NYCU Pattern Recognition, Assignment #1

Student ID: 112550157, Name: 黃冠瑋

1 Methodology

Our implementation consists of several Python modules designed to fulfill the assignment requirements and enable a flexible experimental workflow.

1.1 Data Loading (data_loader.py)

We use the ucimlrepo package to retrieve the four main datasets from the UCI Machine Learning Repository: telescope, bankruptcy, dry bean, and cover type. To accelerate repeated data access, a caching mechanism is implemented via joblib.Memory, which stores preprocessed datasets locally.

The core function _preprocess_data() standardizes features and encodes categorical labels using sklearn.preprocessing.LabelEncoder. This ensures that all datasets are numerical and compatible with the implemented classifiers.

1.2 Classifier Module (classifier.py)

Following the assignment specification, each classifier is implemented as an independent Python class, including MLP, Naive Bayes, SVM, KNN, Random Forest, and XGBoost.

Each class defines two key methods:

- *train(X, y)* —fits the model using the given training data.
- *predict(X)* - outputs both the predicted class labels (y_pred) and discriminant function values (y_scores, e.g., class probabilities).

1.3 Evaluation Module (evaluation.py)

The evaluation process is implemented in the function evaluate_classifier(), which performs both quantitative and visual analyses.

- Confusion Matrix: Computed via sklearn.metrics.confusion_matrix and visualized as a heatmap using Seaborn, then saved as .png.
- Classification Report: Summarizes precision, recall, F1-score, and accuracy via sklearn.metrics.classification_report.
- ROC Curve and AUC: For binary classification tasks, the function automatically generates the ROC curve and computes the AUC (Area Under Curve), also saved as .png.

These metrics collectively assess both overall accuracy and class-level performance.

1.4 Experimental Workflow (main.py)

The main script coordinates dataset loading, model training, and evaluation. Before cross-validation, the dataset is divided into training and testing subsets using an 80/20 stratified split to preserve class balance.

The workflow then proceeds in two stages:

- **Cross Validation**: The function k_fold_cross_validation() performs 5-fold cross-validation to estimate model stability and prevent overfitting.
- **Final Evaluation**: After cross-validation, each model is retrained on the full dataset and re-evaluated on the same data to report its final performance metrics.

2 Experiments and Results

2.1 Experimental Setup

All experiments are conducted using Python 3.12 with scikit-learn and XGBoost. Each dataset is normalized and split with a 80/20 train-test ratio while maintaining class balance. Performance is measured by accuracy, confusion matrix, and ROC—AUC (for binary datasets). Each result represents the mean over 5-fold cross-validation.

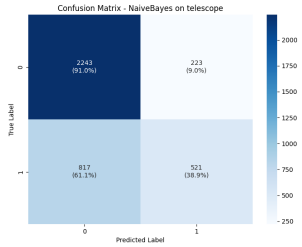
2.2 Dataset-wise Performance

2.2.1 MAGIC Gamma Telescope

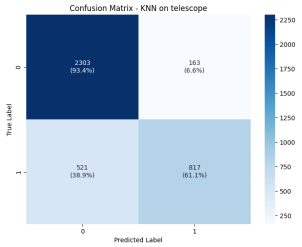
The MAGIC Gamma Telescope dataset is a binary classification task , used to classify high-energy gamma particles. It is a multivariate dataset containing 19020 instances, each described by 10 real-valued features. The performance of the classifiers on this dataset is summarized in Table 1.

Table 1: Telescope dataset: classification performance summary.

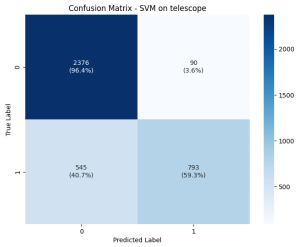
Classifier	**CV Accuracy**	**Accuracy (full)**	**AUC**
Naive Bayes	0.7257 ± 0.0103	0.73	0.7593
KNN	0.8013 ± 0.0044	0.82	0.8441
SVM	0.8198 ± 0.0067	0.83	0.8718
MLP	0.8189 ± 0.0222	0.84	0.8992
Random Forest	0.8761 ± 0.0029	0.89	0.9378
XGBoost	0.8745 ± 0.0030	0.88	0.9382



