110612117 張仲瑜 HW2 report

I. Introduction

1. Goal

In this task, we aim to observe and compare the performance after performing the Fisher Linear Discriminant (FLD) and Principal Component Analysis (PCA), and how they affect Gaussian Naïve Bayes (GNB) classifier and Multinomial Logistic Regression (MLR) classifier.

2. Core Idea

I implemented FLD and PCA module on top of HW1 pipeline, and conducted several experiments to observe how dimensional reduction affects performance.

3. Dataset

- ✧ Obesity level dataset (7 classes with 16 features, #2111)
- ✧ Red wine quality dataset (6 classes with 11 features, #1599)
- ✧ Forest fire happen or not dataset
  (2 classes with 12 features, #244)
- ✧ People churning or not dataset
  (2 classes with 13 features, #3150)

II. Method

1. Load data and preprocessing

Since there exists non-numerical data in datasets, I map those features into numerical type to be accepted by classifiers.

2. Split data to training, validation, and testing

I divided 10% of the data as test dataset, while the other 90% of the data served as the train/valid dataset. In MLR, I use the cross validation (1:9) inside the train/valid dataset to get the average F1 score of the classifier and plot the result of the testing data. Since there's no hyperparameters requiring adjustment, I opted to use the whole 90% train/validation dataset for training without cross-validation. Data stratification is also applied to make sure they

have similar distribution.

3. Implement the FLD and PCA module [1]

   Implement two dimensionality reduction methods. PCA aims to maximize the data variance after projection, while FLD aims to maximize the between-class scatter to within-class scatter ratio.

4. Dimension reduction with FLD and visualize the distribution on 1, 2, 3 and 4 dimensions [2]

5. Performance after PCA under certain variance preservation (HW1 + HW2 experiment) [3]

   In this experiment, I chose the retained dimension as the lowest number of dimensions that can retain 25%, 50%. 75% and 90% of total variance and observe their performance.

6. Performance with or without PCA then Feature Selection (HW1 + HW2 experiment) [4]

   In this experiment, I performed FLD after PCA (90%) to see whether I can further boost the performance.

7. Metrices computation and figure plotting [5]

   Compute the confusion matrix, accuracy, precision, recall, F1, AUC, then plot the class and micro-ROC. I also plot the comparison plot for experiments.

III. Experiment Result and Analysis

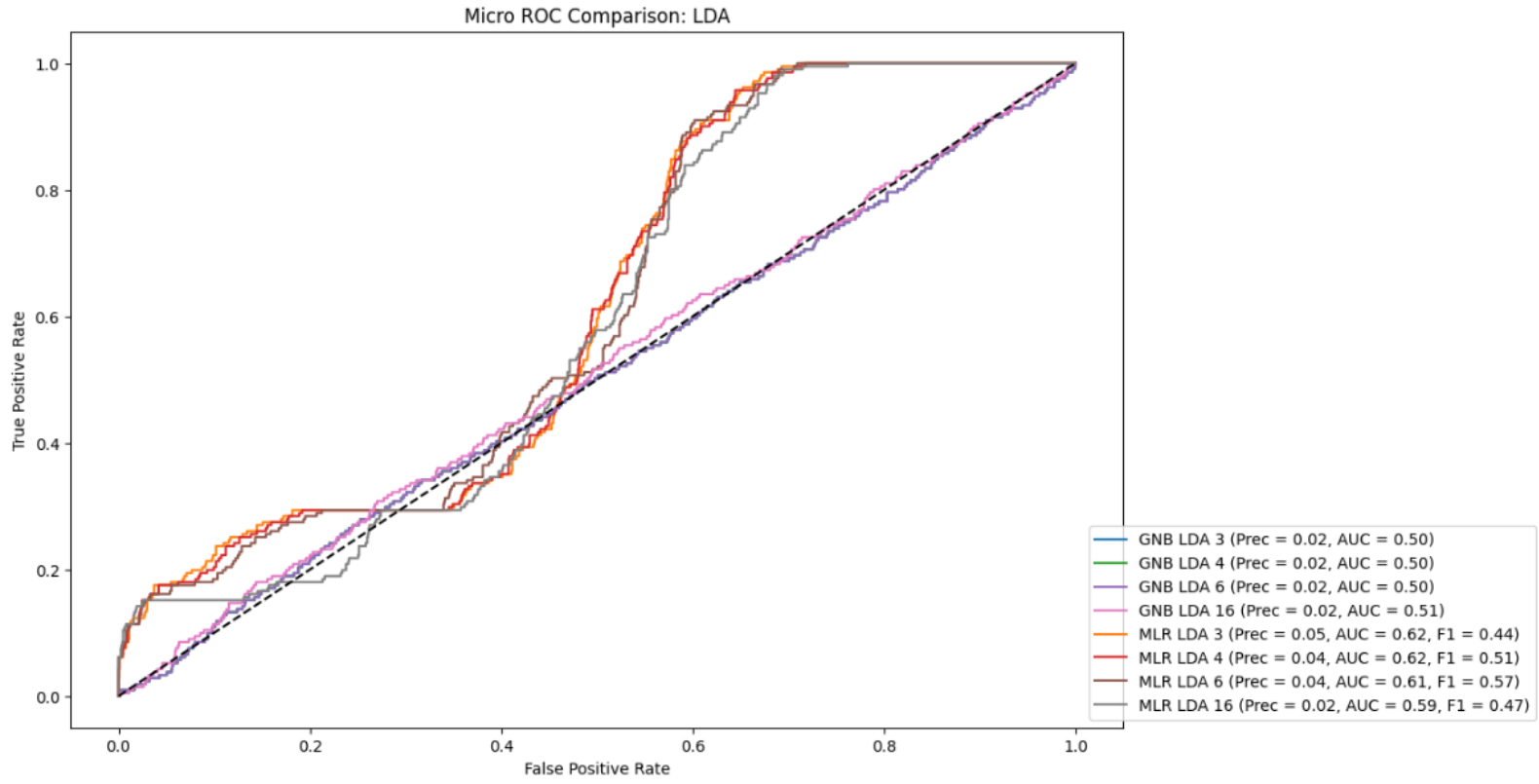
Note:

- ✧ I use macro precision to observe the effect of data imbalance, it's normal that it is small if one of the classes gets 0 prediction.

