;
- **Stability** — estimated via bootstrap resampling, quantified by the median Adjusted Rand Index (ARI) between full-data and bootstrap partitions;
- **Generalizability** — assessed by a train–test split, where cluster assignments were propagated from training to testing samples (via centroids or kNN label transfer), and the Silhouette score on the test set was reported.

For **DBSCAN**, which is density-based and does not rely on K, a two-dimensional grid search was conducted across $\varepsilon \in \{0.3, 0.4, 0.5, 0.6, 0.7\}$ and min_samples $\in \{5, 10, 15\}$. Each configuration was evaluated using the same three validation criteria. Additionally, the k-distance graph was inspected to locate the elbow point corresponding to a suitable ε threshold.

All validation computations were performed on the same reduced feature spaces, PCA (30D) for Spectral Clustering, UMAP(10D) for others, to ensure consistency with the embeddings generated in Section 2(a).

Results of Validation Metrics

Table 1 summarizes the best-performing configurations for each clustering method according to the combined Silhouette, Stability, and Generalizability criteria.

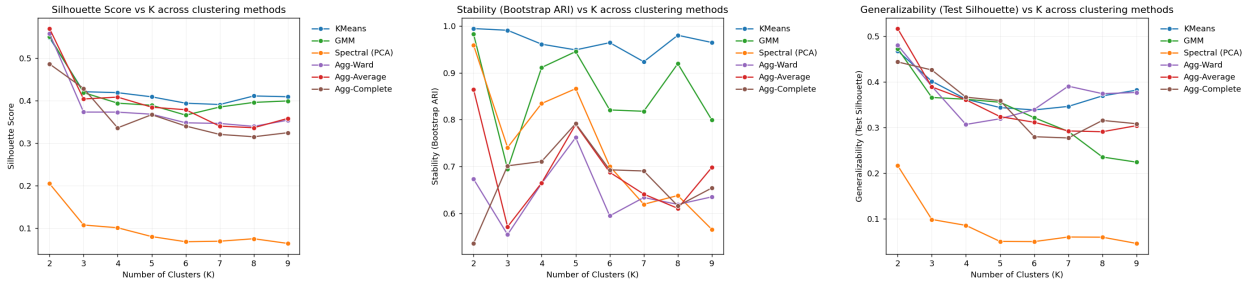
Table 1: Best validation results per method.

Method	K	ε	min_samples	Silhouette	Stability	Generalizability
KMeans	2	–	–	0.549	**0.994**	0.468
GMM	2	–	–	0.551	0.983	0.473
Spectral (PCA)	2	–	–	0.205	0.959	0.217
Agglomerative (Ward)	2	–	–	0.558	0.673	0.481
Agglomerative (Average)	2	–	–	**0.570**	0.864	**0.517**
Agglomerative (Complete)	2	–	–	0.487	0.535	0.444
DBSCAN	–	0.3	15	0.292	0.929	0.344

Across all K-based algorithms, the optimal number of clusters was consistently $K = 2$. This indicates that the BRCA gene-expression manifold is dominated by two large-scale density regions rather than five fully separated clusters, which is biologically plausible since Luminal A/B and HER2-enriched subtypes form a continuous spectrum, while Basal-like and Normal-like remain distinct.

Discussion and Selection of the Best Result

Among all evaluated methods, the **Agglomerative Clustering (Average linkage, $K = 2$)** achieved the highest Silhouette (0.57) together with strong stability (0.86) and generalizability (0.52), providing the best trade-off between cluster compactness and robustness. KMeans and GMM yielded similar two-cluster partitions with excellent stability (> 0.98), indicating consistent global structure but limited ability to separate overlapping subtypes. Spectral Clustering performed worse due to over-smoothing in PCA space, and DBSCAN emphasized density variation rather than discrete boundaries, identifying a few compact regions while labeling sparse points as noise.



(a) Silhouette vs K (b) Stability vs K (c) Generalizability vs K

Figure 4: Validation metrics across different cluster numbers.

Figure 4 illustrates the variation of validation metrics with respect to the number of clusters K.

2.3 Interpret your cluster findings with respect to the metadata provided.

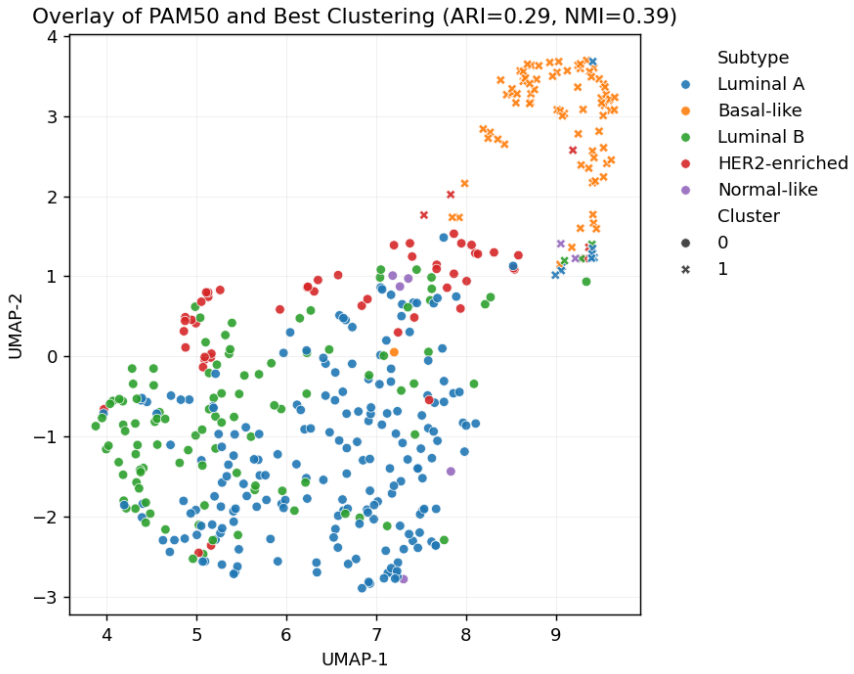


Figure 5: Overlay of PAM50 subtypes and the best clustering result (Agglomerative Average, $K = 2$) in UMAP(2D) space.