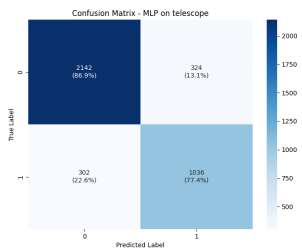
(a) Naive Bayes



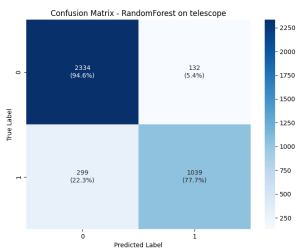
(b) KNN



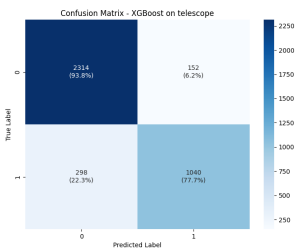
(c) SVM



(d) MLP

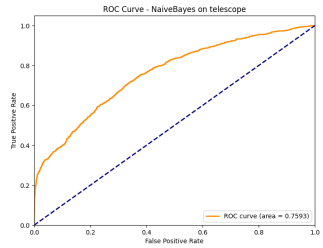


(e) Random Forest

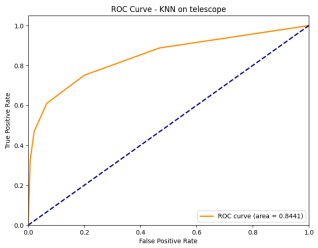


(f) XGBoost

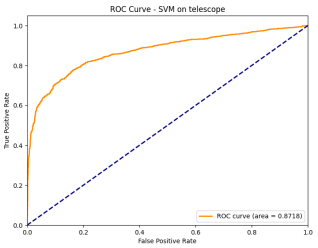
Figure 1: Confusion matrices for the Telescope dataset.



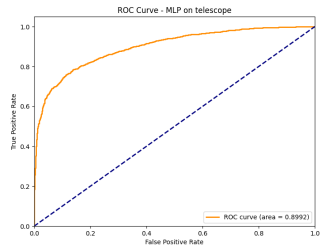
(a) Naive Bayes (AUC=0.7593)



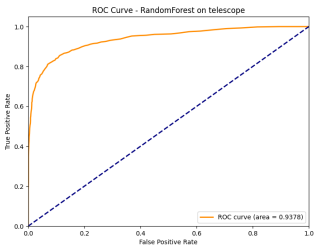
(b) KNN (AUC=0.8441)



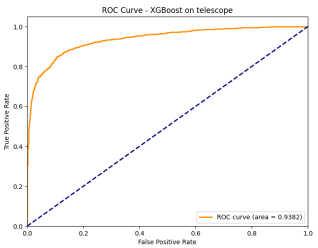
(c) SVM (AUC=0.8718)



(d) MLP (AUC=0.8992)



(e) Random Forest (AUC=0.9378)



(f) XGBoost (AUC=0.9382)

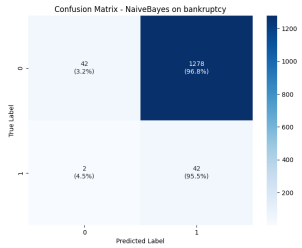
Figure 2: ROC curves and AUC scores for the Telescope dataset.

2.2.2 Taiwanese Bankruptcy Prediction

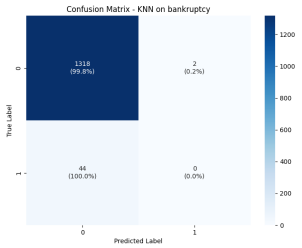
This dataset is a binary classification problem focused on business. It contains 6819 instances, each characterized by a high dimensionality of 95 integer features. The performance of the classifiers on this dataset, shown in Table 2, reveals significant discrepancies.

Table 2: Bankruptcy dataset: classification performance summary.