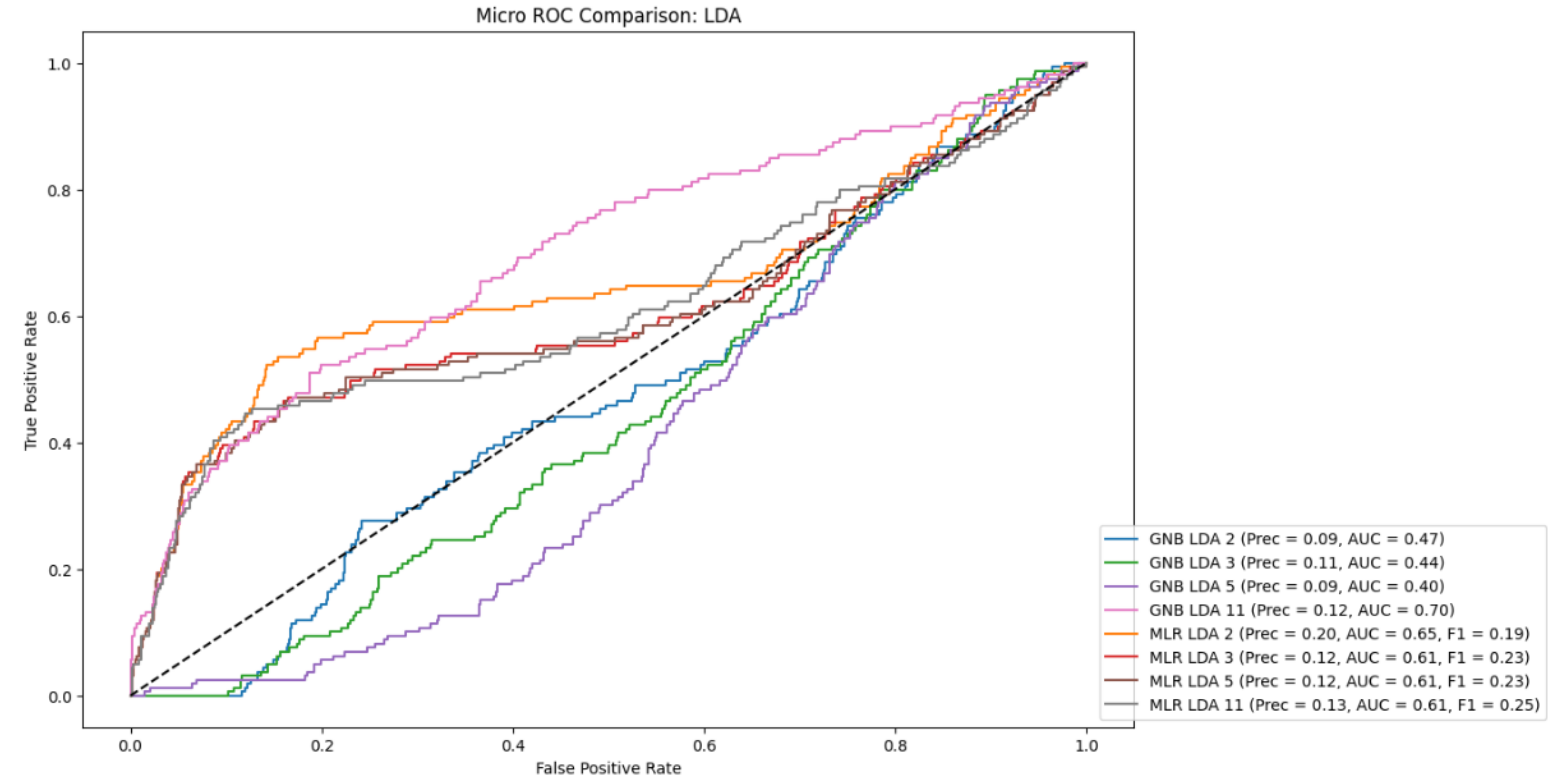
- ✧ In the label, GNB/MLB with the largest number means the original one without projection

1. FLD
   - ✧ Result:

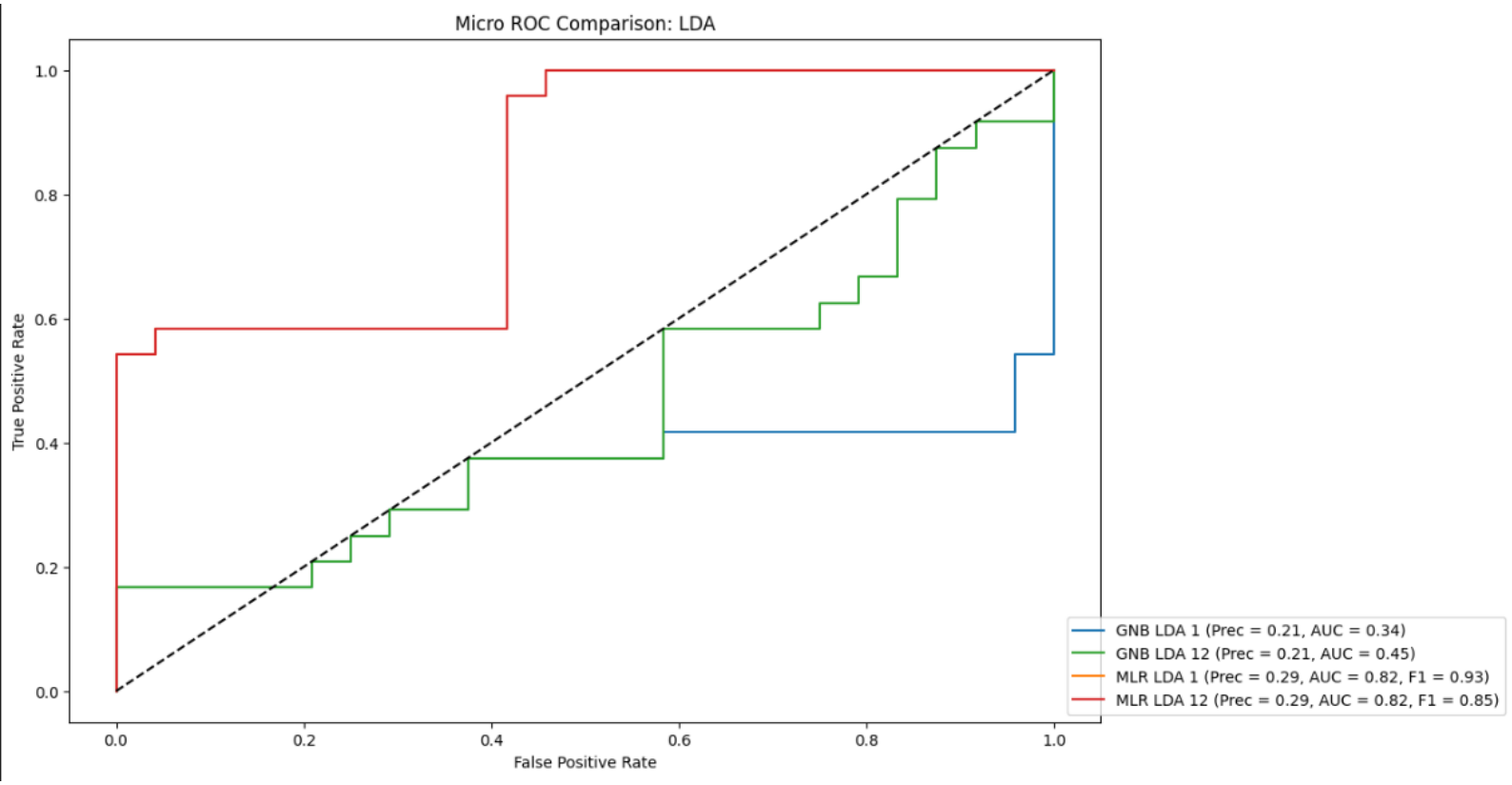
Obesity:



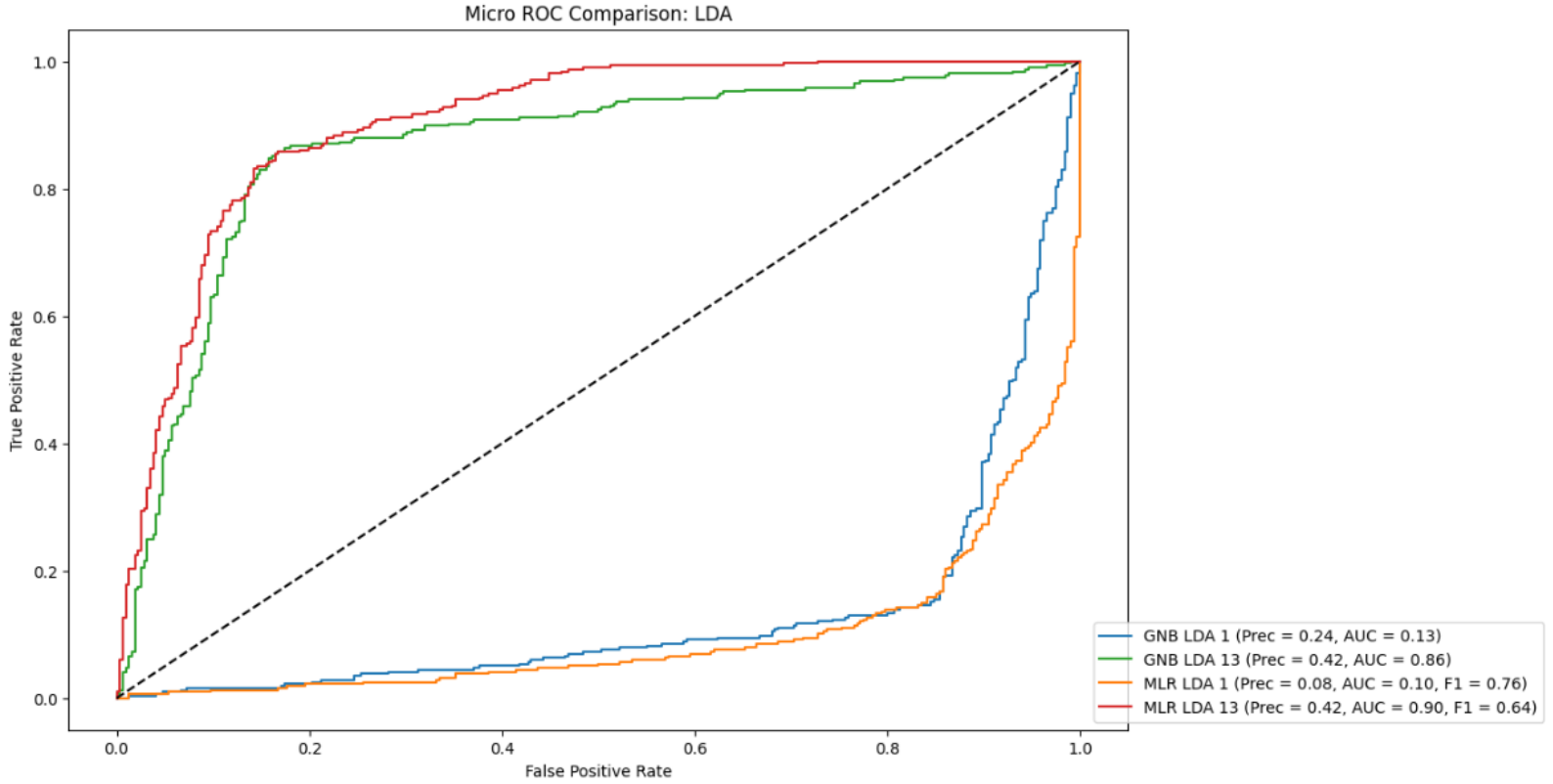
Wine:



Forest:



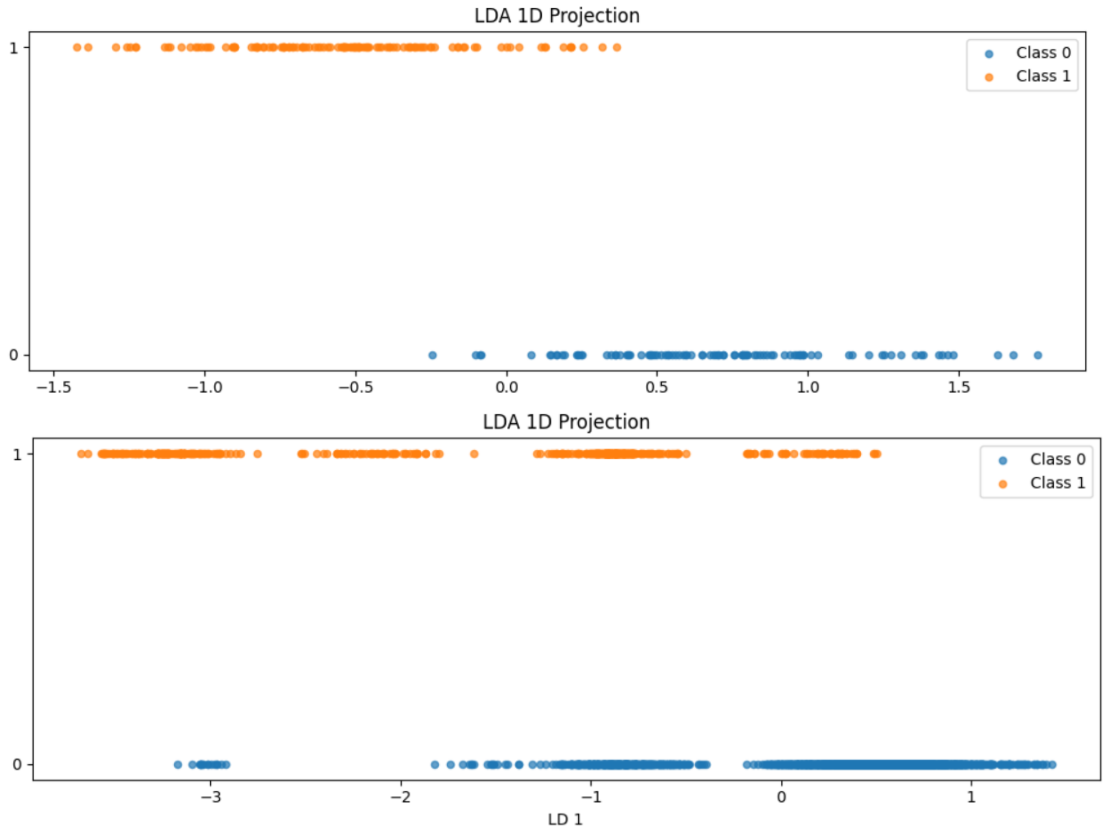
Churn:



- Analysis:
  - FLD causes harm to GNB classifiers in multi-class datasets, Since FLD performs a global linear transformation and mixes features to create new discriminant directions, it introduces strong dependencies between dimensions, while GNB relies on the assumption that features are conditionally independent, the dependencies may violate the assumption. Thus, the AUCs drop in most cases.

  - For MLR classifier, it benefits from discriminative features aligned with class boundaries. The AUCs and macro-precision increases in most multi-class datasets.

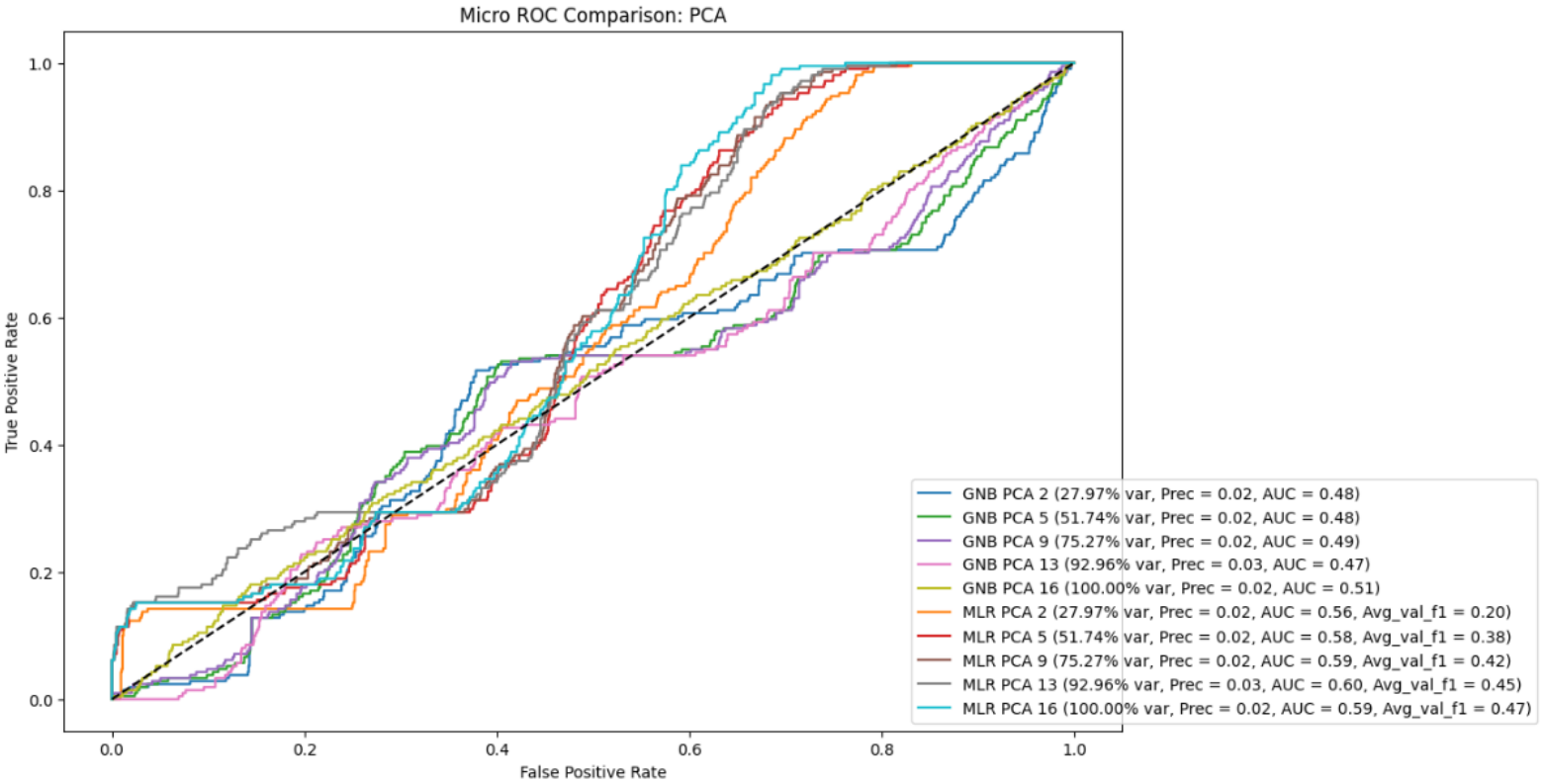
  - For two-class datasets, since FLD can at most project original data to class-1 dimensions, it's reasonable that performances degrade. However, for Forest dataset, it turns out that the data can be well separated after the projection to one dimension.