The two clusters discovered by the optimal Agglomerative (Average) model ($K = 2$) show a clear correspondence with the clinical PAM50 categories. As shown in Figure 5, one cluster primarily groups **Basal-like** and **Normal-like** tumors, while the other aggregates **Luminal A**, **Luminal B**, and **HER2-enriched** samples. This separation captures the well-known *luminal–basal dichotomy* in breast cancer biology: luminal tumors are typically hormone-receptor positive and exhibit similar expression patterns, whereas basal tumors represent the triple-negative subtype with distinct molecular characteristics. Therefore, the unsupervised clustering not only aligns with the known clinical taxonomy, but also highlights that the BRCA gene-expression landscape is dominated by two broad transcriptional regimes rather than five sharply separated classes.

A Appendix: Code Implementation

A.1 Problem 1b.

```
'''
Author: Chuyang Su cs4570@columbia.edu
Date: 2025-10-29 17:59:43
LastEditTime: 2025-10-30 14:06:41
FilePath: /Unsupervised-Learning-Homework/Homework 2/Code/Problem_1_b.py
Description:
    EM algorithm for a mixture of Poisson distributions and fit to the author data with K = 4 clusters.
'''
import os
import json
import numpy as np
import pandas as pd

DATA_PATH = r"Homework 2\Code\Data\authors.csv"
RESULT_DIR = r"Homework 2\Code\Result"
os.makedirs(RESULT_DIR, exist_ok=True)

def load_author_data(path):
    df = pd.read_csv(path)
    cols = list(df.columns)

    author_col = "" if "" in cols else cols[0]

    # Drop BookID column if exists
    df = df.drop(columns=cols[-1])

    y_names, y = np.unique(df[author_col].astype(str).values, return_inverse=True)
    feat_cols = [c for c in df.columns if c != author_col]
    X = df[feat_cols].to_numpy(dtype=float)

    return X, y, y_names, feat_cols

class PoissonMixtureResult:
    __slots__ = ("pi", "lmbda", "gamma", "loglik_hist", "converged", "n_iter")
    def __init__(self, pi, lmbda, gamma, loglik_hist, converged, n_iter):
        self.pi = pi
        self.lmbda = lmbda
        self.gamma = gamma
        self.loglik_hist = loglik_hist
        self.converged = converged
        self.n_iter = n_iter

def _softmax_logspace(logW, axis=1):
    m = np.max(logW, axis=axis, keepdims=True)
    W = np.exp(logW - m)
    W /= W.sum(axis=axis, keepdims=True)
    return W
```

```
48
49 def _logsumexp(A, axis=1):
50     m = np.max(A, axis=axis, keepdims=True)
51     return (m + np.log(np.sum(np.exp(A - m), axis=axis, keepdims=True))).
                                                squeeze(axis)
52
53 def _init_params(X, K, random_state=None):
54     rng = np.random.default_rng(random_state)
55     n, p = X.shape
56     gamma = rng.dirichlet(alpha=np.ones(K), size=n)
57     Nk = gamma.sum(axis=0) + 1e-16
58     pi = Nk / Nk.sum()
59     lmbda = (gamma.T @ X) / Nk[:, None]
60     lmbda = np.clip(lmbda, 1e-8, None)
61     return pi, lmbda, gamma
62
63 def poisson_mixture_em(X, K, tol=1e-6, max_iter=500, random_state=None,
                                        verbose=False):
64     X = np.asarray(X, dtype=float)
65     if np.any(X < 0) or (np.abs(X - np.round(X)) > 1e-10).any():
66         raise ValueError("X must be nonnegative integer counts.")
67     n, p = X.shape
68     rng = np.random.default_rng(random_state)
69
70     pi, lmbda, gamma = _init_params(X, K, random_state=rng)
71     loglik_hist = []
72     eps = 1e-12
73
74     for it in range(1, max_iter + 1):
75         log_pi = np.log(np.clip(pi, eps, 1.0))
76         log_lambda = np.log(np.clip(lmbda, eps, None))
77         log_px_given_k = X @ log_lambda.T - np.sum(lmbda, axis=1)[None, :]
78         log_w = log_pi[None, :] + log_px_given_k
79         gamma = _softmax_logspace(log_w, axis=1)
80
81         Nk = gamma.sum(axis=0) + eps
82         pi = Nk / n
83         lmbda = (gamma.T @ X) / Nk[:, None]
84         lmbda = np.clip(lmbda, 1e-12, None)
85
86         ll_vec = _logsumexp(log_w, axis=1)
87         ll = float(ll_vec.sum())
88         loglik_hist.append(ll)
89
90         if it > 1:
91             ll_prev = loglik_hist[-2]
92             denom = max(1.0, abs(ll_prev))
93             if (ll - ll_prev) / denom < tol:
94                 return PoissonMixtureResult(pi, lmbda, gamma, loglik_hist,
                                                        True, it)
95
96     return PoissonMixtureResult(pi, lmbda, gamma, loglik_hist, False, max_iter
                                        )
97
```

```python
def poisson_mixture_predict_proba(X, result):
    X = np.asarray(X, dtype=float)
    log_pi = np.log(np.clip(result.pi, 1e-12, 1.0))
    log_lambda = np.log(np.clip(result.lmbda, 1e-12, None))
    log_px_given_k = X @ log_lambda.T - np.sum(result.lmbda, axis=1)[None, :]
    log_w = log_pi[None, :] + log_px_given_k
    return _softmax_logspace(log_w, axis=1)

def poisson_mixture_predict(X, result):
    return np.argmax(poisson_mixture_predict_proba(X, result), axis=1)

# Main function
if __name__ == "__main__":
    X, y, y_names, feat_cols = load_author_data(DATA_PATH)

    K = len(y_names)
    res = poisson_mixture_em(X, K=K, tol=1e-6, max_iter=1000, random_state=0,
                                                verbose=False)
    hard = poisson_mixture_predict(X, res)

    cont = np.zeros((K, K), dtype=int)
    for yi, hi in zip(y, hard):
        cont[yi, hi] += 1
    purity = float(cont.max(axis=0).sum() / len(y))
    cluster_to_author = {int(k): str(y_names[int(np.argmax(cont[:, k]))]) for
                                                k in range(K)}

    threshold = 0.6
    max_resp = res.gamma.max(axis=1)
    low_certainty_idx = np.where(max_resp < threshold)[0].tolist()

    low_certainty_info = []
    for i in low_certainty_idx:
        low_certainty_info.append({
            "chapter_index": int(i),
            "author_true": str(y_names[y[i]]),
            "pred_cluster": int(hard[i]),
            "max_gamma": float(max_resp[i]),
            "responsibilities": [float(v) for v in res.gamma[i]]
        })

    out_path = os.path.join(RESULT_DIR, f"poisson_mixture_result.json")
    result_json = {
        "converged": bool(res.converged),
        "n_iter": int(res.n_iter),
        "final_loglik": float(res.loglik_hist[-1]),
        "n_authors": int(K),
        "authors": y_names.tolist(),
        "feature_columns": feat_cols,
        "em": {
            "pi": res.pi.astype(float).tolist(),
            "lambda": res.lmbda.astype(float).tolist(),
            "loglik_hist": [float(v) for v in res.loglik_hist]
        },
```

```
        "predictions": {
            "cluster_pred": [int(v) for v in hard],
            "responsibilities": res.gamma.astype(float).tolist()
        },
        "validation": {
            "purity": float(purity),
            "contingency": cont.astype(int).tolist(),
            "cluster_sizes": np.bincount(hard, minlength=K).astype(int).tolist
                                                    (),
            "cluster_to_author_map": {str(k): v for k, v in cluster_to_author.
                                                    items()},
            "low_certainty_chapters": low_certainty_info,
            "threshold": float(threshold)
        }
    }

    with open(out_path, "w", encoding="utf-8") as f:
        json.dump(result_json, f, ensure_ascii=False, indent=2)
```