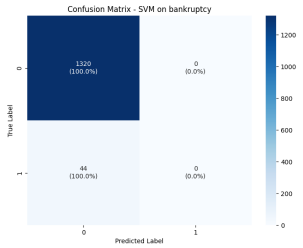
Classifier	**CV Accuracy**	**Accuracy (full)**	**AUC**
Naive Bayes	0.0631 ± 0.0069	0.06	0.6521
KNN	0.9665 ± 0.0012	0.97	0.5540
SVM	0.9677 ± 0.0004	0.97	0.5409
MLP	0.9419 ± 0.0171	0.94	0.5596
Random Forest	0.9696 ± 0.0023	0.97	0.9448
XGBoost	0.9696 ± 0.0051	0.97	0.9671



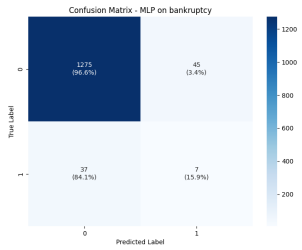
(a) Naive Bayes



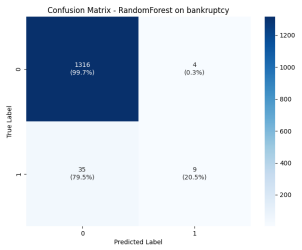
(b) KNN



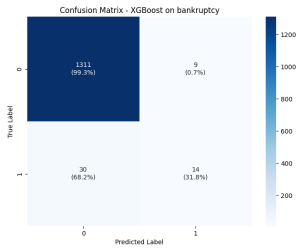
(c) SVM



(d) MLP

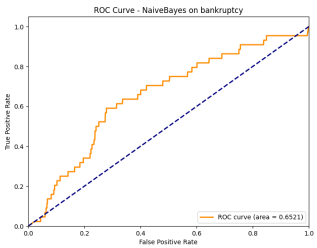


(e) Random Forest

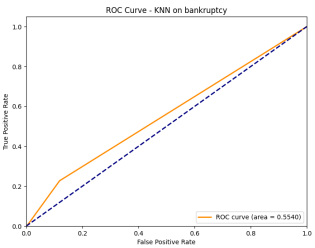


(f) XGBoost

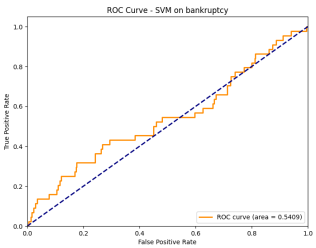
Figure 3: Confusion matrices for the bankruptcy dataset.



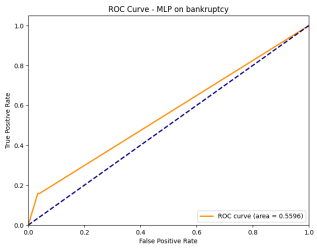
(a) Naive Bayes (AUC=0.7593)



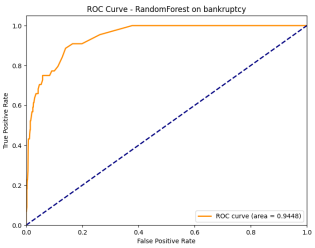
(b) KNN (AUC=0.8441)



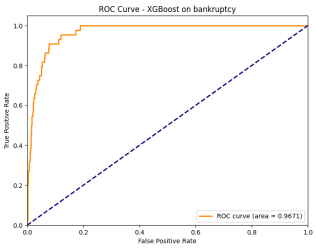
(c) SVM (AUC=0.8718)



(d) MLP (AUC=0.8992)



(e) Random Forest (AUC=0.9378)



(f) XGBoost (AUC=0.9382)

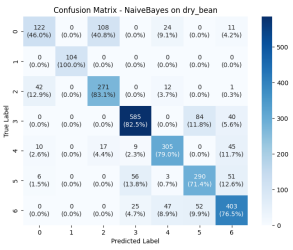
Figure 4: ROC curves and AUC scores for the bankruptcy dataset.

2.2.3 Dry Bean

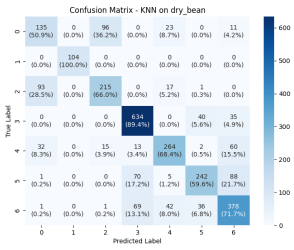
The Dry Bean dataset is a multi-class classification problem, with the goal of classifying 7 different types of registered dry beans. The dataset consists of 13611 instances, each described by 16 real-valued and integer features derived from images. Since this is a multi-class dataset, AUC is not a primary metric. The performance of the classifiers on this dataset is summarized in Table 3.

Table 3: Dry Bean dataset: classification performance summary.

