NYCU Pattern Recognition, Assignment #2

Student ID: 112550157, Name: 黃冠瑋

1 Dataset Introduction

In this report, we evaluate the performance of our classification models on several publicly available datasets from the UCI Machine Learning Repository.

Binary Classification

- **MAGIC Gamma Telescope** (19020 samples, 10 features): A dataset collected for distinguishing gamma rays from hadronic cosmic rays based on high-energy telescope measurements.
- **Taiwanese Bankruptcy Prediction** (6819 samples, 95 features): Financial indicators used to predict corporate bankruptcy risk.

Multi-class Classification

- **Dry Bean** (13611 samples, 16 features): Visual bean characteristics used to classify 7 bean varieties.
- **Covertype** (50000 samples, 54 features): Forest cover classification using cartographic variables across seven terrain types.

Dataset Summary

Dataset	Task Type	#Samples	#Features
MAGIC Gamma Telescope	Binary Classification	19020	10
Taiwanese Bankruptcy Prediction	Binary Classification	6819	95
Dry Bean	Multi-class Classification	13611	16
Covertype	Multi-class Classification	50000	54

Table 1: Summary of datasets used for experimental evaluation.

2 Task 1: FLD/LDA

2.1 Fisher's Linear Discriminant Analysis (FLD/LDA)

In this baseline experiment, Linear Discriminant Analysis (LDA) was applied to four datasets: MAGIC Gamma Telescope, Taiwanese Bankruptcy Prediction, Dry Bean, and Covertype. Each dataset was randomly split into training and testing subsets using an 80/20 ratio with stratified sampling. Standardization

was performed prior to model training. To evaluate linear separability, Fisher's criterion was computed before and after dimensionality reduction:

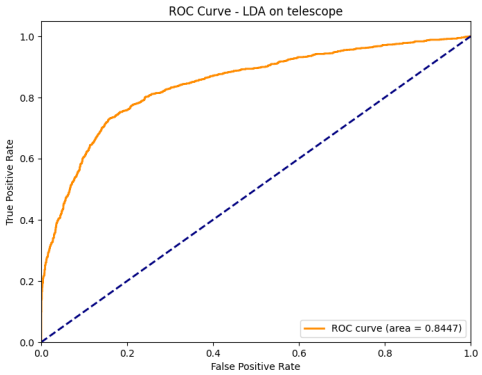
$$\text{Sep}(X, y) = \text{tr}(S_W^{-1} S_B),$$

where S_W and S_B denote the within-class and between-class scatter matrices, respectively.

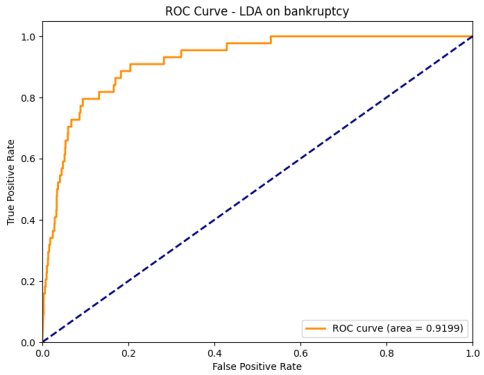
For binary datasets (telescope, bankruptcy) and multi-class datasets (dry_bean, cover_type), the separability values before and after LDA projection were identical. This is expected since LDA projects the data onto the full Fisher discriminant subspace of dimension $C - 1$, and the above trace equals the sum of all non-zero generalized eigenvalues of $S_W^{-1} S_B$, which are preserved under such projection.

Dataset	Accuracy	AUC	Separability (Before = After)
Telescope	0.7884	0.8447	0.4778
Bankruptcy	0.9589	0.9199	0.2701
Dry Bean	0.9045	N/A	31.9622
Covertype	0.6738	N/A	4.6680

For binary datasets, ROC curves were plotted and the resulting AUC values demonstrate strong discriminative ability, especially on the Bankruptcy dataset (AUC = 0.9199), even though accuracy alone may appear high due to class imbalance. The multi-class datasets show varying performance depending on class distribution difficulty, with Covertype exhibiting significant imbalance across classes, resulting in reduced macro-average performance (macro F1 = 0.51).



(a) Telescope



(b) Bankruptcy

These baseline results indicate that LDA provides competitive linear separation capability in binary classification problems. However, performance on complex multi-class problems suggests that additional dimensionality reduction or nonlinear models may be necessary. In subsequent sections, we extend this baseline with dimensional sweep experiments and comparison with PCA and nonlinear embeddings.

2.2 Dimensional Sweep Experiment on Multi-class Datasets

To evaluate the influence of LDA projection dimensionality, we conduct a dimensional sweep experiment on two multi-class datasets (Dry Bean and Covertype), varying the number of retained LDA dimensions $d = 1, \ldots, C - 1$. For each d, we compute the Fisher separability measure

$$\text{Sep}(d) = \text{tr}(S_W^{-1} S_B) = \sum_{i=1}^{d} \lambda_i,$$

and evaluate test accuracy using the corresponding LDA model. Table 2 summarizes the results.

Table 2: Dimensional sweep results for LDA on Dry Bean and Covertype datasets. Increasing projection dimension improves separability but does not change accuracy.

Dimension d	Dry Bean		Covertype	
	Separability	Accuracy	Separability	Accuracy
1	16.6745	0.9100	3.3904	0.6736
2	23.4657	0.9100	4.1875	0.6736
3	26.8633	0.9100	4.4241	0.6736
4	29.7036	0.9100	4.5439	0.6736
5	30.6492	0.9100	4.6174	0.6736
6	31.1237	0.9100	4.6652	0.6736

Analysis. The separability measure increases monotonically with d, consistent with the theoretical interpretation of Fisher eigenvalues where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{C-1} \geq 0$. Once the first discriminant direction is included, additional dimensions contribute progressively smaller improvements in Fisher ratio and thus offer diminishing returns for class discrimination.

For Dry Bean, accuracy saturates at $d = 1$, indicating that most discriminative variance is captured by the leading eigen-direction and the classes are nearly linearly separable in the strongest Fisher dimension. For Covertype, although separability increases substantially (from 3.39 to 4.67), accuracy does not improve due to significant class overlap and complex multi-modal structure that cannot be resolved by a linear boundary. This suggests the necessity of nonlinear approaches such as kernel methods or manifold learning.

Conclusion. The dimensional sweep experiment demonstrates that although increasing LDA projection dimensionality consistently enhances $\text{tr}(S_W^{-1} S_B)$, it does not necessarily improve classification accuracy. This confirms that **separability and accuracy are related but fundamentally distinct metrics**: a higher Fisher separability score does not guarantee enhanced predictive performance, particularly for complex multi-class datasets.

2.3 Effect of Class Priors on an Imbalanced Dataset (Taiwanese Bankruptcy Prediction)

In this experiment, we investigate the effect of class prior probabilities on the performance of Linear Discriminant Analysis (LDA) for a highly imbalanced binary classification dataset, the Taiwanese Bankruptcy Prediction dataset (6819 samples). LDA's decision rule for class ω_i is defined as:

$$\delta_i(x) = x^\top \Sigma^{-1} \mu_i - \frac{1}{2} \mu_i^\top \Sigma^{-1} \mu_i + \log P(\omega_i),$$

where μ_i and Σ denote the class mean and shared covariance matrix respectively, and $P(\omega_i)$ is the class prior. When the class distribution is highly skewed, an empirical prior (estimated from sample proportion) biases the classifier toward the majority class. In contrast, a balanced prior $P(\omega_1) = P(\omega_2) = 0.5$ shifts the decision boundary toward the majority class, improving minority class recognition.

We evaluate two settings: empirical prior vs. balanced prior. Table 3 summarizes the results on the test set.

Prior Setting	Accuracy	Balanced Acc.	Precision	Recall	F1	AUC
Empirical	**0.955**	0.638	**0.375**	0.294	0.330	**0.879**
Balanced	0.922	**0.780**	0.267	**0.627**	**0.374**	**0.879**

Table 3: Performance comparison of LDA under empirical vs. balanced class priors on the Taiwanese Bankruptcy dataset.

Analysis. Although the empirical prior achieves the highest accuracy (0.955), it performs poorly on the minority class, yielding low recall (0.294) and a weak balanced accuracy (0.638). With a balanced prior, recall improves dramatically to 0.627 (a 113% relative increase), and balanced accuracy rises to 0.780, along with an improvement in F1 score. Notably, AUC remains unchanged, indicating that class priors affect the decision boundary threshold rather than the underlying ranking ability of the model.

Conclusion. For imbalanced datasets, accuracy alone is misleading. Adjusting class priors significantly improves fairness and minority class detectability without harming overall ranking performance. Therefore, in real-world financial risk prediction tasks such as bankruptcy forecasting and credit risk management, metrics such as balanced accuracy, recall, and AUC should be prioritized over raw accuracy.

2.4 Summary of Task 1

In Task 1, we investigated the effectiveness of Fisher's Linear Discriminant Analysis (FLD/LDA) as a baseline dimensionality reduction and classification method on multiple datasets. We found that although Fisher separability ($\text{tr}(S_W^{-1} S_B)$) increases monotonically with additional discriminant dimensions, classification accuracy does not necessarily improve, underscoring that separability and predictive performance are correlated but fundamentally distinct metrics. The dimensional sweep experiment on Dry Bean and Covertype demonstrated diminishing returns beyond the leading eigen-direction(s), with accuracy saturated at $d = 1$ for Dry Bean and failing to improve for Covertype due to severe class overlap.