A.2 Problem 1c.

```
'''
Author: Chuyang Su cs4570@columbia.edu
Date: 2025-10-29 21:47:48
LastEditTime: 2025-10-30 14:15:24
FilePath: /Unsupervised-Learning-Homework/Homework 2/Code/Problem_1_c.py
Description:
    Fit a Gaussian mixture model to the author data.
'''
import os
import json
import numpy as np
import pandas as pd

DATA_PATH = r"Homework 2\Code\Data\authors.csv"
RESULT_DIR = r"Homework 2\Code\Result"
os.makedirs(RESULT_DIR, exist_ok=True)

from Problem_1_b import load_author_data

class GMMResult:
    __slots__ = ("pi", "mu", "var", "gamma", "loglik_hist", "converged", "
                                            n_iter")
    def __init__(self, pi, mu, var, gamma, loglik_hist, converged, n_iter):
        self.pi = pi           # (K,)
        self.mu = mu           # (K, p)
        self.var = var         # (K, p)  diagonal covariances
        self.gamma = gamma     # (n, K)
        self.loglik_hist = loglik_hist
        self.converged = converged
        self.n_iter = n_iter

```

```python
def _log_gauss_diag(X, mu, var):
    eps = 1e-10
    var = np.clip(var, eps, None)
    n, p = X.shape
    K = mu.shape[0]
    log_det = np.sum(np.log(var), axis=1)
    inv_var = 1.0 / var
    quad = ((X[:, None, :] - mu[None, :, :]) ** 2 * inv_var[None, :, :]).sum(
                                                axis=2)
    return -0.5 * (p * np.log(2.0 * np.pi) + log_det[None, :] + quad)

def _softmax_logspace(logW, axis=1):
    m = np.max(logW, axis=axis, keepdims=True)
    W = np.exp(logW - m)
    W /= W.sum(axis=axis, keepdims=True)
    return W

def _logsumexp(A, axis=1):
    m = np.max(A, axis=axis, keepdims=True)
    return (m + np.log(np.sum(np.exp(A - m), axis=axis, keepdims=True))).
                                                squeeze(axis)

def _init_gmm_params(X, K, random_state=None):
    rng = np.random.default_rng(random_state)
    n, p = X.shape
    mu = np.empty((K, p))
    idx0 = rng.integers(0, n)
    mu[0] = X[idx0]
    d2 = np.sum((X - mu[0])**2, axis=1) + 1e-12
    for k in range(1, K):
        probs = d2 / d2.sum()
        idx = rng.choice(n, p=probs)
        mu[k] = X[idx]
        d2 = np.minimum(d2, np.sum((X - mu[k])**2, axis=1) + 1e-12)
    dist2 = ((X[:, None, :] - mu[None, :, :])**2).sum(axis=2)
    gamma = np.zeros((n, K))
    gamma[np.arange(n), np.argmin(dist2, axis=1)] = 1.0
    Nk = gamma.sum(axis=0) + 1e-12
    pi = Nk / n
    var = (gamma.T @ (X**2)) / Nk[:, None] - ( (gamma.T @ X) / Nk[:, None] )**
                                                2
    var = np.clip(var, 1e-6, None)
    return pi, mu, var, gamma

def gaussian_mixture_em(
    X, K, tol=1e-6, max_iter=500, random_state=None, verbose=False
):
    X = np.asarray(X, dtype=float)
    n, p = X.shape
    eps = 1e-12

    pi, mu, var, gamma = _init_gmm_params(X, K, random_state=random_state)
    loglik_hist = []