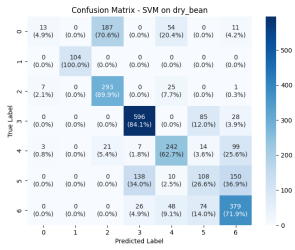
Classifier	**CV Accuracy**	**Accuracy (full)**
Naive Bayes	0.7639 ± 0.0055	0.76
KNN	0.7134 ± 0.0076	0.72
SVM	0.6336 ± 0.0019	0.64
MLP	0.4695 ± 0.1201	0.28
Random Forest	0.9232 ± 0.0054	0.92
XGBoost	0.9291 ± 0.0055	0.92



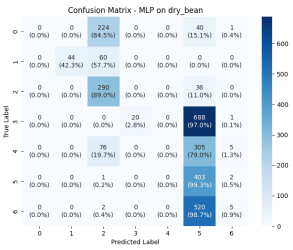
(a) Naive Bayes



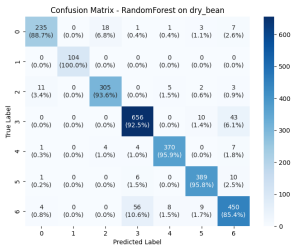
(b) KNN



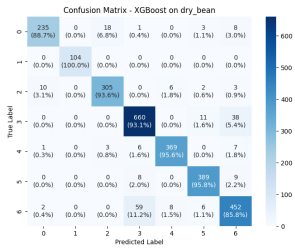
(c) SVM



(d) MLP



(e) Random Forest



(f) XGBoost

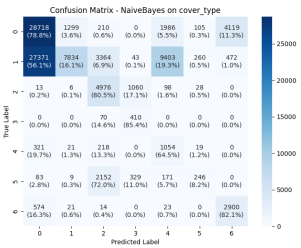
Figure 5: Confusion matrices for the dry bean dataset.

2.2.4 Cover Type

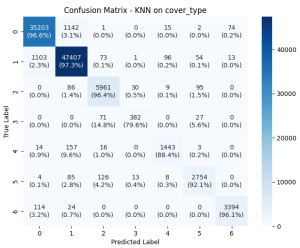
This is a large multi-class classification dataset with 7 distinct forest cover types. The full dataset has 581,012 instances and 54 features (both categorical and integer). Due to its size, we performed our experiments on a random subsample of 50000 instances, as implemented in data_loader.py. The performance of the classifiers on this dataset is summarized in Table 4.

Table 4: Cover Type dataset: classification performance summary.

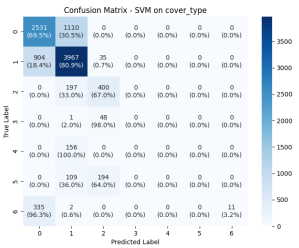
Classifier	**CV Accuracy**	**Accuracy (full)**
Naive Bayes	0.4579 ± 0.0007	0.46
KNN	0.9595 ± 0.0006	0.97
SVM	0.6957 ± 0.0041	0.69
MLP	0.7625 ± 0.0049	0.76
Random Forest	0.9466 ± 0.0012	0.95
XGBoost	0.8386 ± 0.0038	0.84



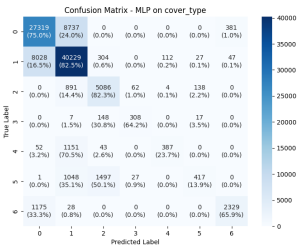
(a) Naive Bayes



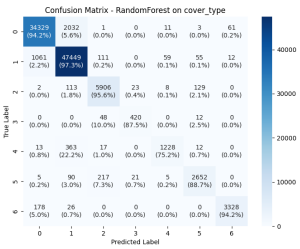
(b) KNN



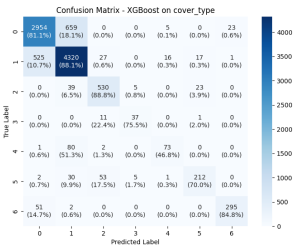
(c) SVM



(d) MLP



(e) Random Forest



(f) XGBoost

Figure 6: Confusion matrices for the cover-type dataset.

3 Analysis

After experimenting with six classifiers on four datasets, we observed several consistent patterns, as well as some interesting exceptions driven by specific data characteristics.