For the Taiwanese Bankruptcy dataset, adjusting LDA class priors notably improved balanced accuracy and minority recall without changing AUC, highlighting the importance of metric selection in imbalanced settings and the effect of decision boundary shifting rather than ranking ability.

Comparison with HW1 Baseline Results

To contextualize the benefits and limitations of LDA, we compare our Task 1 results with the baseline models trained in HW1, which operated directly in the original feature space (without dimensionality reduction). In HW1, traditional classifiers such as Random Forest achieved strong performance on the Dry Bean dataset (92% accuracy), while KNN and SVM performed substantially worse (72% and 64%), indicating that non-linear decision regions were needed. In contrast, the LDA-based model in Task 1 achieved 91% accuracy after projecting to $C - 1 = 6$ dimensions, nearly matching the HW1 best model while using far fewer features. This confirms that Dry Bean exhibits strong linear class separation and LDA effectively preserves discriminative structure.

For the Taiwanese Bankruptcy dataset, HW1 classifiers reached very high raw accuracy (97%) but poor AUC and balanced metrics (AUC ≈ 0.55), reflecting strong sensitivity to class imbalance. LDA in Task 1 showed similar accuracy and AUC performance and did not significantly outperform these baselines, indicating that linear discriminant projections cannot overcome noisy, overlapping financial feature spaces. This aligns with the findings in Tasks 2 and 3 that PCA and Isomap also provide limited benefit on this dataset.

Conclusion. LDA is highly effective on structured multi-class datasets with clear linear separation (e.g., Dry Bean), but provides limited improvement in noisy and imbalanced financial prediction tasks. These

observations motivate the investigation of PCA and nonlinear manifold learning presented in the following sections.

3 Task 2: PCA and classification

3.1 Experiment 1: Explained Variance Analysis

We first analyze the proportion of total variance preserved by the leading principal components in both datasets. Given the eigenvalues $\{\lambda_i\}_{i=1}^{d}$ of the sample covariance matrix, the variance explained by the first k components is

$$r(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i},$$

where $r(k)$ represents the cumulative explained variance ratio.

Taiwanese Bankruptcy Dataset. This dataset contains $d = 95$ financial features. The first few components account for a large portion of the total variance:

$$r(1) = 0.133, \quad r(5) = 0.346, \quad r(10) = 0.487, \quad r(20) = 0.648.$$

To preserve 95% of the variance, approximately $k = 56$ components are required ($r(56) = 0.966$). This implies that although the intrinsic dimensionality is significantly lower than 95, the variance is distributed across many directions, which suggests redundancy and collinearity among original financial indicators.

Dry Bean Dataset. This dataset contains $d = 16$ physical shape features. In contrast to the previous case, the first two components already explain 82% of the variance:

$$r(1) = 0.555, \quad r(2) = 0.819, \quad r(4) = 0.950,$$

and 95% is reached with only $k = 4$. This indicates a much lower intrinsic dimensionality, as most discriminative variance concentrates in a few principal directions.

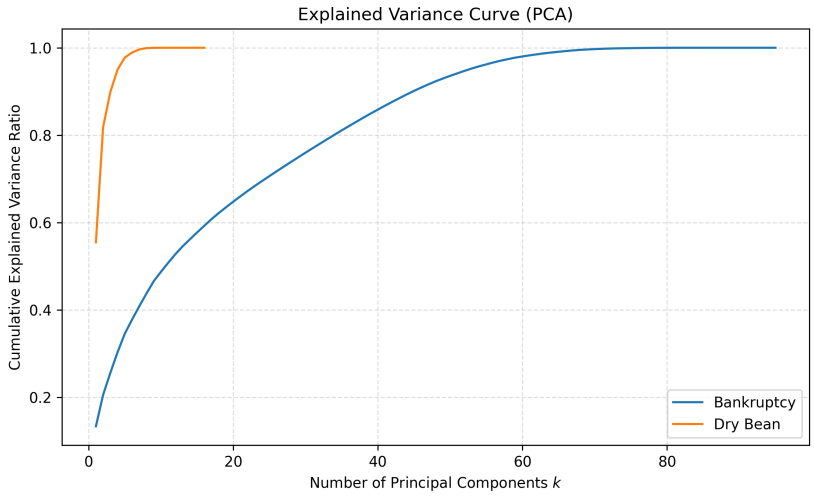


Figure 1: Cumulative explained variance ratio for Taiwanese Bankruptcy and Dry Bean datasets.

Observation. For the high-dimensional financial dataset, many components carry small fractions of variance, and consequently selecting k becomes non-trivial. In contrast, for Dry Bean the selection is straightforward because a small k already captures most essential structure. This motivates the following experiments on how PCA dimensionality affects classification performance and how PCA interacts differently with linear and non-linear classifiers.

3.2 Experiment 2: PCA Dimension Selection on Bankruptcy Dataset

In this experiment, we study how the PCA dimension affects the classification performance on the Taiwanese Bankruptcy dataset using logistic regression. The data are split into train/validation/test subsets, PCA is fitted on the training set and applied to validation and test sets, and we sweep the number of retained components

$$k \in \{2, 5, 10, 20, 30, 40, 60, 80\},$$

with the model without PCA ($k = 0$) as the baseline.

Table 4 summarizes the baseline and the best-performing configuration in terms of validation balanced accuracy. The other dimensions yield performances between these two settings.

Table 4: Logistic regression on Taiwanese Bankruptcy dataset with different PCA dimensions.

k	BA_{val}	BA_{test}	F1_{test}
0	0.588	0.574	0.598
60	0.600	0.587	0.618

The baseline model without PCA already achieves reasonably good performance, but applying PCA with $k = 60$ components (which retains approximately 96% of the variance) slightly improves both validation and test balanced accuracy, and also increases the macro F1-score from 0.598 to 0.618 on the test set. The gain is modest but consistent, suggesting that PCA helps stabilize the linear decision boundary by removing noisy or redundant directions in the high-dimensional financial feature space, while overly aggressive dimensionality reduction (e.g., $k \leq 20$) leads to information loss and degraded performance.

3.3 Experiment 3: Effect of PCA Across Classifiers and Datasets

In this experiment, we evaluate how PCA interacts with different classifiers and datasets. We compare logistic regression (a linear model) and RBF SVM (a non-linear kernel method), trained with and without PCA on both datasets. The PCA dimension is set to retain 95% cumulative variance, resulting in $k = 49$ for Bankruptcy and $k = 4$ for Dry Bean.

Table 5: Classifier comparison on Taiwanese Bankruptcy dataset.

Classifier	PCA	BA_{test}	F1_{test}	Accuracy
Logistic Regression	No	0.574	0.598	0.963
Logistic Regression	Yes ($k = 49$)	**0.575**	**0.600**	0.963
RBF SVM	No	0.500	0.492	**0.968**
RBF SVM	Yes ($k = 49$)	0.500	0.492	**0.968**

For the Bankruptcy dataset, PCA provides a slight improvement for logistic regression, raising the macro-F1 from 0.598 to 0.600 and balanced accuracy from 0.574 to 0.575. This aligns with the hypothesis that PCA stabilizes the linear decision boundary in a high-dimensional feature space with strong collinearity. In contrast, RBF SVM achieves high raw accuracy even without PCA and shows no improvement from dimensionality reduction, indicating that the non-linear kernel already captures high-order interactions without needing PCA preprocessing.

Table 6: Classifier comparison on Dry Bean dataset.

Classifier	PCA	$F1_{test}$	BA_{test}	Accuracy
Logistic Regression	No	**0.933**	**0.932**	**0.921**
Logistic Regression	Yes ($k = 4$)	0.892	0.890	0.889
RBF SVM	No	0.934	**0.933**	0.922
RBF SVM	Yes ($k = 4$)	0.901	0.899	0.900

For the Dry Bean dataset, PCA significantly degrades performance for both models, particularly logistic regression (macro-F1 drops from 0.933 to 0.892). Since the dataset is inherently low-dimensional ($d = 16$), strongly discriminative variance appears to be contained in higher-order components that are removed by PCA. This demonstrates that reducing dimensionality solely based on variance retention may remove label-relevant information when class separability is not aligned with directions of maximal variance.

Conclusion. PCA is beneficial for noisy high-dimensional financial data when combined with linear models, but does not improve non-linear kernels and may harm performance on inherently low-dimensional datasets. This highlights the importance of matching dimensionality reduction strategies to dataset characteristics rather than applying PCA uniformly.

3.4 Summary of Task 2

PCA provides modest but consistent improvement for linear models on high-dimensional data, while it does not benefit non-linear kernel models and may significantly degrade performance on low-dimensional datasets. This motivates exploring non-linear dimensionality reduction in Task 3.

4 Task 3: Nonlinear Dimensionality Reduction

4.1 Experiment 1: 2D Embedding Visualization on Dry Bean

In this task, we investigate how a nonlinear manifold learning method, Isomap, reorganizes the data geometry compared to PCA. We focus on the Dry Bean dataset, which contains seven bean classes in a moderate-dimensional feature space ($d = 16$).