```

```
    for it in range(1, max_iter + 1):
        # E-step
        log_pi = np.log(np.clip(pi, eps, 1.0))  # (K,)
        log_px_given_k = _log_gauss_diag(X, mu, var)  # (n,K)
        log_w = log_pi[None, :] + log_px_given_k
        gamma = _softmax_logspace(log_w, axis=1)      # (n,K)

        # M-step
        Nk = gamma.sum(axis=0) + eps                  # (K,)
        pi = Nk / n
        mu = (gamma.T @ X) / Nk[:, None]              # (K,p)
        # diagonal covariance
        Ex2 = (gamma.T @ (X**2)) / Nk[:, None]        # (K,p)
        var = Ex2 - mu**2
        var = np.clip(var, 1e-6, None)

        # Log-likelihood
        ll = float(_logsumexp(log_w, axis=1).sum())
        loglik_hist.append(ll)

        if verbose and (it == 1 or it % 10 == 0):
            print(f"Iter {it:4d}  loglik={ll:.6f}")

        if it > 1:
            ll_prev = loglik_hist[-2]
            denom = max(1.0, abs(ll_prev))
            if (ll - ll_prev) / denom < tol:
                if verbose:
                    print(f"Converged at iter {it} Δ rel={(ll-ll_prev)/denom:.
                                                          3e}")
                return GMMResult(pi, mu, var, gamma, loglik_hist, True, it)

    if verbose:
        print("Reached max_iter without convergence.")
    return GMMResult(pi, mu, var, gamma, loglik_hist, False, max_iter)

def gmm_predict_proba(X, result):
    log_pi = np.log(np.clip(result.pi, 1e-12, 1.0))
    log_px_given_k = _log_gauss_diag(X, result.mu, result.var)
    log_w = log_pi[None, :] + log_px_given_k
    return _softmax_logspace(log_w, axis=1)

def gmm_predict(X, result):
    return np.argmax(gmm_predict_proba(X, result), axis=1)

# Main function
if __name__ == "__main__":
    X, y, y_names, feat_cols = load_author_data(DATA_PATH)
    K = len(y_names)

    res = gaussian_mixture_em(
        X, K=K, tol=1e-6, max_iter=1000, random_state=0, verbose=False
    )
    hard = gmm_predict(X, res)
```

```
135
136     cont = np.zeros((K, K), dtype=int)
137     for yi, hi in zip(y, hard):
138         cont[yi, hi] += 1
139     purity = float(cont.max(axis=0).sum() / len(y))
140     cluster_to_author = {
141         str(k): str(y_names[int(np.argmax(cont[:, k]))]) for k in range(K)
142     }
143
144     threshold = 0.6
145     max_resp = res.gamma.max(axis=1)
146     low_idx = np.where(max_resp < threshold)[0].tolist()
147     low_certainty_info = []
148     for i in low_idx:
149         low_certainty_info.append({
150             "chapter_index": int(i),
151             "author_true": str(y_names[y[i]]),
152             "pred_cluster": int(hard[i]),
153             "max_gamma": float(max_resp[i]),
154             "responsibilities": [float(v) for v in res.gamma[i]]
155         })
156
157     out_path = os.path.join(RESULT_DIR, f"gmm_result.json")
158
159     result_json = {
160         "converged": bool(res.converged),
161         "n_iter": int(res.n_iter),
162         "final_loglik": float(res.loglik_hist[-1]),
163         "n_authors": int(K),
164         "authors": y_names.tolist(),
165         "feature_columns": feat_cols,
166         "em": {
167             "pi": res.pi.astype(float).tolist(),
168             "mu": res.mu.astype(float).tolist(),
169             "var_diag": res.var.astype(float).tolist(),
170             "loglik_hist": [float(v) for v in res.loglik_hist]
171         },
172         "predictions": {
173             "cluster_pred": [int(v) for v in hard],
174             "responsibilities": res.gamma.astype(float).tolist()
175         },
176         "validation": {
177             "purity": float(purity),
178             "contingency": cont.astype(int).tolist(),
179             "cluster_sizes": np.bincount(hard, minlength=K).astype(int).tolist
                                                                    (),
180             "cluster_to_author_map": {str(k): v for k, v in cluster_to_author.
                                                                    items()},
181             "low_certainty_chapters": low_certainty_info,
182             "threshold": float(threshold)
183         }
184     }
185
186     with open(out_path, "w", encoding="utf-8") as f:
```

```
        json.dump(result_json, f, ensure_ascii=False, indent=2)
```