- **Overall, Ensemble Models (Random Forest and XGBoost) demonstrated the most robust and stable performance**: On the Telescope, Bankruptcy, and Dry Bean datasets, Random Forest and XGBoost significantly outperformed other models across nearly all metrics. This aligns with expectations, as these methods, by combining the predictions of many decision trees, can effectively handle high-dimensional features, reduce variance, and are insensitive to feature scaling. The low standard deviation on the Telescope dataset (std dev ≈ 0.003) and the high AUC on the Bankruptcy dataset (> 0.94) are clear testaments to their stability and accuracy.

- **The Bankruptcy dataset revealed the misleading nature of "Accuracy" on highly imbalanced data**: This was the most significant and unexpected finding. On the Bankruptcy dataset (Table 2), KNN and SVM achieved extremely high CV accuracy ($\approx 97\%$), yet their AUC scores were merely ≈ 0.55 (close to random guessing). A review of their confusion matrices (Figure 3) confirms that these models were predicting the majority class (class 0) almost exclusively. While this inflates the accuracy score, the models completely failed to identify the critical minority class (class 1). In contrast, Random Forest and XGBoost achieved the same 97

- **The performance of Naive Bayes was as expected, correlating strongly with its feature independence assumption**: Naive Bayes performed worst on datasets with a high number of features (Bankruptcy: 95 features, Covertype: 54 features) where feature correlations likely exist. This is expected, as its fundamental assumption of "conditional independence" is likely violated. However, on the Dry Bean dataset, which has fewer features (16), Naive Bayes (76.4% CV Acc) surprisingly outperformed KNN, SVM, and MLP, demonstrating its efficiency when its underlying assumptions are reasonably met.

- **The Covertype dataset was an exception, with KNN performing unexpectedly well**: On the large Covertype dataset (Table 4), KNN unexpectedly emerged as the top performer with $\approx 96\%$ CV accuracy, even surpassing Random Forest ($\approx 94.7\%$). This was surprising, as KNN is often susceptible to the "curse of dimensionality" with 54 features.

A possible explanation is that despite the high dimensionality, the large number of samples (500,000) makes the feature space ”dense” enough for instance-based logic to be highly effective, where a sample's nearest neighbors almost always belong to the same class.

- **MLP and SVM demonstrated sensitivity to data and hyperparameters**: The MLP was extremely unstable on the Dry Bean dataset (CV std dev 0.12) with very low final accuracy (28%). This is expected, as neural network models are highly sensitive to feature scaling, initialization, and hyperparameters (e.g., hidden layer size, learning rate), and can perform poorly without careful tuning. SVM struggled with the imbalanced Bankruptcy (AUC 0.54) and the large, multi-class Covertype datasets, suggesting its default parameters (likely an RBF kernel) were not optimal for these complex tasks.

4 Experiments: Extracting Features

4.1 Motivation

As noted in the main analysis (Section 2.3), the `Naive Bayes` classifier experienced a catastrophic failure on the 95-feature `Bankruptcy` dataset. Its performance (CV ACC 0.06) was far worse than all other models. We hypothesized this was not a failure of the model itself, but a failure caused by the high-dimensional, redundant features violating the model's core assumption of feature independence.

4.2 Method

We used `Random Forest`, one of the top-performing classifiers (AUC 0.9448), as a feature selection tool. We first trained a `Random Forest` model on the full 95-feature dataset and then extracted the `feature_importances_` attribute. We selected the indices of the top 20 most important features and created a new, reduced dataset (‘bankruptcy_top20‘). Finally, we retrained and re-evaluated the `Naive Bayes` classifier using only this 20-feature subset.

4.3 Results and Analysis

The results were dramatic and confirmed our hypothesis. When trained on the reduced 20-feature subset, the `Naive Bayes` classifier's performance surged across all key metrics.

Table 5 provides a direct comparison of the model's performance before and after feature selection. The Area Under Curve (AUC), the most reliable metric for this imbalanced dataset, improved from a near-random 0.6521 to a strong 0.9027.

Table 5: Performance comparison of Naive Bayes on the Bankruptcy dataset.

Naive Bayes Model	**CV Accuracy**	**Accuracy (full)**	**AUC**
Full 95 Features	0.0631 ± 0.0069	0.06	0.6521
Top 20 (from RF)	0.7846 ± 0.3655	0.97	0.9027

This experiment strongly indicates that the model itself is effective, but it was crippled by the 75 redundant or noisy features in the original dataset. This highlights the sensitivity of `Naive Bayes` to its assumptions and demonstrates the utility of ensemble methods as powerful tools for feature selection.