All features are standardized, and we fit both PCA and Isomap on the full dataset. For PCA, we project the data to a two-dimensional subspace spanned by the leading principal components. For Isomap, we construct a k-nearest neighbor graph with $k = 10$, approximate geodesic distances on the graph, and apply multidimensional scaling to obtain a two-dimensional embedding.

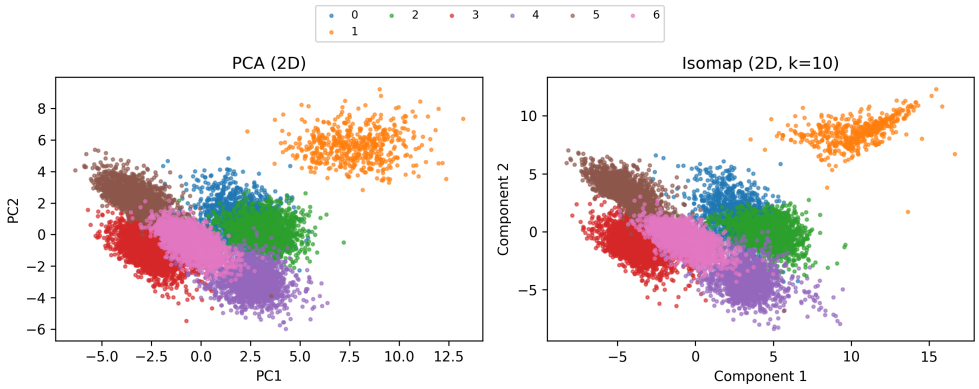


Figure 2: Two-dimensional embeddings of the Dry Bean dataset obtained by PCA (left) and Isomap with $k = 10$ neighbors (right). Colors indicate bean classes.

As shown in Figure 2, both methods produce seven clearly separated clusters, and class "1" forms a well-isolated group in the upper-right region for both embeddings. The PCA projection yields an approximately linear band of classes, while the Isomap embedding slightly warps and unfolds this band along the underlying manifold, emphasising local neighborhood relations. Overall, nonlinear Isomap does not dramatically change the global class separability compared with PCA, but it provides a more curved representation of the class structure that may better reflect the intrinsic geometry of the data.

4.2 Experiment 2: Isomap Dimension Sweep on Bankruptcy Dataset

To evaluate the effect of nonlinear manifold learning on classification performance, we apply Isomap followed by logistic regression on the Taiwanese Bankruptcy dataset. We sweep the output dimensionality of Isomap over

$$k \in \{2, 5, 10, 20, 30\},$$

using n_neighbors $= 10$ and compare against the baseline logistic regression model trained directly on the original standardized feature space ($k = 0$). The dataset is split into training, validation, and test sets using the same 60/20/20 protocol as in Task 2.

Table 7: Logistic regression performance with different Isomap output dimensions.

k	BA_{val}	BA_{test}	F1_{test}
0 (baseline)	0.588	0.574	0.598
2	0.566	0.556	0.590
5	0.567	0.556	0.588
10	0.555	0.555	0.584
20	0.566	0.566	0.597
30	**0.577**	**0.566**	**0.601**

As shown in Table 7, applying Isomap before logistic regression does not yield noticeable performance gains compared with the baseline. The best configuration ($k = 30$) achieves a test balanced accuracy of 0.566 and macro-F1 of 0.601, which is slightly below the baseline (0.574 and 0.598, respectively). Lower-dimensional embeddings (e.g., $k = 2$ or $k = 5$) perform considerably worse, suggesting that aggressive nonlinear dimensionality reduction removes important discriminative information.

These results contrast with the PCA experiment in Task 2, where a moderate reduction to $k = 60$ improved model stability and slightly enhanced performance. Isomap appears less effective for this high-dimensional, noisy, and heavily overlapping financial dataset, likely because the global geometric structure

is poorly approximated using geodesic distances on a sparse neighborhood graph. Nonlinear manifold learning therefore does not necessarily translate into improved downstream classification accuracy for real-world tabular data.

4.3 Experiment 3: Hyperparameter Sensitivity of Isomap

In this experiment, we study the sensitivity of Isomap to its key hyperparameter n_neighbors, which controls the locality of the neighborhood graph underlying the geodesic distance approximation. We fix the output dimension to n_components $= 10$ and vary

$$n\text{\_neighbors} \in \{5, 10, 20, 50\}.$$

We evaluate RBF SVM on the resulting embeddings and compare against a baseline RBF SVM trained directly in the original feature space.

Table 8: Effect of n_neighbors on RBF SVM performance (Bankruptcy dataset).

n_neighbors	BA_test	F1_test	Accuracy
Baseline (no Isomap)	0.500	0.492	0.968
5	0.499	0.491	0.966
10	0.499	0.491	0.966
20	0.500	0.492	0.968
50	0.500	0.492	0.967

As shown in Table 8, varying n_neighbors does not significantly affect classification performance, and all Isomap-based models perform nearly identically to the baseline. This indicates that Isomap provides little benefit for this dataset, regardless of how local geodesic distances are approximated. The Bankruptcy dataset exhibits noisy and highly overlapping class structures in a high-dimensional tabular space, which violates the manifold assumption that Isomap relies on. Consequently, geometric unfolding of the data does not reveal additional class separation that improves downstream classification.

These results reinforce that despite its nonlinear design, Isomap does not outperform the baseline kernel model in this setting, and its performance is insensitive to the choice of neighborhood size n_neighbors, highlighting practical instability and tuning overhead compared to PCA.

4.4 Summary of Task 3

Across three experiments, we compared PCA and Isomap in terms of embedding geometry, dimensionality reduction, and classification performance. On the Dry Bean dataset, Isomap produces a more curved and locally structured embedding than PCA, but both yield similar class separability. On the Taiwanese Bankruptcy dataset, nonlinear manifold learning fails to improve classification accuracy: dimensionality reduction via Isomap does not outperform the PCA baseline, and hyperparameter tuning of $n_{\text{neighbors}}$ has minimal effect. These findings demonstrate that nonlinear manifold learning can be beneficial for visualizing inherently low-dimensional structured data, but does not necessarily improve performance on noisy real-world tabular datasets with weak manifold structure. Therefore, dimensionality reduction should be selected based on dataset characteristics rather than assuming nonlinear methods are universally superior.

5 Final Conclusion

Dimensionality reduction is not a universally beneficial preprocessing step; its impact depends critically on dataset characteristics and model choice. PCA improves linear models on high-dimensional noisy

data, while Isomap offers better visualization but does not improve classification on non-manifold tabular datasets. Therefore, selecting appropriate dimensionality reduction methods requires understanding intrinsic data geometry rather than assuming that more complex nonlinear transformations will always yield benefits.

A Program Listing

data_loader.py

```python
import pandas as pd
import numpy as np
from loguru import logger
from sklearn.preprocessing import LabelEncoder
from typing import Tuple

from ucimlrepo import fetch_ucirepo

import os
import joblib

CACHE_DIR = os.environ.get("UCIMLREPO_DIR", "./data_cache")
os.makedirs(CACHE_DIR, exist_ok=True)
logger.info(f"Using data cache directory: {os.path.abspath(CACHE_DIR)}")

memory = joblib.Memory(CACHE_DIR, verbose=0)


def load_dataset(name: str) -> pd.DataFrame:
    """
    Load dataset by name from the specified directory.

    Parameters:
    - name: Name of the dataset (without file extension).
    - data_dir: Directory where datasets are stored.

    Returns:
    - DataFrame containing the loaded dataset.
    """

    if name == "telescope":
        return load_telescope()
    elif name == "bankruptcy":
        return load_bankruptcy()
    elif name == "cover_type":
        return load_cover_type()
    elif name == "dry_bean":
        return load_dry_bean()
    else:
        logger.error(f"Unknown dataset name: {name}")
        raise ValueError(f"Unknown dataset name: {name}")


def _preprocess_data(
    X: pd.DataFrame, y: pd.DataFrame
) -> Tuple[np.ndarray, np.ndarray]:
    X = X.values
    y = y.values

    le = LabelEncoder()
    y = le.fit_transform(y.ravel())

    return X, y


@memory.cache
def load_telescope() -> pd.DataFrame:
    logger.info("Loading Telescope (id=159) from ucimlrepo...")

    telescope = fetch_ucirepo(id=159)
```

```
    X = telescope.data.features
    y = telescope.data.targets

    return _preprocess_data(X, y)


@memory.cache
def load_bankruptcy() -> pd.DataFrame:
    logger.info("Loading Bank Bankruptcy (id=572) from ucimlrepo...")

    bank_bankruptcy = fetch_ucirepo(id=572)

    X = bank_bankruptcy.data.features
    y = bank_bankruptcy.data.targets

    return _preprocess_data(X, y)


@memory.cache
def load_cover_type() -> pd.DataFrame:
    logger.info("Loading Cover Type (id=31) from ucimlrepo...")

    cover_type = fetch_ucirepo(id=31)

    X = cover_type.data.features
    y = cover_type.data.targets

    sample = np.random.choice(len(X), size=int(5e4), replace=False)
    X = X.iloc[sample]
    y = y.iloc[sample]

    return _preprocess_data(X, y)


@memory.cache
def load_dry_bean() -> pd.DataFrame:
    logger.info("Loading Dry Bean Classification (id=602) from ucimlrepo...")

    dry_bean = fetch_ucirepo(id=602)

    X = dry_bean.data.features
    y = dry_bean.data.targets

    return _preprocess_data(X, y)
```

utils.py

```
import os
import numpy as np
from loguru import logger
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    roc_curve,
    auc,
)
from sklearn.metrics import accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt