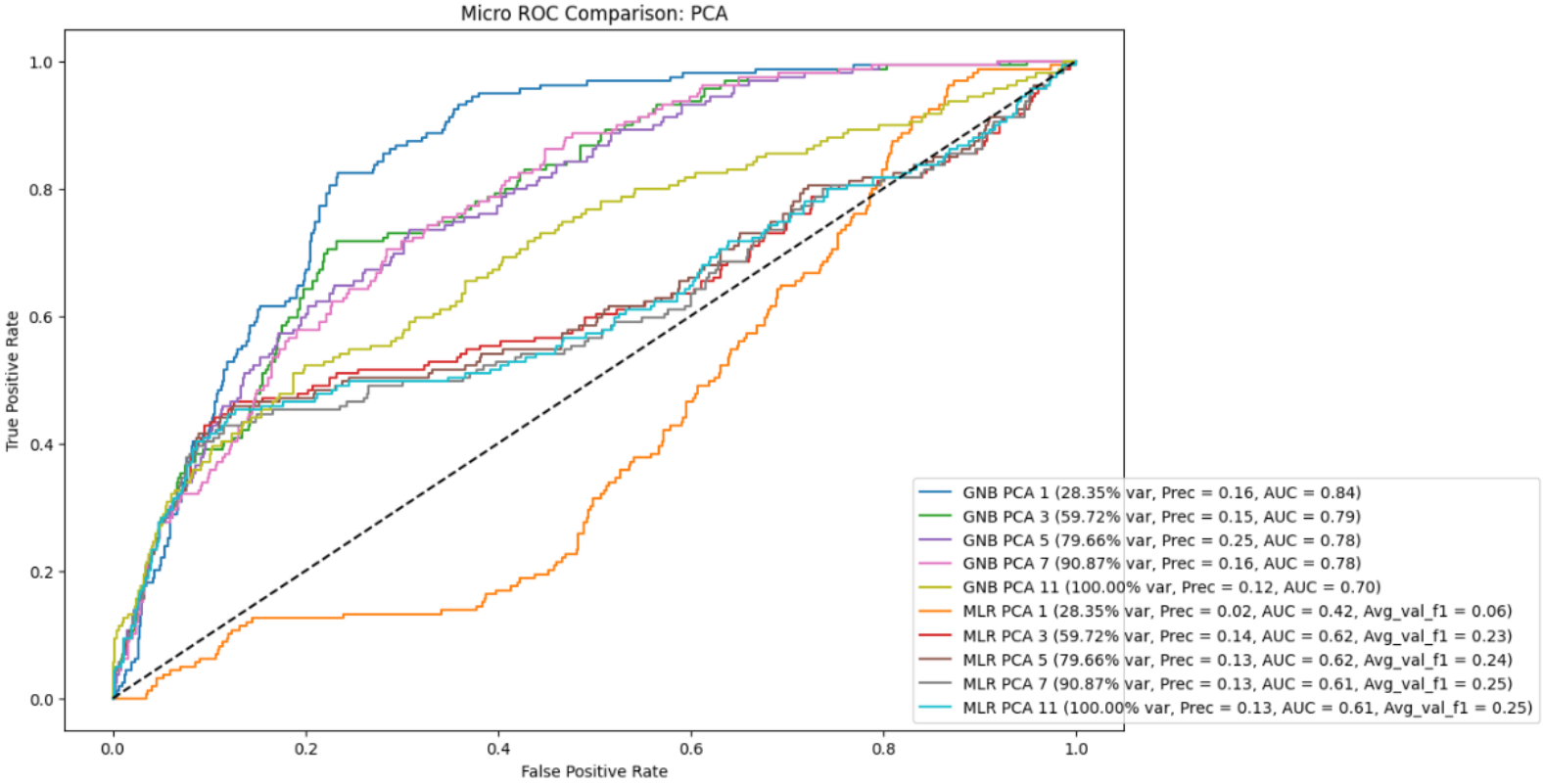
  - Forest (Above), Churn (Below)

2. Primary Component Analysis
   - ✧ Result

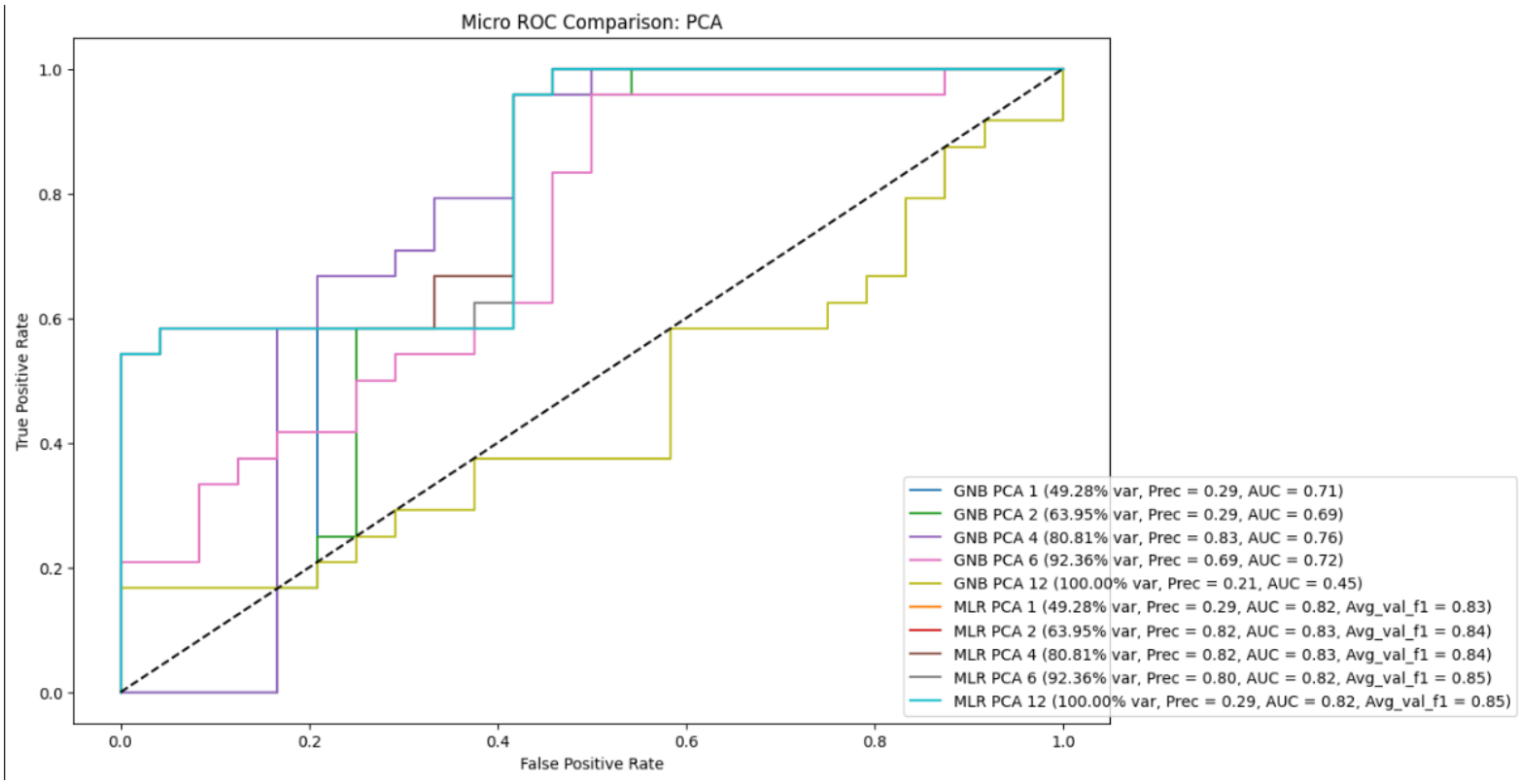
Obesity:



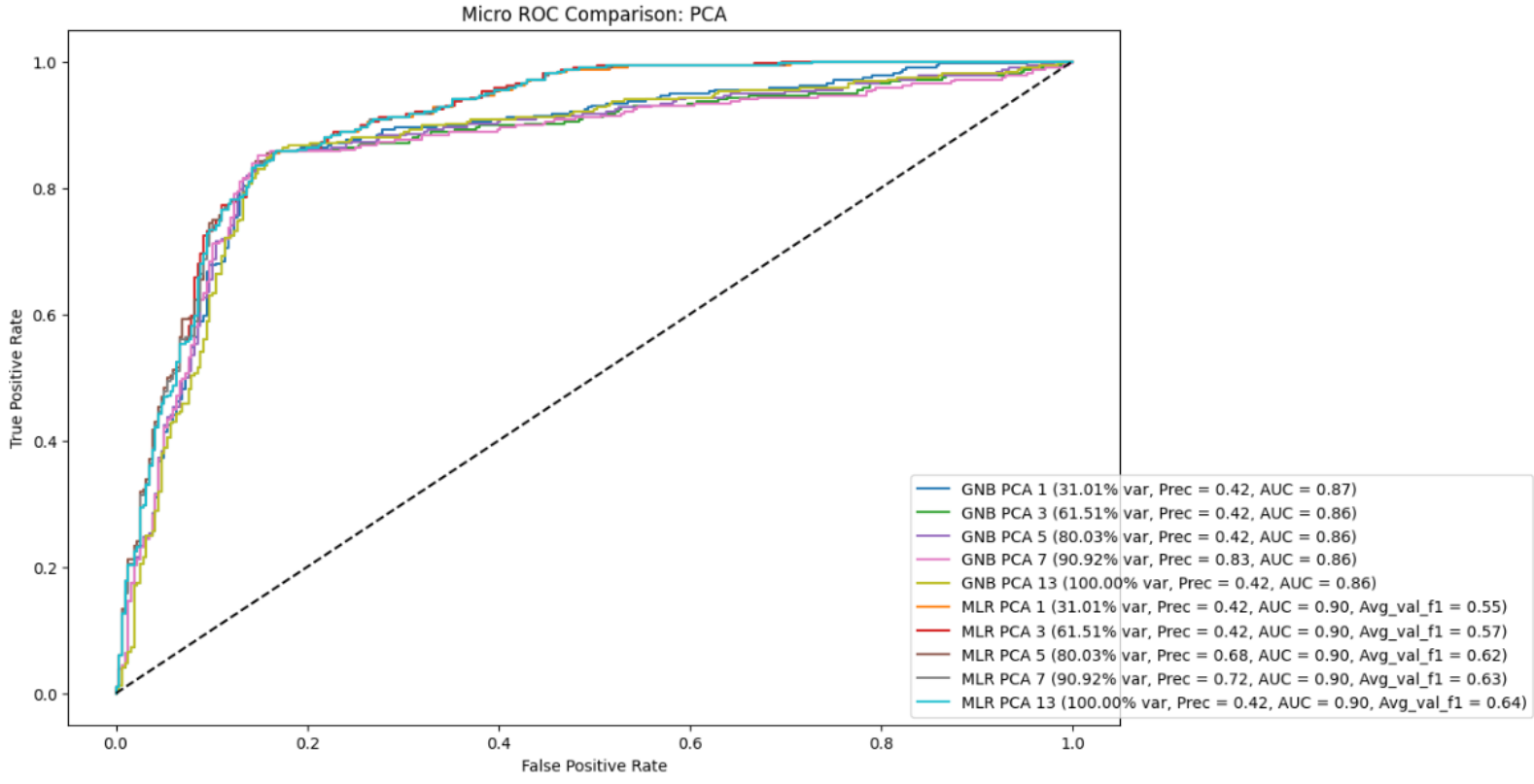
Wine:



Forest:



Churn:

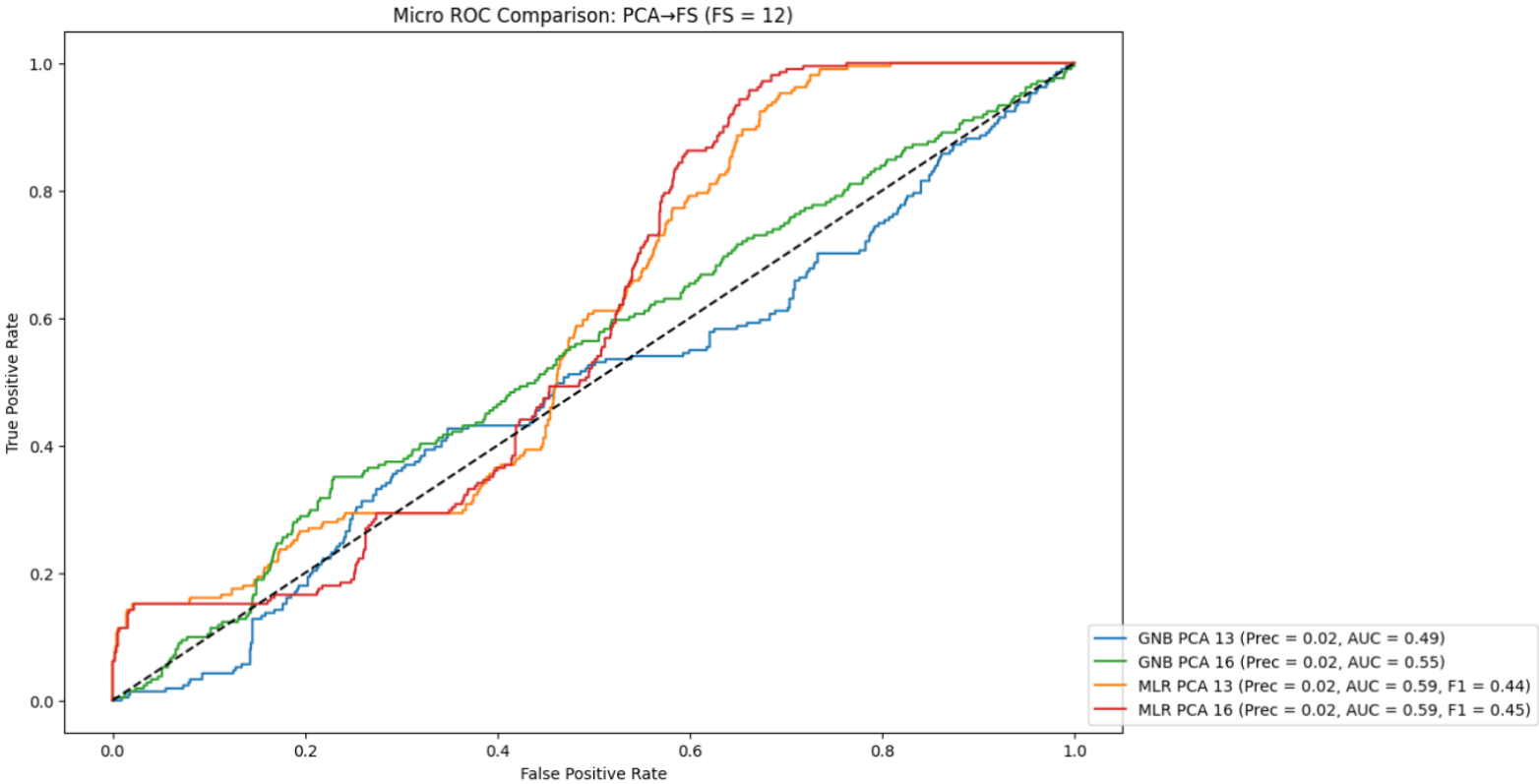


- ✧ Analysis:
  - ➢ PCA has a positive effect for GNB classifier for Wine and Forest Dataset, it's likely that in low-sample scenarios, this helps mitigate variance in parameter estimates and improves generalization, while in other dataset, the assumption that features are conditionally independent is broken, so AUCs slightly drops.