5 Conclusion

This project's experiments revealed two critical insights beyond simple classifier benchmarking. First, the `Bankruptcy` dataset experiments proved that for imbalanced data, accuracy is a misleading metric, and AUC is a far more reliable measure of a model's true predictive skill. Second, our feature selection experiment confirmed that a model's performance is fundamentally tied to its underlying assumptions. By reducing 95 redundant features to the top 20, the `Naive Bayes` classifier's AUC surged from 0.6521 to 0.9027, validating this principle. These results show that successful classification depends not just on algorithm choice but on the critical evaluation of metrics and feature relevance.

A Program Listing

A.1 classifier.py

```python
import numpy as np
from loguru import logger
from typing import Tuple

from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier

class MLP:
    def __init__(self):
        self.model = MLPClassifier()
        logger.info("Initialized Multi-layer Perceptron Classifier.")

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

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

class NaiveBayes:
    def __init__(self):
        self.model = GaussianNB()
        logger.info("Initialized Gaussian Naive Bayes Classifier.")

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

```

```
36      def predict(self, X: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
37          y_pred = self.model.predict(X)
38          y_scores = self.model.predict_proba(X)
39          logger.info("Prediction completed.")
40          return y_pred, y_scores
41
42  class KNN:
43      def __init__(self):
44          self.model = KNeighborsClassifier()
45          logger.info("Initialized K-Nearest Neighbors Classifier.")
46
47      def train(self, X: np.ndarray, y: np.ndarray) -> None:
48          self.model.fit(X, y)
49          logger.info("Model training completed.")
50
51      def predict(self, X: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
52          y_pred = self.model.predict(X)
53          y_scores = self.model.predict_proba(X)
54          logger.info("Prediction completed.")
55          return y_pred, y_scores
56
57
58  class RandomForest:
59      def __init__(self):
60          self.model = RandomForestClassifier()
61          logger.info("Initialized Random Forest Classifier.")
62
63      def train(self, X: np.ndarray, y: np.ndarray) -> None:
64          self.model.fit(X, y)
65          logger.info("Model training completed.")
66
67      def predict(self, X: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
68          y_pred = self.model.predict(X)
69          y_scores = self.model.predict_proba(X)
70          logger.info("Prediction completed.")
71          return y_pred, y_scores
72
73
74  class SVM:
```

```
    def __init__(self):
        self.model = SVC(probability=True)
        logger.info("Initialized Support Vector Machine Classifier.")

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

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


class XGBoost:
    def __init__(self):
        self.model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
        logger.info("Initialized XGBoost Classifier.")

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

    def predict(self, X: np.ndarray) -> Tuple[np.ndarray, np.ndarray]:
        y_pred = self.model.predict(X)
        y_scores = self.model.predict_proba(X)
        logger.info("Prediction completed.")
        return y_pred, y_scores
```

A.2 data_loader.py

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

```

```python
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(5E5), 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)
```

A.3 evaluation.py

```
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc
from loguru import logger
import matplotlib.pyplot as plt
import seaborn as sns
import os
from typing import Optional

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)

```

```python

def evaluate_classifier(y_true: np.ndarray,
                        y_pred: np.ndarray,
                        y_scores: np.ndarray,
                        dataset_name: str,
                        classifier_name: str,
                        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}] ---")

    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]
```

```
95
96              fpr, tpr, thresholds = roc_curve(y_true, scores_for_roc)
97
98              roc_auc = auc(fpr, tpr)
99              logger.success(f"Area Under Curve (AUC): {roc_auc:.4f}")
100
101             plt.figure(figsize=(8, 6))
102             plt.plot(fpr, tpr, color='darkorange', lw=2,
103                      label=f'ROC curve (area = {roc_auc:.4f})')
104             plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
105             plt.xlim([0.0, 1.0])
106             plt.ylim([0.0, 1.05])
107             plt.xlabel('False Positive Rate')
108             plt.ylabel('True Positive Rate')
109             plt.title(f'ROC Curve - {classifier_name} on {dataset_name}')
110             plt.legend(loc="lower right")
111
112             os.makedirs(results_dir, exist_ok=True)
113             save_path = os.path.join(results_dir, f"ROC_{dataset_name}_{classifier_name}.png")
114             plt.savefig(save_path)
115             logger.info(f"ROC curve saved to {save_path}")
116             plt.close()
117
118         else:
119             logger.info(f"Skipping ROC/AUC calculation ({num_classes} classes detected).")
120
121     except Exception as e:
122         logger.error("An error occurred during evaluation.")
123         logger.exception(e)
124         raise
125
126     logger.info(f"--- End of Evaluation for [{classifier_name}] ---")
```

A.4 main.py