```

```python
def k_fold_cross_validation(classifier_class, X, y, k=5, random_state=42):
    """
    Perform k-fold cross validation on the given classifier.

    Args:
        classifier_class (class): The classifier class to instantiate.
        X (np.ndarray): Feature matrix
        y (np.ndarray): Target labels
        k (int): Number of folds
        random_state (int): Random seed for reproducibility

    Returns:
        dict: Dictionary containing fold results and average metrics
    """
    logger.info(
        f"Starting {k}-fold cross validation for {classifier_class.__name__}..."
    )

    kfold = StratifiedKFold(n_splits=k, shuffle=True, random_state=random_state)

    fold_results = []

    for fold_idx, (train_idx, val_idx) in enumerate(kfold.split(X, y), 1):
        logger.info(f"\n{'='*60}")
        logger.info(f"Fold {fold_idx}/{k}")
        logger.info(f"{'='*60}")

        # Split data for this fold
        X_train, X_val = X[train_idx], X[val_idx]
        y_train, y_val = y[train_idx], y[val_idx]

        logger.info(
            f"Train size: {len(X_train)}, Validation size: {len(X_val)}"
        )

        classifier = classifier_class()
        classifier.train(X_train, y_train)

        # Make predictions
        y_pred, y_scores = classifier.predict(X_val)

        # Calculate accuracy
        accuracy = accuracy_score(y_val, y_pred)
        logger.success(f"Fold {fold_idx} Accuracy: {accuracy:.4f}")

        # Store results
        fold_results.append(
            {
                "fold": fold_idx,
                "accuracy": accuracy,
                "y_true": y_val,
                "y_pred": y_pred,
                "y_scores": y_scores,
            }
        )

    # Calculate average metrics
    accuracies = [result["accuracy"] for result in fold_results]
    avg_accuracy = np.mean(accuracies)
    std_accuracy = np.std(accuracies)

    logger.info(f"\n{'='*60}")
    logger.info("Cross Validation Results Summary")
    logger.info(f"{'='*60}")
    logger.success(
```

```
        f"Average Accuracy: {avg_accuracy:.4f} (+/- {std_accuracy:.4f})"
    )
    logger.info(
        f"Individual fold accuracies: {[f'{acc:.4f}' for acc in accuracies]}"
    )

    return {
        "fold_results": fold_results,
        "avg_accuracy": avg_accuracy,
        "std_accuracy": std_accuracy,
        "accuracies": accuracies,
    }


def _plot_confusion_matrix(
    cm: np.ndarray, dataset_name: str, classifier_name: str, results_dir: str
):
    """
    Plot and save confusion matrix heatmap.
    """
    try:
        plt.figure(figsize=(8, 6))

        num_classes = cm.shape[0]

        show_percent = True
        if num_classes > 10:
            show_percent = False

        cm_percent = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]

        annot_labels = (
            np.asarray(
                [
                    f"{val}\n({perc:.1%})" if show_percent else f"{val}"
                    for val, perc in zip(cm.flatten(), cm_percent.flatten())
                ]
            )
        ).reshape(cm.shape)

        sns.heatmap(cm, annot=annot_labels, fmt="", cmap="Blues", cbar=True)

        plt.xlabel("Predicted Label")
        plt.ylabel("True Label")
        plt.title(f"Confusion Matrix - {classifier_name} on {dataset_name}")

        os.makedirs(results_dir, exist_ok=True)
        save_path = os.path.join(
            results_dir, f"CM_{dataset_name}_{classifier_name}.png"
        )
        plt.savefig(save_path)
        logger.info(f"Confusion Matrix heatmap saved to {save_path}")
        plt.close()

    except Exception as e:
        logger.error("Failed to plot confusion matrix heatmap.")
        logger.exception(e)


def evaluate_classifier(
    y_true: np.ndarray,
    y_pred: np.ndarray,
    y_scores: np.ndarray,
    dataset_name: str,
    classifier_name: str,
```

```python
    results_dir: str = "results",
) -> None:
    """
    Evaluate classifier performance and generate reports.

    Args:
        y_true (np.ndarray): Ground truth labels
        y_pred (np.ndarray): Predicted labels from the model
        y_scores (np.ndarray): Model discriminant function values (e.g., probabilities)
        dataset_name (str): Dataset name (used for logs and filenames)
        classifier_name (str): Classifier name (used for logs and filenames)
        results_dir (str): Directory to save plots (e.g., ROC curve)
    """

    logger.info(
        f"--- Evaluation for [{classifier_name}] on [{dataset_name}] ---"
    )

    # calculate accuracy
    accuracy = accuracy_score(y_true, y_pred)
    logger.success(f"Accuracy: {accuracy:.4f}")

    try:
        logger.info("Confusion Matrix:")
        cm = confusion_matrix(y_true, y_pred)

        # _plot_confusion_matrix(cm, dataset_name, classifier_name, results_dir)

        logger.info(f"\n{cm}")

        logger.info("Classification Report:")
        report = classification_report(y_true, y_pred, zero_division=0)
        logger.info(f"\n{report}")

        num_classes = len(np.unique(y_true))

        if num_classes == 2:
            logger.info("Two-class dataset detected. Calculating ROC/AUC...")

            scores_for_roc = y_scores[:, 1]

            fpr, tpr, thresholds = roc_curve(y_true, scores_for_roc)

            roc_auc = auc(fpr, tpr)
            logger.success(f"Area Under Curve (AUC): {roc_auc:.4f}")

            plt.figure(figsize=(8, 6))
            plt.plot(
                fpr,
                tpr,
                color="darkorange",
                lw=2,
                label=f"ROC curve (area = {roc_auc:.4f})",
            )
            plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")
            plt.xlim([0.0, 1.0])
            plt.ylim([0.0, 1.05])
            plt.xlabel("False Positive Rate")
            plt.ylabel("True Positive Rate")
            plt.title(f"ROC Curve - {classifier_name} on {dataset_name}")
            plt.legend(loc="lower right")

            os.makedirs(results_dir, exist_ok=True)
            save_path = os.path.join(
                results_dir, f"ROC_{dataset_name}_{classifier_name}.png"
```

```
            )
            plt.savefig(save_path)
            logger.info(f"ROC curve saved to {save_path}")
            plt.close()

        else:
            logger.info(
                f"Skipping ROC/AUC calculation ({num_classes} classes detected)."
            )

    except Exception as e:
        logger.error("An error occurred during evaluation.")
        logger.exception(e)
        raise

    logger.info(f"--- End of Evaluation for [{classifier_name}] ---")
```

task1.py

```
import os
import argparse
from typing import Tuple
import numpy as np
import pandas as pd
from loguru import logger
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score,
    balanced_accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_auc_score,
)

from data_loader import load_dataset
from utils import k_fold_cross_validation, evaluate_classifier


class LDA:
    def __init__(self, n_components=None, priors=None):
        self.model = LinearDiscriminantAnalysis(n_components=n_components, priors=priors)
        logger.info("Initialized Linear Discriminant Analysis Classifier.")

    def fit(self, X: np.ndarray, y: np.ndarray) -> None:
        self.model.fit(X, y)
        logger.success("Model training completed.")

    def transform(self, X: np.ndarray) -> np.ndarray:
        return self.model.transform(X)

    def predict(self, X: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
        y_pred = self.model.predict(X)
        y_scores = self.model.predict_proba(X)
        return y_pred, y_scores


def compute_separability(X: np.ndarray, y: np.ndarray) -> float:
    """
    Compute the separability measure before projection using Fisher's criterion.