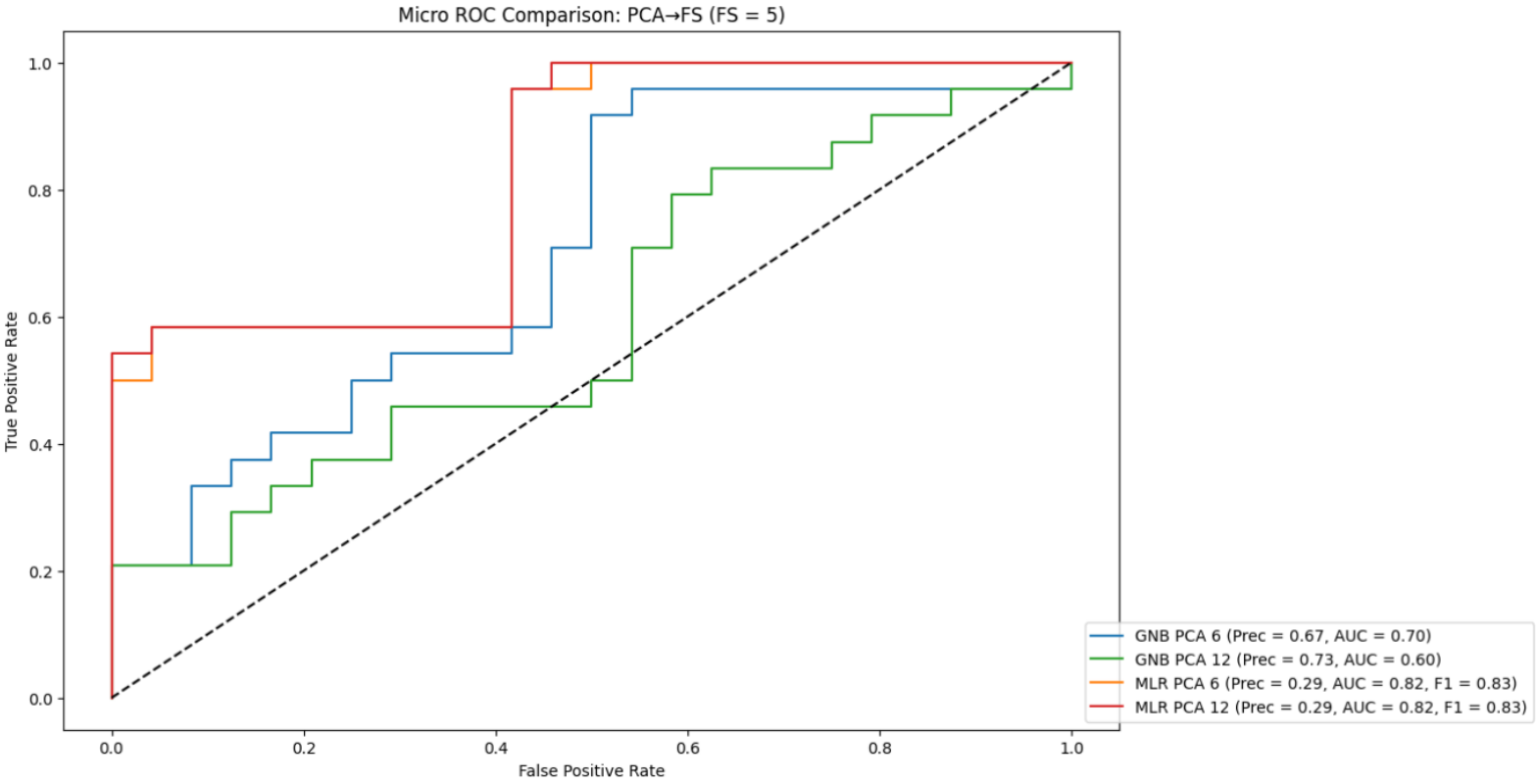
  - ➢ For MLR, applying PCA slightly improves the precision. This suggests that PCA effectively removes noisy or redundant features, allowing MLR to focus on more informative directions that align better with the decision boundary.

  - ➢ When retaining components that preserve at least 75% to 90% of the total variance, AUCs reach the peak.

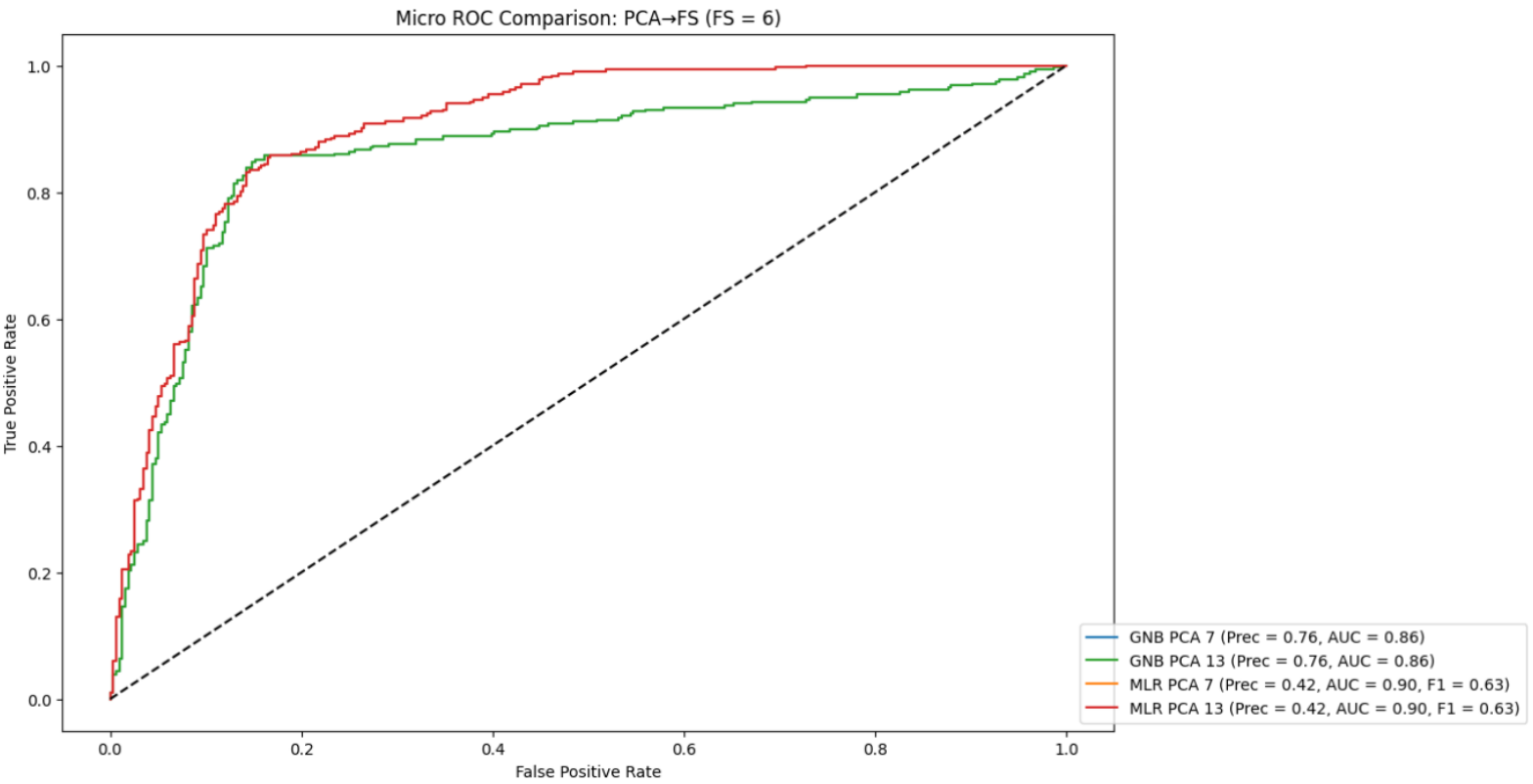
3. With PCA or without PCA before Feature Selection
   - ✧ Result:
     Obesity:



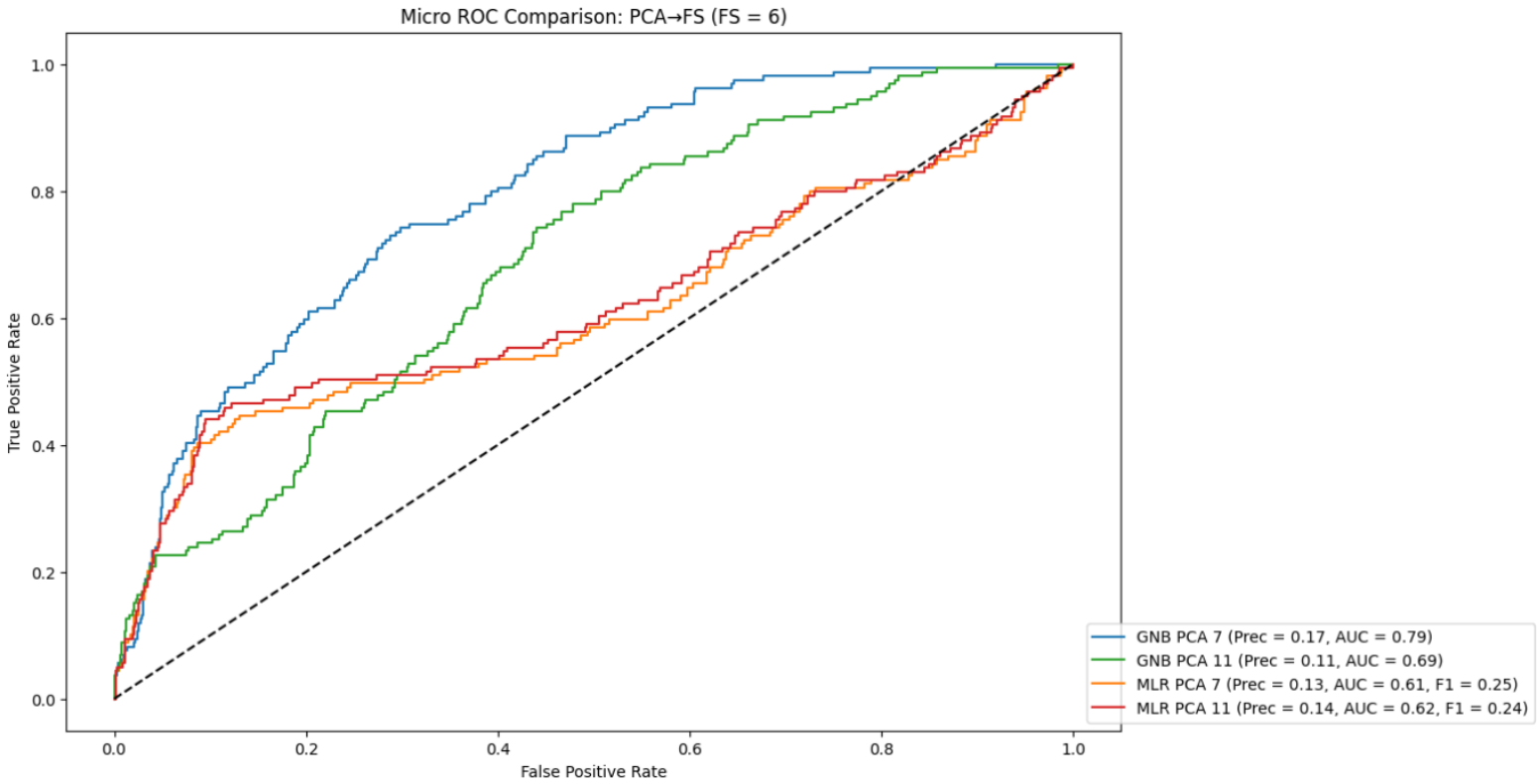
Wine:


Micro ROC Comparison: PCA→FS (FS = 5)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
GNB PCA 6 (Prec = 0.67, AUC = 0.70)
GNB PCA 12 (Prec = 0.73, AUC = 0.60)
MLR PCA 6 (Prec = 0.29, AUC = 0.82, F1 = 0.83)
MLR PCA 12 (Prec = 0.29, AUC = 0.82, F1 = 0.83)

Forest:


Micro ROC Comparison: PCA→FS (FS = 6)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
GNB PCA 7 (Prec = 0.76, AUC = 0.86)
GNB PCA 13 (Prec = 0.76, AUC = 0.86)
MLR PCA 7 (Prec = 0.42, AUC = 0.90, F1 = 0.63)
MLR PCA 13 (Prec = 0.42, AUC = 0.90, F1 = 0.63)

Churn:



- Analysis:
  Since there are too many pair results, I can't find the principle across every dataset to determine whether it's definite useful to do PCA before FS. It depends on the dataset, which means try and error may be the best way to maximize the performance, but it's worth trying when the features are more than 15 and target classes are complicated.

IV. Appendix

1. implements Linear Discriminant Analysis (LDA) for dimensionality reduction. It projects the input data into a lower-dimensional space that maximizes class separability by solving the generalized eigenvalue problem between the between-class scatter matrix Sb and the within-class scatter matrix Sw.

```
    n_samples, n_features = X.shape
    class_labels = np.unique(Y)
    n_classes = len(class_labels)

    if num_components is None:
        num_components = n_classes - 1

    mean_overall = np.mean(X, axis=0)

    S_w = np.zeros((n_features, n_features))
    S_b = np.zeros((n_features, n_features))

    for c in class_labels:
        X_c = X[Y == c]
        mean_c = np.mean(X_c, axis=0)
        S_w += (X_c - mean_c).T @ (X_c - mean_c)
        n_c = X_c.shape[0]
        mean_diff = (mean_c - mean_overall).reshape(n_features, 1)
        S_b += n_c * (mean_diff @ mean_diff.T)

    eigvals, eigvecs = np.linalg.eig(np.linalg.pinv(S_w) @ S_b)

    sorted_indices = np.argsort(np.real(eigvals))[::-1]
    topk_eigvecs = np.real(eigvecs[:, sorted_indices[:num_components]])

    X_lda = X @ topk_eigvecs

    return X_lda, topk_eigvecs
```