```
1  import argparse
2  import numpy as np
3  from loguru import logger
```

```python
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score
import sys

from data_loader import load_dataset
from evaluation import evaluate_classifier

from classifier import MLP, NaiveBayes, KNN, RandomForest, SVM, XGBoost

CLASSIFIER_MAP = {
    'MLP': MLP,
    'NaiveBayes': NaiveBayes,
    'KNN': KNN,
    'RandomForest': RandomForest,
    'SVM': SVM,
    'XGBoost': XGBoost
}


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 = KFold(n_splits=k, shuffle=True, random_state=random_state)

    fold_results = []

```

```
    for fold_idx, (train_idx, val_idx) in enumerate(kfold.split(X), 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
    }


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Test Classifier with k-fold cross validation')
    parser.add_argument('--model', type=str, required=True,
                        help=f'Classifier to use. Available: {list(CLASSIFIER_MAP.keys())}')
    parser.add_argument('--dataset', type=str, required=True,
                        help='Dataset to load (e.g., iris, breast_cancer, bank_note, glass)')
    parser.add_argument('--k', type=int, default=5,
                        help='Number of folds for cross validation (default: 5)')
    parser.add_argument('--random_state', type=int, default=42,
                        help='Random seed for reproducibility (default: 42)')
    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}")

    classifier_class = CLASSIFIER_MAP.get(args.model)
    if classifier_class is None:
        logger.error(f"Unknown model: '{args.model}'")
        logger.info(f"Available models: {list(CLASSIFIER_MAP.keys())}")
        sys.exit(1)

    logger.info(f"Selected model: {args.model}")

    cv_results = k_fold_cross_validation(
        classifier_class, X, y, k=args.k, random_state=args.random_state
    )

```

```
    logger.info(f"\n{'='*60}")
    logger.info("Training final model on entire dataset for evaluation...")
    logger.info(f"{'='*60}")

    classifier = classifier_class()
    classifier.train(X, y)
    y_pred_final, y_scores_final = classifier.predict(X)

    evaluate_classifier(
        y_true=y,
        y_pred=y_pred_final,
        y_scores=y_scores_final,
        dataset_name=args.dataset,
        classifier_name=args.model
    )
```

A.5 subset.py

```
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from loguru import logger

from data_loader import load_dataset
from classifier import NaiveBayes
from main import k_fold_cross_validation, evaluate_classifier


if __name__ == "__main__":
    dataset = "bankruptcy"

    model = RandomForestClassifier()
    logger.info("Initialized Random Forest Classifier.")

    X, y = load_dataset(dataset)
    model.fit(X, y)

    importances = model.feature_importances_
    top_20_indices = np.argsort(importances)[::-1][:20]
```

```
21
22      logger.info(f"Top 20 most important feature indices: {top_20_indices}")
23
24      X_top20 = X[:, top_20_indices]
25      logger.info(f"Created new dataset with top 20 features: X shape={X_top20.shape}")
26
27      logger.info("Running K-fold CV for NaiveBayes on TOP 20 FEATURES...")
28      nb_top20_cv_results = k_fold_cross_validation(
29          NaiveBayes, X_top20, y
30      )
31
32      logger.info("Evaluating final NaiveBayes model on TOP 20 FEATURES...")
33      nb_top20_final = NaiveBayes()
34      nb_top20_final.train(X_top20, y)
35      y_pred_nb_top20, y_scores_nb_top20 = nb_top20_final.predict(X_top20)
36
37      evaluate_classifier(
38          y_true=y,
39          y_pred=y_pred_nb_top20,
40          y_scores=y_scores_nb_top20,
41          dataset_name="bankruptcy_top20",
42          classifier_name="NaiveBayes"
43      )
```