A.3 Problem 2a.

```
'''
Author: Chuyang Su cs4570@columbia.edu
Date: 2025-10-30 14:36:11
LastEditTime: 2025-10-31 02:14:40
FilePath: /Unsupervised-Learning-Homework/Homework 2/Code/Problem_2_a.py
Description:
    Apply clustering techniques(KMeans, GMM, Spectral Clustering,
                                Agglomerative Clustering, DBSCAN)
                                to explore the BRCA gene expression
                                 data set.
'''
import os
os.environ.setdefault("OMP_NUM_THREADS", "2")
os.environ.setdefault("MKL_NUM_THREADS", "2")
os.environ.setdefault("LOKY_MAX_CPU_COUNT", "8")

import warnings
from matplotlib import MatplotlibDeprecationWarning
warnings.filterwarnings("ignore", category=UserWarning,
                        message="n_jobs value 1 overridden to 1 by setting
                                                            random_state",
                        module=r"umap\.umap_")
warnings.filterwarnings("ignore", category=UserWarning,
                        message="KMeans is known to have a memory leak on
                                                            Windows with
                                                            MKL",
                        module=r"sklearn\.cluster\._kmeans")
warnings.filterwarnings("ignore", category=UserWarning,
                        message="Could not find the number of physical cores",
                        module=r"joblib\.externals\.loky\.backend\.context")
warnings.filterwarnings("ignore", category=MatplotlibDeprecationWarning)

from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, SpectralClustering,
                                    AgglomerativeClustering, DBSCAN
from sklearn.mixture import GaussianMixture
from packaging.version import parse as vparse
import sklearn
import umap

def load_brca(data_path: str):
```

```
    df = pd.read_csv(data_path, index_col=0)
    df.columns = (
        df.columns.astype(str)
        .str.strip()
        .str.replace("-", "_", regex=False)
        .str.replace(" ", "_", regex=False)
        .str.lower()
    )
    clinical_cols = [c for c in ["subtype", "er_status", "pr_status", "her2_status", "node", "metastasis"]
                     if c in df.columns]
    gene_cols = [c for c in df.columns if c not in clinical_cols]

    X = df[gene_cols].apply(pd.to_numeric, errors="coerce")
    X = X.fillna(X.median())
    X = StandardScaler().fit_transform(X.values)

    meta = df[clinical_cols].copy()
    return X, meta

def _encode(series):
    cats = series.astype(str).fillna("NA").values
    uniq = sorted(np.unique(cats))
    mapping = {c: i for i, c in enumerate(uniq)}
    idx = np.array([mapping[c] for c in cats], dtype=int)
    return idx, mapping

PAM50_COLOR_MAP = {
    "Luminal A": "#1f77b4",      # blue
    "Luminal B": "#2ca02c",      # green
    "Basal-like": "#ff7f0e",     # orange
    "HER2-enriched": "#d62728", # red
    "Normal-like": "#9467bd",    # purple
}

def scatter_by_labels(
    X2,
    labels,
    title,
    outpath,
    xlab="Dim 1",
    ylab="Dim 2",
    color_map=None,
    order=None,
    s=28,
    legend_title=None,
):
    from pathlib import Path
    outpath = Path(outpath); outpath.parent.mkdir(parents=True, exist_ok=True)

    lbl = pd.Series(labels)
    if pd.api.types.is_numeric_dtype(lbl):
        uniq_vals = sorted(lbl.unique())
        if -1 in uniq_vals:
```

```
            uniq_vals = [v for v in uniq_vals if v != -1] + [-1]
        uniq = [str(v) for v in uniq_vals]
        lbl_str = lbl.astype(str)
        legend_names = []
        for u in uniq_vals:
            if u == -1:
                legend_names.append("Noise")
            else:
                legend_names.append(f"Cluster {int(u)+1}")
    else:
        lbl_str = lbl.astype(str).fillna("NA")
        if order is None:
            uniq = sorted(lbl_str.unique())
        else:
            present = [u for u in order if str(u) in set(lbl_str)]
            rest = [u for u in lbl_str.unique() if u not in present]
            uniq = present + sorted(rest)
        legend_names = uniq

    colors = []
    if color_map is not None:
        for u in uniq:
            key = u if u in color_map else u.lower()
            colors.append(color_map.get(key, "#9E9E9E"))
    else:
        # clean, professional palette (Tableau 20-ish extended) + reserved gray for Noise
        base = ["#1f77b4","#ff7f0e","#2ca02c","#d62728","#9467bd",
                "#8c564b","#e377c2","#7f7f7f","#bcbd22","#17becf",
                "#4C78A8","#F58518","#54A24B","#E45756","#72B7B2",
                "#EECA3B","#B279A2","#FF9DA6","#9D755D","#BAB0AC"]
        # if there is a "Noise" class, make it light gray
        if pd.api.types.is_numeric_dtype(lbl) and (-1 in lbl.unique()):
            # map last label (Noise) to gray
            need = len(uniq) - 1
            while len(base) < need:
                base = base + base
            colors = base[:need] + ["#C0C0C0"]
        else:
            need = len(uniq)
            while len(base) < need:
                base = base + base
            colors = base[:need]

    # map label string -> palette index
    idx_map = {u: i for i, u in enumerate(uniq)}
    idx = lbl_str.map(idx_map).values
    point_colors = [colors[i] for i in idx]

    plt.figure(figsize=(7.2, 5.6), dpi=130)
    plt.scatter(X2[:, 0], X2[:, 1], c=point_colors, s=s, alpha=0.95, edgecolors='none')
    plt.xlabel(xlab); plt.ylabel(ylab); plt.title(title, pad=8)
    plt.grid(True, linewidth=0.3, alpha=0.25)
```

```
147
148     handles = [plt.Line2D([0], [0], marker='o', linestyle='',
149                           markersize=6, markerfacecolor=colors[i],
150                           markeredgecolor='none') for i in range(len(uniq))]
151     if legend_title is None:
152         legend_title = "Cluster" if pd.api.types.is_numeric_dtype(lbl) else "
                                                    Label"
153     if len(legend_names) <= 25:
154         plt.legend(handles, legend_names,
155                    bbox_to_anchor=(1.02, 1), loc='upper left',
156                    fontsize=9, frameon=False, title=legend_title)
157
158     plt.tight_layout()
159     plt.savefig(outpath, bbox_inches='tight')
160     plt.close()
161
162 def plot_pam50(X2, pam50_series, title, outpath, xlab, ylab):
163     outpath.parent.mkdir(parents=True, exist_ok=True)
164
165     # enforce canonical label order
166     ordered_labels = ["Basal-like", "Luminal A", "Luminal B", "HER2-enriched",
                                                  "Normal-like"]
167     pam50 = pam50_series.astype(str).fillna("NA")
168
169     # color list in that order
170     colors = [PAM50_COLOR_MAP.get(lbl, "#999999") for lbl in ordered_labels]
171
172     # map label -> index for plotting
173     idx_map = {lbl: i for i, lbl in enumerate(ordered_labels)}
174     idx = [idx_map.get(v, -1) for v in pam50]
175     point_colors = [colors[i] if i >= 0 else "#999999" for i in idx]
176
177     plt.figure(figsize=(7, 5.5), dpi=130)
178     plt.scatter(X2[:, 0], X2[:, 1], c=point_colors, s=30, alpha=0.95,
                                                 edgecolors="none")
179     plt.xlabel(xlab); plt.ylabel(ylab)
180     plt.title(title, pad=8)
181     plt.grid(True, linewidth=0.3, alpha=0.25)
182
183     # legend - fixed text order and color
184     handles = [plt.Line2D([0], [0], marker='o', linestyle='',
185                           markersize=6, markerfacecolor=colors[i],
186                           markeredgecolor='none')
187                for i in range(len(ordered_labels))]
188     legend_labels = ["Basal-like", "Luminal A", "Luminal B",
189                      "HER2-enriched", "Normal-like"]
190     plt.legend(handles, legend_labels,
191                bbox_to_anchor=(1.02, 1), loc='upper left',
192                fontsize=9, frameon=False, title="Subtype")
193
194     plt.tight_layout()
195     plt.savefig(outpath, bbox_inches="tight")
196     plt.close()
197
```

```python
def compute_embeddings(X, seed=0, n_pcs_feat=30):
    # PCA for visualization (2D) and as preprocessor for UMAP
    X_pca2 = PCA(n_components=2, random_state=seed).fit_transform(X)
    X_pca30 = PCA(n_components=min(n_pcs_feat, X.shape[1]), random_state=seed)
                                                .fit_transform(X)

    # UMAP on PCA-30
    u10 = umap.UMAP(n_components=10, n_neighbors=10, min_dist=0.0,
                    random_state=seed, metric="euclidean")
    X_umap10 = u10.fit_transform(X_pca30)

    u2 = umap.UMAP(n_components=2, n_neighbors=10, min_dist=0.0,
                   random_state=seed, metric="euclidean")
    X_umap2 = u2.fit_transform(X_pca30)

    return X_pca2, X_pca30, X_umap10, X_umap2

def clustering_all(X_umap10, X_umap2, X_spectral_input, outdir: Path, seed=0):
    k = 5  # PAM50 count

    labels = KMeans(n_clusters=k, n_init=20, random_state=seed).fit_predict(
                                                X_umap10)
    scatter_by_labels(
        X_umap2, labels,
        "KMeans (K=5) - UMAP view",
        outdir / "cluster__kmeans_k5_umap2.png",
        "UMAP Dim 1", "UMAP Dim 2",
        legend_title="Cluster"
    )

    # Spectral on PCA
    labels = SpectralClustering(
        n_clusters=k, affinity="nearest_neighbors",
        n_neighbors=10, assign_labels="kmeans",
        random_state=seed, n_init=10
    ).fit_predict(X_spectral_input)
    scatter_by_labels(
        X_umap2, labels,
        "Spectral (fit on PCA, view in UMAP)",
        outdir / "cluster__spectral_pca_umap2.png",
        "UMAP Dim 1", "UMAP Dim 2",
        legend_title="Cluster"
    )

    labels = GaussianMixture(
        n_components=k, covariance_type="full",
        random_state=seed, n_init=5
    ).fit_predict(X_umap10)
    scatter_by_labels(
        X_umap2, labels,
        "GMM (K=5, full) - UMAP view",
        outdir / "cluster__gmm_k5_umap2.png",
        "UMAP Dim 1", "UMAP Dim 2",
        legend_title="Cluster"
```

```
    )

    labels = DBSCAN(eps=0.5, min_samples=10, metric="euclidean").fit_predict(
                                                X_umap10)
    scatter_by_labels(
        X_umap2, labels,
        "DBSCAN (eps=0.5, min=10) - UMAP view",
        outdir / "cluster__dbscan_eps0p5_min10_umap2.png",
        "UMAP Dim 1", "UMAP Dim 2",
        legend_title="Cluster"
    )

    new_api = vparse(sklearn.__version__) >= vparse("1.2")

    labels = AgglomerativeClustering(n_clusters=k, linkage="ward").fit_predict
                                                (X_umap10)
    scatter_by_labels(
        X_umap2, labels,
        "Agglomerative - ward (K=5) - UMAP view",
        outdir / "cluster__agg_ward_k5_umap2.png",
        "UMAP Dim 1", "UMAP Dim 2",
        legend_title="Cluster"
    )

    if new_api:
        model = AgglomerativeClustering(n_clusters=k, linkage="average",
                                                    metric="euclidean")
    else:
        model = AgglomerativeClustering(n_clusters=k, linkage="average",
                                                    affinity="euclidean")
    labels = model.fit_predict(X_umap10)
    scatter_by_labels(
        X_umap2, labels,
        "Agglomerative - average (K=5) - UMAP view",
        outdir / "cluster__agg_average_k5_umap2.png",
        "UMAP Dim 1", "UMAP Dim 2",
        legend_title="Cluster"
    )

    if new_api:
        model = AgglomerativeClustering(n_clusters=k, linkage="complete",
                                                    metric="euclidean")
    else:
        model = AgglomerativeClustering(n_clusters=k, linkage="complete",
                                                    affinity="euclidean")
    labels = model.fit_predict(X_umap10)
    scatter_by_labels(
        X_umap2, labels,
        "Agglomerative - complete (K=5) - UMAP view",
        outdir / "cluster__agg_complete_k5_umap2.png",
        "UMAP Dim 1", "UMAP Dim 2",
        legend_title="Cluster"
    )