```

```
    Separability is computed as the ratio of between-class scatter to within-class scatter.
    For multi-class problems, we use the trace of S_B @ inv(S_W) where:
    - S_B is the between-class scatter matrix
    - S_W is the within-class scatter matrix

    Args:
        X: Feature matrix of shape (n_samples, n_features)
        y: Target labels of shape (n_samples,)

    Returns:
        separability: A scalar value representing the separability measure
    """
    classes = np.unique(y)
    n_features = X.shape[1]

    # Compute overall mean
    overall_mean = np.mean(X, axis=0)

    # Initialize scatter matrices
    S_W = np.zeros((n_features, n_features))  # Within-class scatter
    S_B = np.zeros((n_features, n_features))  # Between-class scatter

    for c in classes:
        X_c = X[y == c]
        n_c = X_c.shape[0]

        # Class mean
        mean_c = np.mean(X_c, axis=0)

        # Within-class scatter for class c
        S_W += (X_c - mean_c).T @ (X_c - mean_c)

        # Between-class scatter for class c
        mean_diff = (mean_c - overall_mean).reshape(-1, 1)
        S_B += n_c * (mean_diff @ mean_diff.T)

    # Compute separability as trace(S_B @ inv(S_W))
    # Add regularization to avoid singular matrix
    regularization = 1e-6
    S_W_reg = S_W + regularization * np.eye(n_features)

    try:
        separability = np.trace(np.linalg.solve(S_W_reg, S_B))
    except np.linalg.LinAlgError:
        logger.warning(
            "Could not compute separability due to singular matrix. Returning 0."
        )
        separability = 0.0

    return separability


def split_data(X: np.ndarray, y: np.ndarray, test_size=0.2, random_state=42):
    """
    Split the dataset into training and testing sets with stratification and feature scaling.
    """

    # Split the dataset
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=random_state
    )

    # Feature Scaling
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
```

```
    X_test = scaler.transform(X_test)
    logger.success("Feature scaling completed.\n")

    return X_train, X_test, y_train, y_test


def baseline(X: np.ndarray, y: np.ndarray, dataset: str):
    # split the dataset
    X_train, X_test, y_train, y_test = split_data(
        X, y, test_size=0.2, random_state=42
    )

    separability_before = compute_separability(X_train, y_train)

    # Apply LDA
    lda = LDA(n_components=None)
    lda.fit(X_train, y_train)
    X_train_lda = lda.transform(X_train)
    X_test_lda = lda.transform(X_test)

    separability_after = compute_separability(X_train_lda, y_train)

    y_pred, y_score = lda.predict(X_test)
    evaluate_classifier(
        y_true=y_test,
        y_pred=y_pred,
        y_scores=y_score,
        dataset_name=dataset,
        classifier_name="LDA",
        results_dir="results/task1/baseline",
    )

    logger.info(f"--- Task 1 ({dataset}) completed ---")
    logger.info("Summary:")
    logger.info(f"Separability before projection: {separability_before:.4f}")
    logger.info(f"Separability after projection: {separability_after:.4f}")
    logger.info("ROC/AUC plots saved to 'results/task1/baseline' dtirectory.")


def dimension_sweep(X: np.ndarray, y: np.ndarray, dataset: str):
    """
    Sweep LDA projection dimensions d = 1, ..., C-1 and
    record separability and test accuracy for each d.
    Results are saved to results/task1/dim_sweep/{dataset}_dim_sweep.csv
    """

    # split the dataset
    X_train, X_test, y_train, y_test = split_data(
        X, y, test_size=0.2, random_state=42
    )

    classes = np.unique(y_train)
    C = len(classes)
    max_dim = min(C - 1, X_train.shape[1])

    if max_dim < 1:
        logger.error(
            f"dim_sweep not applicable: C={C}, features={X_train.shape[1]}."
        )
        return

    logger.info(
        f"[{dataset}] dim_sweep: number of classes = {C}, "
        f"feature dim = {X_train.shape[1]}, max_dim = {max_dim}"
    )
```

```python

    results = []

    for d in range(1, max_dim + 1):
        lda = LDA(n_components=d)
        lda.fit(X_train, y_train)

        # Project to LDA subspace for computing separability
        X_train_lda = lda.transform(X_train)
        sep_d = compute_separability(X_train_lda, y_train)

        # Classification: use LDA's decision rule in the original (scaled) space
        y_pred, y_score = lda.predict(X_test)
        acc_d = accuracy_score(y_test, y_pred)

        logger.info(
            f"[{dataset}] dim={d}: separability={sep_d:.4f}, "
            f"accuracy={acc_d:.4f}"
        )

        results.append(
            {
                "dim": d,
                "separability": sep_d,
                "accuracy": acc_d,
            }
        )

    import os

    os.makedirs("results/task1/dim_sweep", exist_ok=True)
    df = pd.DataFrame(results)
    out_path = f"results/task1/dim_sweep/{dataset}_dim_sweep.csv"
    df.to_csv(out_path, index=False)

    logger.success(
        f"[{dataset}] dim_sweep finished. Results saved to '{out_path}'."
    )


def prior_experiment(X: np.ndarray, y: np.ndarray, dataset: str):
    """
    Compare LDA with empirical priors vs balanced priors on an imbalanced dataset
    (here, Taiwanese Bankruptcy).

    Results (accuracy, balanced accuracy, precision, recall, F1, AUC) are saved to
    results/task1/prior/{dataset}_prior.json
    """

    classes, counts = np.unique(y, return_counts=True)
    C = len(classes)
    if C != 2:
        logger.error(
            f"prior_experiment is designed for binary datasets, got C={C}."
        )
        return

    logger.info(
        f"[{dataset}] prior_experiment: classes={classes}, counts={counts}"
    )

    X_train, X_test, y_train, y_test = split_data(
        X, y, test_size=0.2, random_state=42
    )

```

```
    # 1. empirical prior
    lda_emp = LDA(n_components=None, priors=None)
    lda_emp.fit(X_train, y_train)
    y_pred_emp, y_score_emp = lda_emp.predict(X_test)

    # 2. balanced prior
    balanced_prior = np.ones(C) / C
    lda_bal = LDA(n_components=None, priors=balanced_prior)
    lda_bal.fit(X_train, y_train)
    y_pred_bal, y_score_bal = lda_bal.predict(X_test)

    metrics = {}

    metrics["empirical"] = {
        "accuracy": float(accuracy_score(y_test, y_pred_emp)),
        "balanced_accuracy": float(balanced_accuracy_score(y_test, y_pred_emp)),
        "precision": float(precision_score(y_test, y_pred_emp)),
        "recall": float(recall_score(y_test, y_pred_emp)),
        "f1": float(f1_score(y_test, y_pred_emp)),
        "auc": float(roc_auc_score(y_test, y_score_emp[:, 1])),
    }

    metrics["balanced"] = {
        "accuracy": float(accuracy_score(y_test, y_pred_bal)),
        "balanced_accuracy": float(balanced_accuracy_score(y_test, y_pred_bal)),
        "precision": float(precision_score(y_test, y_pred_bal)),
        "recall": float(recall_score(y_test, y_pred_bal)),
        "f1": float(f1_score(y_test, y_pred_bal)),
        "auc": float(roc_auc_score(y_test, y_score_bal[:, 1])),
    }

    # save results
    os.makedirs("results/task1/prior", exist_ok=True)
    out_path = f"results/task1/prior/{dataset}_prior.json"

    import json

    with open(out_path, "w") as f:
        json.dump(metrics, f, indent=2)

    logger.info(f"[{dataset}] prior_experiment metrics:\n{metrics}")
    logger.success(
        f"[{dataset}] prior_experiment finished. Results saved to '{out_path}'."
    )


if __name__ == "__main__":
    parser = argparse.ArgumentParser(
        description="Test Classifier with dimension reduction techniques"
    )
    parser.add_argument(
        "--dataset",
        type=str,
        required=True,
        choices=["telescope", "bankruptcy", "cover_type", "dry_bean"],
        help="Dataset to load (e.g., telescope, bankruptcy, cover_type, dry_bean)",
    )
    parser.add_argument(
        "--mode",
        type=str,
        required=True,
        choices=["baseline", "dim_sweep", "prior"],
        help="Experiment mode: baseline or dimension sweep or prior",
    )

```

```
    args = parser.parse_args()

    # Load dataset
    X, y = load_dataset(args.dataset)
    logger.info(
        f"Loaded {args.dataset} dataset: X shape={X.shape}, y shape={y.shape}\n"
    )

    # run the selected experiment mode
    if args.mode == "baseline":
        baseline(X, y, args.dataset)

    elif args.mode == "dim_sweep":
        dimension_sweep(X, y, args.dataset)

    elif args.mode == "prior":
        if args.dataset != "bankruptcy":
            logger.error(
                "Prior mode is only applicable to the bankruptcy dataset."
            )
            exit(1)

        prior_experiment(X, y, args.dataset)
```

