110612117 張仲瑜 HW1 report

I. Introduction

1. Goal

In this task, we aim to conduct experiments on two types of classifiers, observing their performance on four datasets, two of them are two-class and the others are multi-class tasks.

2. Core Idea

I implemented two classifiers, which are Gaussian Naïve Bayes (GNB) classifier and Multinomial Logistic Regression (MLR) classifier, then conducted experiments including dimension reduction through Primary Component Analysis, dimension reduction through feature selection to observe the performance indicators (ROC, AUC) under those settings.

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 [1]

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 [2]

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 GNB and MLR classifiers [3]

   Implement two classes with "fit" and "predict" functions, allowing the training and testing processes separated.

4. Dimension reduction with PCA (experiment) [4]

   Implement the Primary Component Analysis to reduce the dimension to the given dimension.

5. Dimension reduction with Feature Selection (experiment) [5]

   Calculate the mutual information between each feature and targeted feature to choose features that have similar distribution.

6. Metrices computation and figure plotting [6]

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

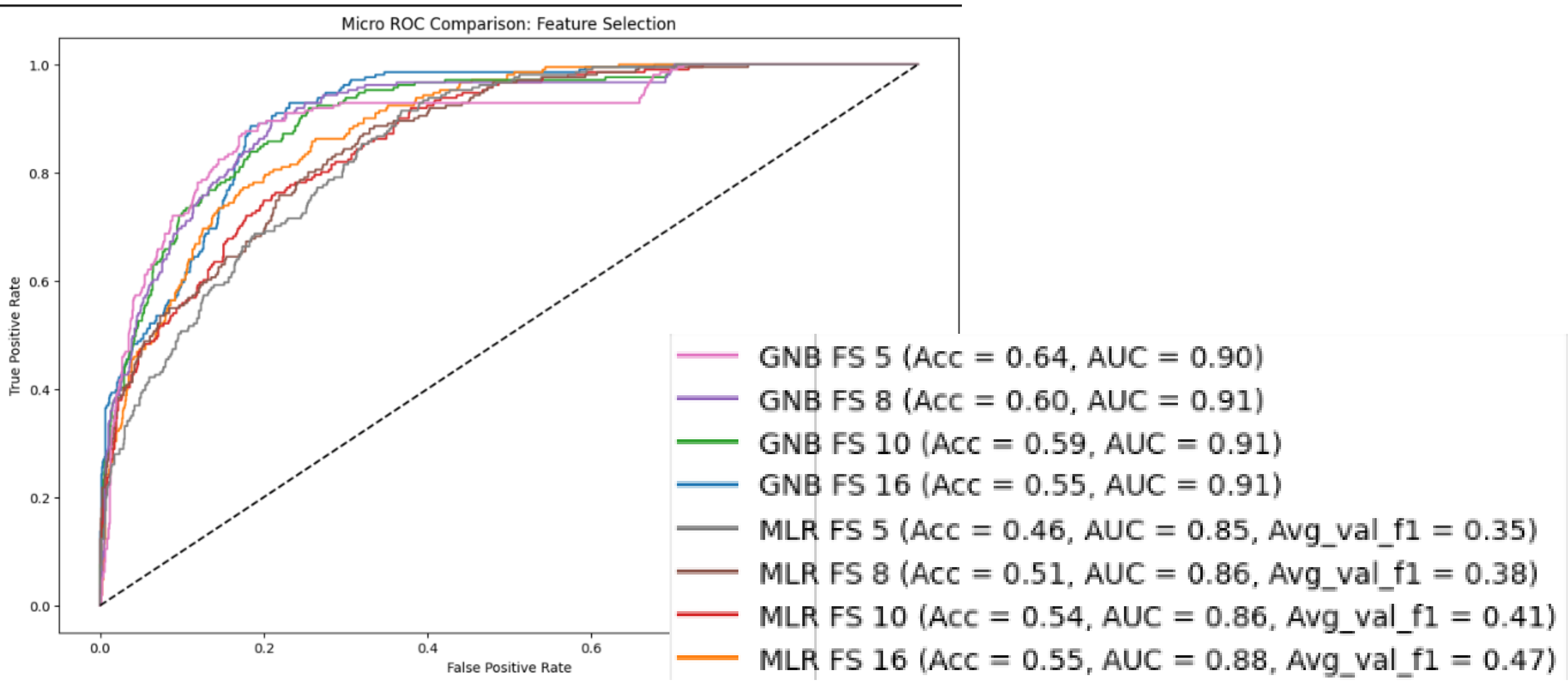
III. Experiment Result and Analysis

1. Feature selection

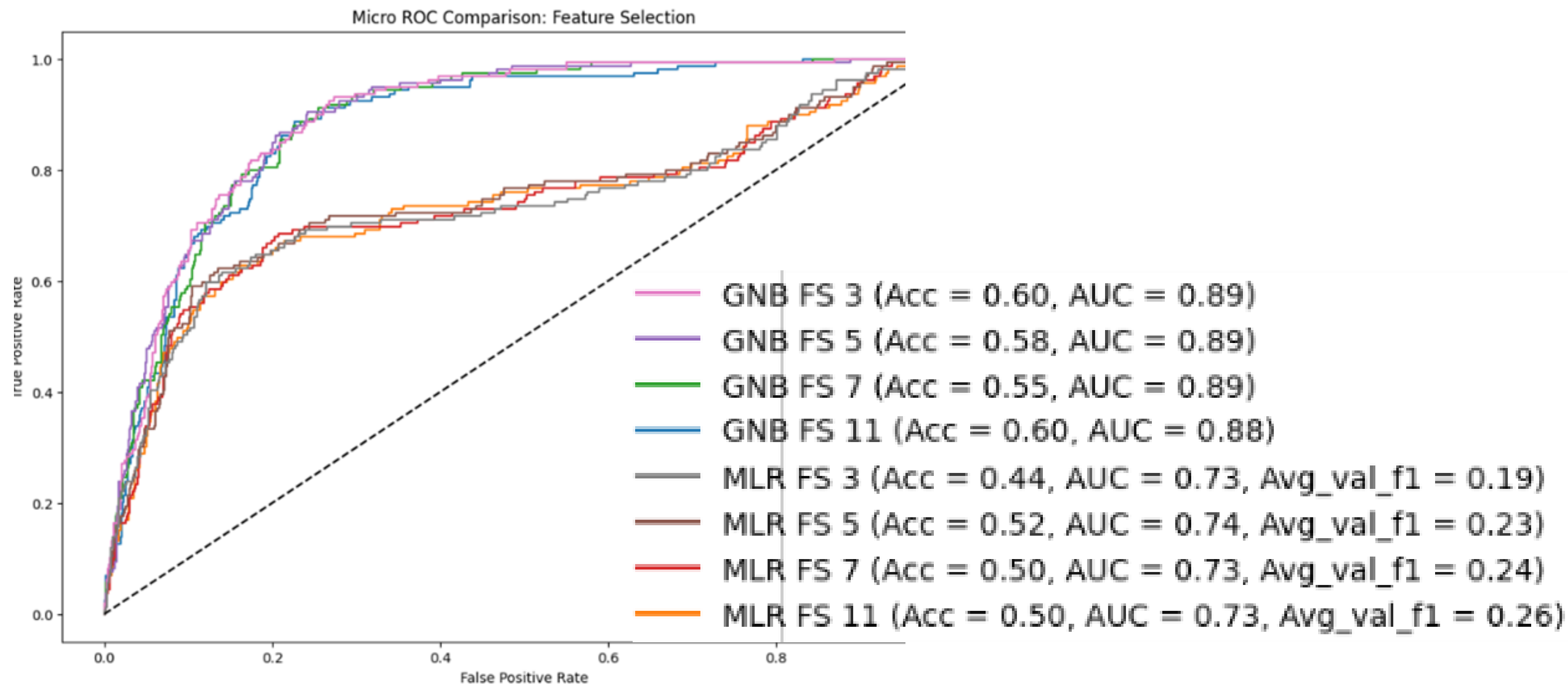
   ✧ Expectation:
   It sounds reasonable to classify the data with similar distribution since it may indicate that they are highly related, so I expect it may be helpful for most of the dataset.