```

```

def run(
    data_path="Homework 2/Code/Data/BRCA_data.csv",
    outdir="Homework 2/Code/Result/Problem_2",
    seed=25
):
    outdir = Path(outdir); outdir.mkdir(parents=True, exist_ok=True)

    # data
    X, meta = load_brca(data_path)
    pam50 = meta["subtype"] if "subtype" in meta.columns else pd.Series(["NA"] * X.shape[0])

    # embeddings (PCA-2 and UMAP-2 are the only two embedding figures we save)
    X_pca2, X_pca30, X_umap10, X_umap2 = compute_embeddings(X, seed=seed, n_pcs_feat=30)
    plot_pam50(X_pca2, pam50, "PAM50 - PCA (2D)",
               outdir / "pam50__pca2.png", "PCA Dim 1", "PCA Dim 2")
    plot_pam50(X_umap2, pam50, "PAM50 - UMAP (2D)",
               outdir / "pam50__umap2.png", "UMAP Dim 1", "UMAP Dim 2")
    clustering_all(X_umap10, X_umap2, X_pca30, outdir, seed=seed)


if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument("--data_path", type=str, default="Homework 2/Code/Data/BRCA_data.csv")
    parser.add_argument("--outdir", type=str, default="Homework 2\Latex\Figures")
    parser.add_argument("--seed", type=int, default=25)
    args = parser.parse_args()
    run(data_path=args.data_path, outdir=args.outdir, seed=args.seed)
```

A.4 Problem 2b

```
'''
Author: Chuyang Su cs4570@columbia.edu
Date: 2025-10-31 02:09:44
LastEditTime: 2025-10-31 02:50:01
FilePath: /Unsupervised-Learning-Homework/Homework 2/Code/Problem_2_b.py
Description:
    Validation.
'''
import os
os.environ.setdefault("OMP_NUM_THREADS", "2")
os.environ.setdefault("MKL_NUM_THREADS", "2")
os.environ.setdefault("LOKY_MAX_CPU_COUNT", "8")

import warnings
warnings.filterwarnings("ignore")

```

```python
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path

from packaging.version import parse as vparse
import sklearn
from sklearn.cluster import KMeans, SpectralClustering,
                                        AgglomerativeClustering, DBSCAN
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, adjusted_rand_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.utils import check_random_state

from Problem_2_a import load_brca, compute_embeddings

def _kmeans_fit_predict(X, k, seed):
    return KMeans(n_clusters=k, n_init=20, random_state=seed).fit_predict(X)

def _gmm_fit_predict(X, k, seed):
    gm = GaussianMixture(n_components=k, covariance_type="full", random_state=
                                            seed, n_init=5)
    return gm.fit_predict(X), gm

def _spectral_fit_predict(X_pca30, k, seed):
    return SpectralClustering(
        n_clusters=k, affinity="nearest_neighbors",
        n_neighbors=10, assign_labels="kmeans",
        random_state=seed, n_init=10
    ).fit_predict(X_pca30)

def _agglo_fit_predict(X, k, linkage):
    new_api = vparse(sklearn.__version__) >= vparse("1.2")
    if linkage == "ward":
        model = AgglomerativeClustering(n_clusters=k, linkage="ward")
    else:
        if new_api:
            model = AgglomerativeClustering(n_clusters=k, linkage=linkage,
                                                    metric="euclidean")
        else:
            model = AgglomerativeClustering(n_clusters=k, linkage=linkage,
                                                    affinity="euclidean")
    return model.fit_predict(X)

def _dbscan_fit_predict(X, eps, min_samples):
    return DBSCAN(eps=eps, min_samples=min_samples, metric="euclidean").
                                            fit_predict(X)

def _silhouette_safe(X, labels):
    labs = np.asarray(labels)
    if len(np.unique(labs)) <= 1 or np.all(labs == -1):
        return np.nan
```

```
    try:
        return silhouette_score(X, labs)
    except Exception:
        return np.nan

def _bootstrap_stability(X, fit_fn, n_boot=10, seed=0):
    rng = check_random_state(seed)
    full_labels = fit_fn(X)
    if len(np.unique(full_labels)) <= 1:
        return np.nan
    n = X.shape[0]
    aris = []
    for _ in range(n_boot):
        idx = rng.choice(n, size=int(0.8 * n), replace=False)
        boot_labels = fit_fn(X[idx])
        if len(np.unique(boot_labels)) <= 1:
            aris.append(np.nan)
        else:
            aris.append(adjusted_rand_score(full_labels[idx], boot_labels))
    aris = np.array(aris, dtype=float)
    return float(np.nanmedian(aris))

def _generalizability_test_silhouette(X, labels, tag, k=None, seed=0):
    """
    80/20 split: fit clusterer on train, assign test labels via:
      - kmeans/gmm: model predict
      - spectral/agg/dbscan: kNN(label transfer) from train to test
    Return test silhouette.
    """
    rng = check_random_state(seed)
    n = X.shape[0]
    perm = rng.permutation(n)
    cut = int(0.8 * n)
    tr, te = perm[:cut], perm[cut:]
    Xtr, Xte = X[tr], X[te]

    # fit on train
    if tag == "kmeans":
        km = KMeans(n_clusters=k, n_init=20, random_state=seed).fit(Xtr)
        yte = km.predict(Xte)
    elif tag == "gmm":
        gm = GaussianMixture(n_components=k, covariance_type="full",
                                                random_state=seed, n_init=5).
                                                fit(Xtr)
        yte = gm.predict(Xte)
    else:
        # label transfer by kNN with train labels from full-data 'labels'
        ytr = np.asarray(labels)[tr]
        knn = KNeighborsClassifier(n_neighbors=10, weights="distance")
        knn.fit(Xtr, ytr)
        yte = knn.predict(Xte)

    return _silhouette_safe(Xte, yte)