task2.py

```
import argparse
from pathlib import Path
from typing import Tuple, Literal, Dict

import numpy as np
from loguru import logger
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, balanced_accuracy_score, f1_score

from data_loader import load_dataset


DatasetName = Literal["bankruptcy", "dry_bean"]


# Utils
def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(
        description="PR HW2 - Task 2: PCA experiments"
    )

    parser.add_argument(
        "--dataset",
        type=str,
        choices=["bankruptcy", "dry_bean"],
        required=True,
        help="Which dataset to run on.",
    )
    parser.add_argument(
        "--experiment",
        type=str,
        choices=[
            "explained_variance",  # Experiment 1
```

```
            "dim_sweep",  # Experiment 2
            "clf_compare",  # Experiment 3
        ],
        required=True,
        help="Which experiment to run.",
    )
    parser.add_argument(
        "--output-dir",
        type=str,
        default="results/task2",
        help="Directory to save CSVs / logs for the report.",
    )
    parser.add_argument(
        "--random-state",
        type=int,
        default=42,
        help="Random seed for reproducibility.",
    )
    parser.add_argument(
        "--test-size",
        type=float,
        default=0.2,
        help="Test set ratio (for train/val/test splitting).",
    )
    parser.add_argument(
        "--val-size",
        type=float,
        default=0.25,
        help="Validation ratio within train+val "
        "(e.g. 0.25 means train:val:test = 0.6:0.2:0.2).",
    )
    return parser.parse_args()


def ensure_output_dir(path_str: str) -> Path:
    out_dir = Path(path_str)
    out_dir.mkdir(parents=True, exist_ok=True)
    return out_dir


def split_and_scale(
    X: np.ndarray,
    y: np.ndarray,
    test_size: float,
    val_size: float,
    random_state: int,
) -> Tuple[
    np.ndarray, np.ndarray, np.ndarray, np.ndarray, np.ndarray, np.ndarray
]:
    """
    Split into train/val/test and apply StandardScaler fitted on train only.
    """
    X_trainval, X_test, y_trainval, y_test = train_test_split(
        X,
        y,
        test_size=test_size,
        random_state=random_state,
        stratify=y,
    )
    X_train, X_val, y_train, y_val = train_test_split(
        X_trainval,
        y_trainval,
        test_size=val_size,
        random_state=random_state,
        stratify=y_trainval,
```

```
    )

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)

    return X_train, X_val, X_test, y_train, y_val, y_test


def build_logreg(random_state: int) -> LogisticRegression:
    return LogisticRegression(
        max_iter=1000,
        random_state=random_state,
        n_jobs=-1,
    )


def build_rbf_svm(random_state: int) -> SVC:
    return SVC(
        kernel="rbf",
        probability=False,
        random_state=random_state,
    )


def evaluate_clf(
    clf,
    X_tr: np.ndarray,
    y_tr: np.ndarray,
    X_val: np.ndarray,
    y_val: np.ndarray,
) -> Dict[str, float]:
    clf.fit(X_tr, y_tr)
    y_pred = clf.predict(X_val)

    return {
        "acc": accuracy_score(y_val, y_pred),
        "balanced_acc": balanced_accuracy_score(y_val, y_pred),
        "f1_macro": f1_score(y_val, y_pred, average="macro"),
    }


# Experiment 1:
# Explained variance curve
def run_experiment_explained_variance(
    dataset: DatasetName,
    args: argparse.Namespace,
) -> None:
    logger.info(f"[Experiment 1] Explained variance on dataset = {dataset}")

    X, y = load_dataset(dataset)
    logger.info(
        f"Loaded dataset {dataset}: X shape={X.shape}, y shape={y.shape}"
    )

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    n_features = X_scaled.shape[1]
    pca = PCA(n_components=n_features)
    pca.fit(X_scaled)

    explained = pca.explained_variance_ratio_
    cumulative = np.cumsum(explained)
```

```python

    out_dir = ensure_output_dir(args.output_dir)
    out_csv = out_dir / f"explained_variance_{dataset}.csv"

    import csv

    with out_csv.open("w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["k", "explained_var_ratio", "cumulative_var_ratio"])
        for k, (e, c) in enumerate(zip(explained, cumulative), start=1):
            writer.writerow([k, e, c])

    logger.success(f"Saved explained variance curve to: {out_csv}")


# Experiment 2:
# Dimension sweep on Bankruptcy + LogReg
def run_experiment_dim_sweep(
    dataset: DatasetName,
    args: argparse.Namespace,
) -> None:
    if dataset != "bankruptcy":
        logger.warning(
            "Experiment 2 (dim_sweep) is mainly designed for the "
            "Taiwanese Bankruptcy dataset. You are running it on "
            f"{dataset}, which is fine but less emphasized in the report."
        )

    logger.info(f"[Experiment 2] PCA dimension sweep on dataset = {dataset}")

    X, y = load_dataset(dataset)
    logger.info(
        f"Loaded dataset {dataset}: X shape={X.shape}, y shape={y.shape}"
    )

    X_train, X_val, X_test, y_train, y_val, y_test = split_and_scale(
        X,
        y,
        test_size=args.test_size,
        val_size=args.val_size,
        random_state=args.random_state,
    )

    n_features = X_train.shape[1]

    candidate_dims = [2, 5, 10, 20, 30, 40, 60, 80]
    candidate_dims = [k for k in candidate_dims if k <= n_features]

    out_dir = ensure_output_dir(args.output_dir)
    out_csv = out_dir / f"dim_sweep_{dataset}.csv"

    import csv

    with out_csv.open("w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(
            [
                "k",
                "val_acc",
                "val_balanced_acc",
                "val_f1_macro",
                "test_acc",
                "test_balanced_acc",
                "test_f1_macro",
            ]
```

```
        )

        # Baseline: no PCA
        logger.info("Running baseline (no PCA)...")
        clf_base = build_logreg(args.random_state)
        metrics_val = evaluate_clf(clf_base, X_train, y_train, X_val, y_val)
        metrics_test = evaluate_clf(
            clf_base,
            np.vstack([X_train, X_val]),
            np.hstack([y_train, y_val]),
            X_test,
            y_test,
        )

        writer.writerow(
            [
                0,
                metrics_val["acc"],
                metrics_val["balanced_acc"],
                metrics_val["f1_macro"],
                metrics_test["acc"],
                metrics_test["balanced_acc"],
                metrics_test["f1_macro"],
            ]
        )

        logger.success(
            f"Baseline (no PCA) - "
            f"val_acc={metrics_val['acc']:.4f}, "
            f"val_balanced_acc={metrics_val['balanced_acc']:.4f}"
        )

        # PCA + LogReg for each k
        for k in candidate_dims:
            logger.info(f"Running PCA + LogReg with k={k}...")
            pca = PCA(n_components=k, random_state=args.random_state)
            X_train_pca = pca.fit_transform(X_train)
            X_val_pca = pca.transform(X_val)
            X_test_pca = pca.transform(X_test)

            clf = build_logreg(args.random_state)
            metrics_val = evaluate_clf(
                clf, X_train_pca, y_train, X_val_pca, y_val
            )
            metrics_test = evaluate_clf(
                clf,
                np.vstack([X_train_pca, X_val_pca]),
                np.hstack([y_train, y_val]),
                X_test_pca,
                y_test,
            )

            writer.writerow(
                [
                    k,
                    metrics_val["acc"],
                    metrics_val["balanced_acc"],
                    metrics_val["f1_macro"],
                    metrics_test["acc"],
                    metrics_test["balanced_acc"],
                    metrics_test["f1_macro"],
                ]
            )

            logger.success(
```

```
                f"k={k}: val_acc={metrics_val['acc']:.4f}, "
                f"val_balanced_acc={metrics_val['balanced_acc']:.4f}"
            )

    logger.success(f"Saved dimension sweep results to: {out_csv}")


# Experiment 3:
# Compare PCA effects across classifiers & datasets
def run_experiment_classifier_compare(
    dataset: DatasetName,
    args: argparse.Namespace,
) -> None:
    logger.info(f"[Experiment 3] Classifier comparison on dataset = {dataset}")

    X, y = load_dataset(dataset)
    logger.info(
        f"Loaded dataset {dataset}: X shape={X.shape}, y shape={y.shape}"
    )

    X_train, X_val, X_test, y_train, y_val, y_test = split_and_scale(
        X,
        y,
        test_size=args.test_size,
        val_size=args.val_size,
        random_state=args.random_state,
    )

    out_dir = ensure_output_dir(args.output_dir)
    out_csv = out_dir / f"classifier_compare_{dataset}.csv"

    import csv

    with out_csv.open("w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(
            [
                "classifier",
                "use_pca",
                "pca_dim",
                "val_acc",
                "val_balanced_acc",
                "val_f1_macro",
                "test_acc",
                "test_balanced_acc",
                "test_f1_macro",
            ]
        )

        # 1. Baseline: LogReg / RBF-SVM without PCA
        for clf_name in ["logreg", "rbf_svm"]:
            logger.info(f"Running baseline without PCA for {clf_name}...")

            if clf_name == "logreg":
                clf = build_logreg(args.random_state)
            else:
                clf = build_rbf_svm(args.random_state)

            metrics_val = evaluate_clf(clf, X_train, y_train, X_val, y_val)
            metrics_test = evaluate_clf(
                clf,
                np.vstack([X_train, X_val]),
                np.hstack([y_train, y_val]),
                X_test,
                y_test,
```

```
            )

            writer.writerow(
                [
                    clf_name,
                    0,
                    0,
                    metrics_val["acc"],
                    metrics_val["balanced_acc"],
                    metrics_val["f1_macro"],
                    metrics_test["acc"],
                    metrics_test["balanced_acc"],
                    metrics_test["f1_macro"],
                ]
            )

        # 2. PCA + classifier
        logger.info("Fitting PCA (95% variance) for classifier comparison...")
        pca = PCA(
            n_components=0.95, svd_solver="full", random_state=args.random_state
        )
        X_train_pca = pca.fit_transform(X_train)
        X_val_pca = pca.transform(X_val)
        X_test_pca = pca.transform(X_test)
        k_eff = X_train_pca.shape[1]

        logger.info(f"Effective PCA dimension for 95% variance: k={k_eff}")

        for clf_name in ["logreg", "rbf_svm"]:
            logger.info(f"Running {clf_name} with PCA (k={k_eff})...")

            if clf_name == "logreg":
                clf = build_logreg(args.random_state)
            else:
                clf = build_rbf_svm(args.random_state)

            metrics_val = evaluate_clf(
                clf, X_train_pca, y_train, X_val_pca, y_val
            )
            metrics_test = evaluate_clf(
                clf,
                np.vstack([X_train_pca, X_val_pca]),
                np.hstack([y_train, y_val]),
                X_test_pca,
                y_test,
            )

            writer.writerow(
                [
                    clf_name,
                    1,
                    k_eff,
                    metrics_val["acc"],
                    metrics_val["balanced_acc"],
                    metrics_val["f1_macro"],
                    metrics_test["acc"],
                    metrics_test["balanced_acc"],
                    metrics_test["f1_macro"],
                ]
            )

    logger.success(f"Saved classifier comparison results to: {out_csv}")


if __name__ == "__main__":
```

```
    args = parse_args()

    logger.info(
        f"Running Task 2 with dataset={args.dataset}, "
        f"experiment={args.experiment}"
    )

    if args.experiment == "explained_variance":
        run_experiment_explained_variance(args.dataset, args)
    elif args.experiment == "dim_sweep":
        run_experiment_dim_sweep(args.dataset, args)
    elif args.experiment == "clf_compare":
        run_experiment_classifier_compare(args.dataset, args)
    else:
        raise ValueError(f"Unknown experiment: {args.experiment}")
```