2. Plot data distribution after LDA projections for 1, 2, 3, 4 dimensions.

```
def plot_1d_projection(X_proj, Y, title="1D Projection", filename="1d_projection.png", label_prefix="Component"):
    import matplotlib.pyplot as plt
    Y = np.array(Y)
    X_proj = np.array(X_proj).flatten()

    plt.figure(figsize=(10, 4))
    classes = np.unique(Y)
    for c in classes:
        plt.scatter(X_proj[Y == c], [c] * np.sum(Y == c), label=f"Class {c}", alpha=0.7, s=20)

    plt.xlabel(f"{label_prefix} 1")
    plt.yticks(classes)
    plt.title(title)
    plt.legend()
    plt.tight_layout()
    plt.savefig(filename)
    plt.close()
```

```python
def plot_2d_projection(X_proj, Y, title="2D Projection", filename="2d_projection.png", label_prefix="Component"):
    Y = np.array(Y)
    plt.figure(figsize=(8, 6))
    classes = np.unique(Y)
    for c in classes:
        plt.scatter(X_proj[Y == c, 0], X_proj[Y == c, 1], label=f'Class {c}', alpha=0.7)

    plt.xlabel(f"{label_prefix} 1")
    plt.ylabel(f"{label_prefix} 2")
    plt.title(title)
    plt.legend()
    plt.tight_layout()
    plt.savefig(filename)
    plt.close()
def plot_3d_projection(X_proj, Y, title="3D Projection", filename="3d_projection.png", label_prefix="Component"):
    Y = np.array(Y)
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    classes = np.unique(Y)

    for c in classes:
        ax.scatter(X_proj[Y == c, 0], X_proj[Y == c, 1], X_proj[Y == c, 2], label=f'Class {c}', alpha=0.7)

    ax.set_xlabel(f"{label_prefix} 1")
    ax.set_ylabel(f"{label_prefix} 2")
    ax.set_zlabel(f"{label_prefix} 3")
    ax.set_title(title)
    ax.legend()
    plt.tight_layout()
    plt.savefig(filename)
    plt.close()
def plot_4d_projection(X_proj, Y, title="4D Projection (3D + color)", filename="4d_projection.png", label_prefix="Component"):
    Y = np.array(Y)
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    scatter = ax.scatter(
        X_proj[:, 0], X_proj[:, 1], X_proj[:, 2],
        c=X_proj[:, 3], cmap='viridis', alpha=0.7
    )

    ax.set_xlabel(f"{label_prefix} 1")
    ax.set_ylabel(f"{label_prefix} 2")
    ax.set_zlabel(f"{label_prefix} 3")
    ax.set_title(title)
    cbar = fig.colorbar(scatter, ax=ax, label=f"{label_prefix} 4")
    plt.tight_layout()
    plt.savefig(filename)
    plt.close()
```

3. Choose PCA retained dimensions according to the explained variance ratio.

```
    for threshold in variance_threshold:
        n_components = int(np.argmax(cumulative_variance >= threshold) + 1)
        if n_components == 0:
            n_components = 1
        pca_dims.append(n_components)
        print(f"Number of components for {threshold*100:.1f}% variance: {n_components}")
```

Find the projection that can maximize the variance.

```
if num_components is not None:
    pca_results = {}
    W = sorted_eigenvectors[:, :num_components]
    X_pca = np.dot(X, W)
    var_ratio = explained_variance_ratio[:num_components]
    pca_results = (X_pca, W, var_ratio)
    return pca_results, num_components, sorted_eigenvalues
else:
    pca_dims = []
    for threshold in variance_threshold:
        n_components = int(np.argmax(cumulative_variance >= threshold) + 1)
        if n_components == 0:
            n_components = 1
        pca_dims.append(n_components)
        print(f"Number of components for {threshold*100:.1f}% variance: {n_components}")

    max_feature = X.shape[1]
    pca_dims.append(max_feature)
    print(f"Number of components for 100% variance: {max_feature}")
    pca_dims = sorted(list(set(pca_dims)))

    pca_results = {}
    for n_components in pca_dims:
        W = sorted_eigenvectors[:, :n_components]
        X_pca = np.dot(X, W)
        var_ratio = explained_variance_ratio[:n_components]
        pca_results[n_components] = (X_pca, W, var_ratio)

    return pca_results, pca_dims, sorted_eigenvalues
```

4. Use mutual information to perform feature selection

```python
def compute_mutual_information(X, Y):
    n_samples, n_features = X.shape
    n_classes = len(np.unique(Y))
    mi = np.zeros(n_features)

    for f in range(n_features):
        X_feature = X[:, f]
        bins = np.linspace(np.min(X_feature), np.max(X_feature), 16)
        X_feature_binned = np.digitize(X_feature, bins) - 1
        X_feature_binned = np.clip(X_feature_binned, 0, len(bins) - 2)

        joint_hist = np.zeros((n_classes, len(bins) - 1))
        for i in range(n_samples):
            joint_hist[Y[i], X_feature_binned[i]] += 1
        joint_hist /= n_samples

        p_y = np.sum(joint_hist, axis=1)  # Y 的邊緣分佈
        p_x = np.sum(joint_hist, axis=0)  # X 的邊緣分佈

        mi_feature = 0
        for y in range(n_classes):
            for x in range(len(bins) - 1):
                if joint_hist[y, x] > 0 and p_y[y] > 0 and p_x[x] > 0:
                    mi_feature += joint_hist[y, x] * np.log(joint_hist[y, x] / (p_y[y] * p_x[x]))

        mi[f] = mi_feature

    return mi
```

5. Plot the AUC curve

```python
print(f"\nPCA→FS (FS={fs_dim}) Results:")
print("GNB:")
for dim in pca_dims:
    if (dim, fs_dim) in gnb_results:
        r = gnb_results[(dim, fs_dim)]
        print(f"  PCA {dim} → FS {fs_dim} - Prec: {r['prec']:.4f}, AUC: {r['auc_micro']:.4f}")
print("MLR:")
for dim in pca_dims:
    if (dim, fs_dim) in mlr_results:
        r = mlr_results[(dim, fs_dim)]
        print(f"  PCA {dim} → FS {fs_dim} - Prec: {r['prec']:.4f}, AUC: {r['auc_micro']:.4f}, Avg Val F1: {r['avg_val_f1']:.4f}")
```

```python
def compute_roc_auc(y_true, y_score, num_classes):
    y_true_one_hot = np.eye(num_classes)[y_true]
    fpr, tpr, auc = [], [], []

    for i in range(num_classes):
        sorted_indices = np.argsort(y_score[:, i])[::-1] # [:, i] for all rows, i-th column, [::-1] for descending order, start, end, step
        y_true_sorted = y_true_one_hot[:, i][sorted_indices] # rearranging the order along the column in y_true_one_hot according to the sorted indices
        y_score_sorted = y_score[:, i][sorted_indices]

        tp, fp = 0, 0
        tpr_i, fpr_i = [0], [0]
        total_positives = np.sum(y_true_one_hot[:, i])
        total_negatives = len(y_true_sorted) - total_positives

        for j in range(len(y_true_sorted)):
            if y_true_sorted[j] == 1:
                tp += 1
            else:
                fp += 1
            tpr_i.append(tp / total_positives if total_positives > 0 else 0)
            fpr_i.append(fp / total_negatives if total_negatives > 0 else 0)

        fpr.append(fpr_i)
        tpr.append(tpr_i)
        # 使用梯形法則計算 AUC
        auc_i = np.trapz(tpr_i, fpr_i)
        auc.append(auc_i if auc_i >= 0 else 0)  # 確保不小於 0

    # Micro ROC 和 Micro AUC
    y_true_flat = y_true_one_hot.ravel()
    y_score_flat = y_score.ravel()
    sorted_indices = np.argsort(y_score_flat)[::-1]
    y_true_sorted = y_true_flat[sorted_indices]
    y_score_sorted = y_score_flat[sorted_indices]

    tp, fp = 0, 0
    tpr_micro, fpr_micro = [0], [0]
    total_positives = np.sum(y_true_flat)
    total_negatives = len(y_true_flat) - total_positives

    for j in range(len(y_true_sorted)):
        if y_true_sorted[j] == 1:
            tp += 1
        else:
            fp += 1
        tpr_micro.append(tp / total_positives if total_positives > 0 else 0)
        fpr_micro.append(fp / total_negatives if total_negatives > 0 else 0)

    auc_micro = np.trapz(tpr_micro, fpr_micro)
    auc_micro = auc_micro if auc_micro >= 0 else 0  # 確保不小於 0

    return fpr, tpr, auc, fpr_micro, tpr_micro, auc_micro
```

Thank you for reading, hope you have a nice day. :)