```

```python
# ----------------- validation core -----------------
def validate_all(
    X_pca30, X_umap10,
    K_grid=range(2, 10),
    dbscan_eps=(0.3, 0.4, 0.5, 0.6, 0.7),
    dbscan_min_samples=(5, 10, 15),
    n_boot=10,
    seed=25
):
    rng = check_random_state(seed)
    records = []

    # --- K-based methods ---
    for K in K_grid:
        # KMeans (fit/view on UMAP10)
        km_labels = _kmeans_fit_predict(X_umap10, K, seed)
        sil = _silhouette_safe(X_umap10, km_labels)
        stab = _bootstrap_stability(X_umap10, lambda Z: _kmeans_fit_predict(Z, K, rng.randint(1e9)), n_boot, seed)
        gen = _generalizability_test_silhouette(X_umap10, km_labels, "kmeans", k=K, seed=seed)
        records.append(("kmeans", K, np.nan, np.nan, sil, stab, gen))

        # GMM (UMAP10)
        gmm_labels, _ = _gmm_fit_predict(X_umap10, K, seed)
        sil = _silhouette_safe(X_umap10, gmm_labels)
        stab = _bootstrap_stability(X_umap10, lambda Z: _gmm_fit_predict(Z, K, rng.randint(1e9))[0], n_boot, seed)
        gen = _generalizability_test_silhouette(X_umap10, gmm_labels, "gmm", k=K, seed=seed)
        records.append(("gmm", K, np.nan, np.nan, sil, stab, gen))

        # Spectral (fit on PCA30)
        sp_labels = _spectral_fit_predict(X_pca30, K, seed)
        sil = _silhouette_safe(X_pca30, sp_labels)
        stab = _bootstrap_stability(X_pca30, lambda Z: _spectral_fit_predict(Z, K, rng.randint(1e9)), n_boot, seed)
        gen = _generalizability_test_silhouette(X_pca30, sp_labels, "spectral", k=K, seed=seed)
        records.append(("spectral_pca", K, np.nan, np.nan, sil, stab, gen))

        # Agglomerative (UMAP10): ward/average/complete
        for lk in ("ward", "average", "complete"):
            ag_labels = _agglo_fit_predict(X_umap10, K, lk)
            sil = _silhouette_safe(X_umap10, ag_labels)
            stab = _bootstrap_stability(X_umap10, lambda Z, _lk=lk: _agglo_fit_predict(Z, K, _lk), n_boot, seed)
            gen = _generalizability_test_silhouette(X_umap10, ag_labels, f"agg_{lk}", k=K, seed=seed)
            records.append((f"agg_{lk}", K, np.nan, np.nan, sil, stab, gen))
```

```
160
161     # --- DBSCAN grid (UMAP10) ---
162     for eps in dbscan_eps:
163         for m in dbscan_min_samples:
164             db_labels = _dbscan_fit_predict(X_umap10, eps, m)
165             sil = _silhouette_safe(X_umap10, db_labels)
166             stab = _bootstrap_stability(
167                 X_umap10, lambda Z, _e=eps, _m=m: _dbscan_fit_predict(Z, _e,
                                                        _m), n_boot, seed
168             )
169             gen  = _generalizability_test_silhouette(X_umap10, db_labels, "
                                                    dbscan", k=None, seed=seed)
170             records.append(("dbscan", np.nan, eps, m, sil, stab, gen))
171
172     cols = ["method", "K", "eps", "min_samples", "silhouette", "stability", "
                                            generalizability"]
173     return pd.DataFrame.from_records(records, columns=cols)
174
175 def pick_best_per_method(df: pd.DataFrame) -> pd.DataFrame:
176     out = []
177     for meth in df["method"].unique():
178         sub = df[df["method"] == meth].copy()
179         if sub.empty:
180             continue
181         # rank by (silhouette, stability, generalizability)
182         sub["_rank"] = list(zip(-sub["silhouette"].fillna(-1e9),
183                                 -sub["stability"].fillna(-1e9),
184                                 -sub["generalizability"].fillna(-1e9)))
185         sub = sub.sort_values("_rank").drop(columns=["_rank"])
186         out.append(sub.iloc[[0]])
187     return pd.concat(out, ignore_index=True) if out else pd.DataFrame()
188
189 def run(
190     data_path="Homework 2/Code/Data/BRCA_data.csv",
191     outdir="Homework 2/Code/Result/Problem_2",
192     seed=25
193 ):
194     outdir = Path(outdir); outdir.mkdir(parents=True, exist_ok=True)
195
196     # data + embeddings (reuse 2a)
197     X, _ = load_brca(data_path)
198     # (pca2 not needed here)
199     _, X_pca30, X_umap10, _ = compute_embeddings(X, seed=seed, n_pcs_feat=30)
200
201     # validation
202     df = validate_all(
203         X_pca30, X_umap10,
204         K_grid=range(2, 10),
205         dbscan_eps=(0.3, 0.4, 0.5, 0.6, 0.7),
206         dbscan_min_samples=(5, 10, 15),
207         n_boot=10,
208         seed=seed
209     )
210
```

```
    # save all + best-per-method
    all_csv = outdir / "problem2b_validation.csv"
    best_csv = outdir / "problem2b_best.csv"
    df.to_csv(all_csv, index=False)
    pick_best_per_method(df).to_csv(best_csv, index=False)

    dfk = df[df["K"].notna()].copy()
    dfk["K"] = dfk["K"].astype(int)

    # unify method labels for legend
    label_map = {
        "kmeans": "KMeans",
        "gmm": "GMM",
        "spectral_pca": "Spectral (PCA)",
        "agg_ward": "Agg-Ward",
        "agg_average": "Agg-Average",
        "agg_complete": "Agg-Complete",
    }
    dfk["Method"] = dfk["method"].map(label_map)

    palette = {
        "KMeans": "#1f77b4",
        "GMM": "#2ca02c",
        "Spectral (PCA)": "#ff7f0e",
        "Agg-Ward": "#9467bd",
        "Agg-Average": "#d62728",
        "Agg-Complete": "#8c564b",
    }

    # ---- plotting helper ----
    def plot_metric(metric, ylabel):
        plt.figure(figsize=(7,5), dpi=130)
        sns.lineplot(data=dfk, x="K", y=metric, hue="Method", marker="o",
                                                palette=palette)
        plt.xlabel("Number of Clusters (K)")
        plt.ylabel(ylabel)
        plt.title(f"{ylabel} vs K across clustering methods")
        plt.legend(bbox_to_anchor=(1.02,1), loc="upper left", frameon=False)
        plt.grid(True, linewidth=0.4, alpha=0.3)
        plt.tight_layout()
        plt.savefig(outdir / f"validation_{metric.lower()}_vs_K.png",
                                                bbox_inches="tight")
        plt.close()

    # ---- draw three figures ----
    plot_metric("silhouette", "Silhouette Score")
    plot_metric("stability", "Stability (Bootstrap ARI)")
    plot_metric("generalizability", "Generalizability (Test Silhouette)")

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument("--data_path", type=str, default="Homework 2/Code/Data
                                            /BRCA_data.csv")
```

```
    parser.add_argument("--outdir", type=str, default="Homework 2/Latex/Figures")
    parser.add_argument("--seed", type=int, default=25)
    args = parser.parse_args()
    run(data_path=args.data_path, outdir=args.outdir, seed=args.seed)
```