task2_plot.py

```
import pandas as pd
import matplotlib.pyplot as plt

bank = pd.read_csv("results/task2/explained_variance_bankruptcy.csv")
bean = pd.read_csv("results/task2/explained_variance_dry_bean.csv")

plt.figure(figsize=(8,5))
plt.plot(bank["k"], bank["cumulative_var_ratio"], label="Bankruptcy")
plt.plot(bean["k"], bean["cumulative_var_ratio"], label="Dry Bean")

plt.xlabel("Number of Principal Components $k$")
plt.ylabel("Cumulative Explained Variance Ratio")
plt.title("Explained Variance Curve (PCA)")
plt.grid(True, linestyle="--", alpha=0.4)
plt.legend()

plt.tight_layout()
# plt.savefig("figs/task2_explained_variance.pdf")
plt.savefig("results/task2/explained_variance.png", dpi=300)
```

task3.py

```
#!/usr/bin/env python
import argparse
from pathlib import Path
from typing import Literal

import numpy as np
from loguru import logger
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import Isomap
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, balanced_accuracy_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

from data_loader import load_dataset


DatasetName = Literal["bankruptcy", "dry_bean", "telescope", "cover_type"]


```

```python
# Utils
def parse_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(
        description="PR HW2 - Task 3: Nonlinear dimensionality reduction (Isomap) experiments"
    )

    parser.add_argument(
        "--dataset",
        type=str,
        choices=["bankruptcy", "dry_bean", "telescope", "cover_type"],
        default="dry_bean",
        help="Which dataset to run on (default: dry_bean).",
    )
    parser.add_argument(
        "--experiment",
        type=str,
        choices=[
            "embedding_viz",  # Experiment 1: 2D embedding visualization
            "dim_sweep_isomap",  # Experiment 2: Isomap dimension sweep (to be implemented)
            "neighbor_sensitivity",  # Experiment 3: n_neighbors sensitivity (to be implemented)
        ],
        required=True,
        help="Which experiment to run.",
    )
    parser.add_argument(
        "--output-dir",
        type=str,
        default="results/task3",
        help="Directory to save figures / CSVs.",
    )
    parser.add_argument(
        "--random-state",
        type=int,
        default=42,
        help="Random seed for reproducibility.",
    )
    parser.add_argument(
        "--n-neighbors",
        type=int,
        default=10,
        help="Number of neighbors for Isomap (default: 10).",
    )
    return parser.parse_args()


def ensure_output_dir(path_str: str) -> Path:
    out_dir = Path(path_str)
    out_dir.mkdir(parents=True, exist_ok=True)
    return out_dir


def split_and_scale(
    X: np.ndarray,
    y: np.ndarray,
    random_state: int,
):
    """
    Split into train/val/test and apply StandardScaler fitted on train only.
    We use the same 60/20/20 split protocol as in Task 2:
    - First split: train+val vs test = 0.8 / 0.2
    - Second split: train vs val = 0.75 / 0.25 of (train+val)
    """
    X_trainval, X_test, y_trainval, y_test = train_test_split(
        X,
        y,
```

```
        test_size=0.2,
        random_state=random_state,
        stratify=y,
    )

    X_train, X_val, y_train, y_val = train_test_split(
        X_trainval,
        y_trainval,
        test_size=0.25,  # 0.8 * 0.25 = 0.2  =>  train:val:test = 0.6:0.2:0.2
        random_state=random_state,
        stratify=y_trainval,
    )

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)

    return X_train, X_val, X_test, y_train, y_val, y_test


def build_logreg(random_state: int) -> LogisticRegression:
    return LogisticRegression(
        max_iter=1000,
        random_state=random_state,
        n_jobs=-1,
    )


def build_rbf_svm(random_state: int) -> SVC:
    return SVC(
        kernel="rbf",
        probability=False,
        random_state=random_state,
    )


def evaluate_clf(
    clf,
    X_tr: np.ndarray,
    y_tr: np.ndarray,
    X_val: np.ndarray,
    y_val: np.ndarray,
):
    clf.fit(X_tr, y_tr)
    y_pred = clf.predict(X_val)

    return {
        "acc": accuracy_score(y_val, y_pred),
        "balanced_acc": balanced_accuracy_score(y_val, y_pred),
        "f1_macro": f1_score(y_val, y_pred, average="macro"),
    }


# Experiment 1:
# 2D Embedding Visualization (PCA vs Isomap)
def run_experiment_embedding_viz(
    dataset: DatasetName,
    args: argparse.Namespace,
) -> None:
    """
    Experiment 1:
    Compare 2D embeddings produced by PCA and Isomap on a given dataset.
    This is mainly designed for Dry Bean, but can be run on other datasets as well.
    """
```

```
    if dataset != "dry_bean":
        logger.warning(
            "Experiment 1 (embedding_viz) is primarily designed for the Dry Bean dataset, "
            f"but you are running it on {dataset}."
        )

    logger.info(
        f"[Experiment 1] 2D embedding visualization on dataset = {dataset}"
    )

    # Load full dataset (no split; this is for visualization)
    X, y = load_dataset(dataset)
    logger.info(
        f"Loaded dataset {dataset}: X shape={X.shape}, y shape={y.shape}"
    )

    # Standardize features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # 2D PCA
    logger.info("Fitting PCA (2D)...")
    pca = PCA(n_components=2, random_state=args.random_state)
    X_pca = pca.fit_transform(X_scaled)

    # 2D Isomap
    logger.info(f"Fitting Isomap (2D) with n_neighbors={args.n_neighbors}...")
    iso = Isomap(
        n_components=2,
        n_neighbors=args.n_neighbors,
    )
    X_iso = iso.fit_transform(X_scaled)

    out_dir = ensure_output_dir(args.output_dir)

    # Save the embeddings as CSV for later plotting or analysis
    import csv

    pca_csv = out_dir / f"embedding_{dataset}_pca2.csv"
    iso_csv = out_dir / f"embedding_{dataset}_isomap2_k{args.n_neighbors}.csv"

    logger.info(f"Saving PCA 2D embedding to: {pca_csv}")
    with pca_csv.open("w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["z1", "z2", "label"])
        for (z1, z2), label in zip(X_pca, y):
            writer.writerow([z1, z2, label])

    logger.info(f"Saving Isomap 2D embedding to: {iso_csv}")
    with iso_csv.open("w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["z1", "z2", "label"])
        for (z1, z2), label in zip(X_iso, y):
            writer.writerow([z1, z2, label])

    # Optionally, generate a side-by-side scatter plot for report
    try:
        import matplotlib.pyplot as plt

        logger.info("Generating 2D scatter plots for PCA and Isomap...")

        classes = np.unique(y)

        fig, axes = plt.subplots(1, 2, figsize=(10, 4))