A.5 Problem 2c.

```
'''
Author: Chuyang Su cs4570@columbia.edu
Date: 2025-10-31 02:57:20
LastEditTime: 2025-10-31 02:57:22
FilePath: /Unsupervised-Learning-Homework/Homework 2/Code/Problem_2_c.py
Description:
    plot the best model.
'''
import os
os.environ.setdefault("OMP_NUM_THREADS", "2")
os.environ.setdefault("MKL_NUM_THREADS", "2")

import warnings
warnings.filterwarnings("ignore")

from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import adjusted_rand_score, normalized_mutual_info_score

from Problem_2_a import load_brca, compute_embeddings, scatter_by_labels

def best_clustering_labels(X_umap10, k=2):
    model = AgglomerativeClustering(n_clusters=k, linkage="average")
    return model.fit_predict(X_umap10)

def run(
    data_path="Homework 2/Code/Data/BRCA_data.csv",
    outdir="Homework 2/Code/Result/Problem_2",
    seed=25
):
    outdir = Path(outdir)
    outdir.mkdir(parents=True, exist_ok=True)

    X, meta = load_brca(data_path)
    X_pca2, X_pca30, X_umap10, X_umap2 = compute_embeddings(X, seed=seed, n_pcs_feat=30)

    pam50 = meta["subtype"].astype(str).fillna("NA").values
    labels_best = best_clustering_labels(X_umap10, k=2)
```

```python

    ari = adjusted_rand_score(pam50, labels_best)
    nmi = normalized_mutual_info_score(pam50, labels_best)
    print(f"ARI = {ari:.3f}, NMI = {nmi:.3f}")

    # Define consistent color map
    PAM50_COLOR_MAP = {
        "Luminal A": "#1f77b4",
        "Luminal B": "#2ca02c",
        "Basal-like": "#ff7f0e",
        "HER2-enriched": "#d62728",
        "Normal-like": "#9467bd",
    }

    # Best clustering (Agg-Average, K=2)
    cmap_best = {str(i): c for i, c in enumerate(["#4C78A8", "#E45756"])}
    scatter_by_labels(
        X_umap2, labels_best.astype(str),
        title="Best Clustering (Agglomerative Average, K=2)",
        outpath=outdir / "2c_bestcluster_umap2.png",
        color_map=cmap_best,
        legend_title="Cluster"
    )

    # Combined overlay (color by PAM50, edge by cluster)
    df_plot = pd.DataFrame({
        "x": X_umap2[:,0], "y": X_umap2[:,1],
        "Cluster": labels_best.astype(str),
        "Subtype": pam50
    })

    plt.figure(figsize=(7,5.5), dpi=130)
    sns.scatterplot(
        data=df_plot, x="x", y="y",
        hue="Subtype", style="Cluster",
        palette=PAM50_COLOR_MAP, alpha=0.9, s=28
    )
    plt.title(f"Overlay of PAM50 and Best Clustering (ARI={ari:.2f}, NMI={nmi
                                            :.2f})")
    plt.xlabel("UMAP-1"); plt.ylabel("UMAP-2")
    plt.legend(bbox_to_anchor=(1.02,1), loc="upper left", frameon=False)
    plt.grid(True, linewidth=0.3, alpha=0.3)
    plt.tight_layout()
    plt.savefig(outdir / "2c_overlay_umap2.png", bbox_inches="tight")
    plt.close()

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument("--data_path", type=str, default="Homework 2/Code/Data
                                            /BRCA_data.csv")
    parser.add_argument("--outdir", type=str, default="Homework 2/Latex/
                                            Figures")
    parser.add_argument("--seed", type=int, default=25)
```

```
    args = parser.parse_args()
    run(data_path=args.data_path, outdir=args.outdir, seed=args.seed)
```