```

```
        # PCA plot
        ax = axes[0]
        for c in classes:
            idx = y == c
            ax.scatter(
                X_pca[idx, 0],
                X_pca[idx, 1],
                s=5,
                alpha=0.6,
                label=str(c),
            )
        ax.set_title("PCA (2D)")
        ax.set_xlabel("PC1")
        ax.set_ylabel("PC2")

        # Isomap plot
        ax = axes[1]
        for c in classes:
            idx = y == c
            ax.scatter(
                X_iso[idx, 0],
                X_iso[idx, 1],
                s=5,
                alpha=0.6,
                label=str(c),
            )
        ax.set_title(f"Isomap (2D, k={args.n_neighbors})")
        ax.set_xlabel("Component 1")
        ax.set_ylabel("Component 2")

        # Legend: 只在右圖畫一次，避免擠爆
        handles, labels = axes[1].get_legend_handles_labels()
        fig.legend(
            handles,
            labels,
            loc="upper center",
            ncol=min(len(classes), 6),
            fontsize=8,
        )

        plt.tight_layout(rect=[0, 0, 1, 0.90])

        fig_path_pdf = out_dir / f"task3_{dataset}_pca_isomap_2d.pdf"
        fig_path_png = out_dir / f"task3_{dataset}_pca_isomap_2d.png"

        fig.savefig(fig_path_pdf)
        fig.savefig(fig_path_png, dpi=300)

        plt.close(fig)

        logger.success(
            f"Saved 2D embedding figure to: {fig_path_pdf} and {fig_path_png}"
        )

    except ImportError:
        logger.warning(
            "matplotlib is not available; skipped figure generation. "
            "You can still use the saved CSV embeddings for plotting."
        )


# Experiment 2:
# Isomap Dimension Sweep
def run_experiment_dim_sweep_isomap(
    dataset: DatasetName,
```

```python
    args: argparse.Namespace,
) -> None:
    """
    Isomap dimension sweep experiment.

    - Split dataset into train/val/test using the same 60/20/20 protocol as Task 2.
    - For a range of Isomap dimensions (e.g., k  {2, 5, 10, 20, 30}),
      fit Isomap on the training data (after standardization) and
      evaluate logistic regression on val/test.
    - Save all metrics to CSV.
    """
    if dataset != "bankruptcy":
        logger.warning(
            "Experiment 'dim_sweep_isomap' is mainly designed for the "
            "Taiwanese Bankruptcy dataset, but you are running it on "
            f"{dataset}."
        )

    logger.info(f"[Experiment 2] Isomap dimension sweep on dataset = {dataset}")

    # 1. Load dataset
    X, y = load_dataset(dataset)
    logger.info(f"Loaded dataset {dataset}: X shape={X.shape}, y shape={y.shape}")

    # 2. Split + scale
    X_train, X_val, X_test, y_train, y_val, y_test = split_and_scale(
        X,
        y,
        random_state=args.random_state,
    )

    n_features = X_train.shape[1]

    # 3. Candidate Isomap dimensions (bounded by #features)
    candidate_dims = [2, 5, 10, 20, 30]
    candidate_dims = [k for k in candidate_dims if k <= n_features]

    out_dir = ensure_output_dir(args.output_dir)
    out_csv = out_dir / f"dim_sweep_isomap_{dataset}.csv"

    import csv

    with out_csv.open("w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(
            [
                "k",
                "val_acc",
                "val_balanced_acc",
                "val_f1_macro",
                "test_acc",
                "test_balanced_acc",
                "test_f1_macro",
            ]
        )

        # 4. Baseline: no dimensionality reduction (just logistic regression on scaled space)
        logger.info("Running baseline (no dimensionality reduction)...")
        clf_base = build_logreg(args.random_state)
        metrics_val = evaluate_clf(clf_base, X_train, y_train, X_val, y_val)
        metrics_test = evaluate_clf(
            clf_base,
            np.vstack([X_train, X_val]),
            np.hstack([y_train, y_val]),
            X_test,
```

```
            y_test,
        )

        writer.writerow(
            [
                0,
                metrics_val["acc"],
                metrics_val["balanced_acc"],
                metrics_val["f1_macro"],
                metrics_test["acc"],
                metrics_test["balanced_acc"],
                metrics_test["f1_macro"],
            ]
        )

        logger.success(
            "Baseline (no DR) - "
            f"val_bal_acc={metrics_val['balanced_acc']:.4f}, "
            f"test_bal_acc={metrics_test['balanced_acc']:.4f}"
        )

        # 5. For each Isomap dimension, fit Isomap + LogReg
        for k in candidate_dims:
            logger.info(
                f"Running Isomap + Logistic Regression with n_components={k}, "
                f"n_neighbors={args.n_neighbors}..."
            )

            iso = Isomap(
                n_components=k,
                n_neighbors=args.n_neighbors,
            )

            # Fit Isomap on training data only, then transform val/test
            X_train_iso = iso.fit_transform(X_train)
            X_val_iso = iso.transform(X_val)
            X_test_iso = iso.transform(X_test)

            clf = build_logreg(args.random_state)

            metrics_val = evaluate_clf(
                clf,
                X_train_iso,
                y_train,
                X_val_iso,
                y_val,
            )
            metrics_test = evaluate_clf(
                clf,
                np.vstack([X_train_iso, X_val_iso]),
                np.hstack([y_train, y_val]),
                X_test_iso,
                y_test,
            )

            writer.writerow(
                [
                    k,
                    metrics_val["acc"],
                    metrics_val["balanced_acc"],
                    metrics_val["f1_macro"],
                    metrics_test["acc"],
                    metrics_test["balanced_acc"],
                    metrics_test["f1_macro"],
                ]
```

```
            )

            logger.success(
                f"k={k}: val_bal_acc={metrics_val['balanced_acc']:.4f}, "
                f"test_bal_acc={metrics_test['balanced_acc']:.4f}"
            )

    logger.success(f"Saved Isomap dimension sweep results to: {out_csv}")


# Experiment 3:
def run_experiment_neighbor_sensitivity(
    dataset: DatasetName,
    args: argparse.Namespace,
) -> None:
    """
    Hyperparameter sensitivity experiment for Isomap.

    - Fix Isomap output dimension (n_components = 10).
    - Vary n_neighbors (e.g., {5, 10, 20, 50}).
    - Evaluate performance of an RBF SVM classifier on val/test for each setting.
    - Also include a baseline RBF SVM without Isomap.
    - Save all metrics to CSV.
    """
    logger.info(
        f"[Experiment 3] Isomap neighbor sensitivity on dataset = {dataset}"
    )

    # 1. Load dataset
    X, y = load_dataset(dataset)
    logger.info(f"Loaded dataset {dataset}: X shape={X.shape}, y shape={y.shape}")

    # 2. Split + scale (same 60/20/20 protocol)
    X_train, X_val, X_test, y_train, y_val, y_test = split_and_scale(
        X,
        y,
        random_state=args.random_state,
    )

    out_dir = ensure_output_dir(args.output_dir)
    out_csv = out_dir / f"neighbor_sensitivity_isomap_{dataset}.csv"

    # Choose a reasonable fixed output dimension for Isomap
    n_components = min(10, X_train.shape[1])

    # Candidate neighbor sizes
    candidate_neighbors = [5, 10, 20, 50]

    import csv

    with out_csv.open("w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(
            [
                "use_isomap",
                "n_components",
                "n_neighbors",
                "val_acc",
                "val_balanced_acc",
                "val_f1_macro",
                "test_acc",
                "test_balanced_acc",
                "test_f1_macro",
            ]
        )
```

```
477
478         # 3. Baseline: RBF SVM without Isomap
479         logger.info("Running baseline RBF SVM (no Isomap)...")
480         clf_base = build_rbf_svm(args.random_state)
481         metrics_val = evaluate_clf(clf_base, X_train, y_train, X_val, y_val)
482         metrics_test = evaluate_clf(
483             clf_base,
484             np.vstack([X_train, X_val]),
485             np.hstack([y_train, y_val]),
486             X_test,
487             y_test,
488         )
489
490         writer.writerow(
491             [
492                 0,          # use_isomap = 0
493                 0,          # n_components
494                 0,          # n_neighbors
495                 metrics_val["acc"],
496                 metrics_val["balanced_acc"],
497                 metrics_val["f1_macro"],
498                 metrics_test["acc"],
499                 metrics_test["balanced_acc"],
500                 metrics_test["f1_macro"],
501             ]
502         )
503
504         logger.success(
505             "Baseline RBF SVM - "
506             f"val_bal_acc={metrics_val['balanced_acc']:.4f}, "
507             f"test_bal_acc={metrics_test['balanced_acc']:.4f}"
508         )
509
510         # 4. Isomap + RBF SVM for each neighbor setting
511         for k_nn in candidate_neighbors:
512             logger.info(
513                 f"Running Isomap (n_components={n_components}, "
514                 f"n_neighbors={k_nn}) + RBF SVM..."
515             )
516
517             iso = Isomap(
518                 n_components=n_components,
519                 n_neighbors=k_nn,
520             )
521
522             # Fit Isomap on training data only
523             X_train_iso = iso.fit_transform(X_train)
524             X_val_iso = iso.transform(X_val)
525             X_test_iso = iso.transform(X_test)
526
527             clf = build_rbf_svm(args.random_state)
528
529             metrics_val = evaluate_clf(
530                 clf,
531                 X_train_iso,
532                 y_train,
533                 X_val_iso,
534                 y_val,
535             )
536             metrics_test = evaluate_clf(
537                 clf,
538                 np.vstack([X_train_iso, X_val_iso]),
539                 np.hstack([y_train, y_val]),
540                 X_test_iso,
541                 y_test,
```

```
            )

            writer.writerow(
                [
                    1,              # use_isomap = 1
                    n_components,   # n_components
                    k_nn,           # n_neighbors
                    metrics_val["acc"],
                    metrics_val["balanced_acc"],
                    metrics_val["f1_macro"],
                    metrics_test["acc"],
                    metrics_test["balanced_acc"],
                    metrics_test["f1_macro"],
                ]
            )

            logger.success(
                f"n_neighbors={k_nn}: "
                f"val_bal_acc={metrics_val['balanced_acc']:.4f}, "
                f"test_bal_acc={metrics_test['balanced_acc']:.4f}"
            )

    logger.success(
        f"Saved Isomap neighbor sensitivity results to: {out_csv}"
    )


if __name__ == "__main__":
    args = parse_args()

    logger.info(
        f"Running Task 3 with dataset={args.dataset}, "
        f"experiment={args.experiment}, "
        f"n_neighbors={args.n_neighbors}"
    )

    if args.experiment == "embedding_viz":
        run_experiment_embedding_viz(args.dataset, args)
    elif args.experiment == "dim_sweep_isomap":
        run_experiment_dim_sweep_isomap(args.dataset, args)
    elif args.experiment == "neighbor_sensitivity":
        run_experiment_neighbor_sensitivity(args.dataset, args)
    else:
        raise ValueError(f"Unknown experiment: {args.experiment}")
```