✧ Result:

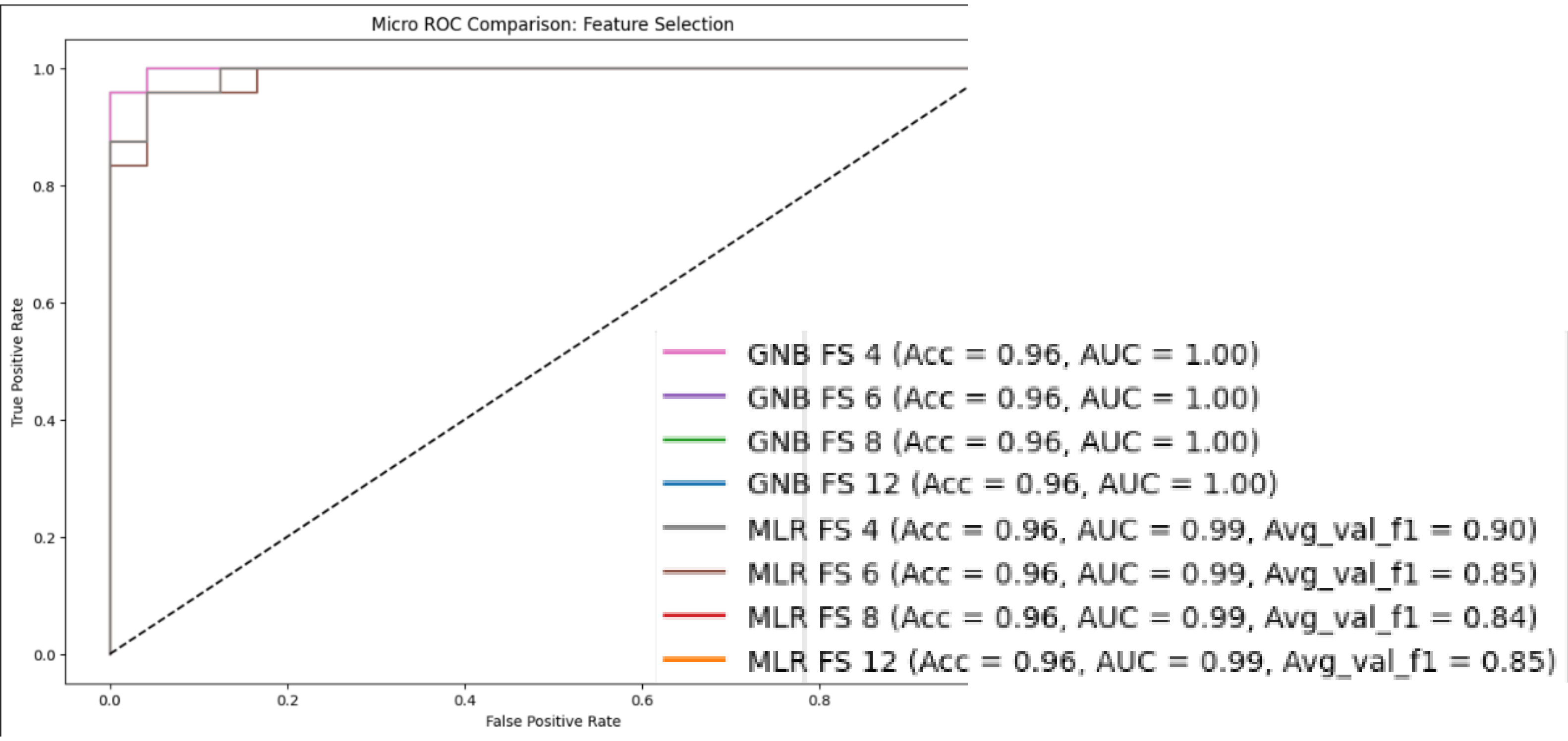
Obesity:



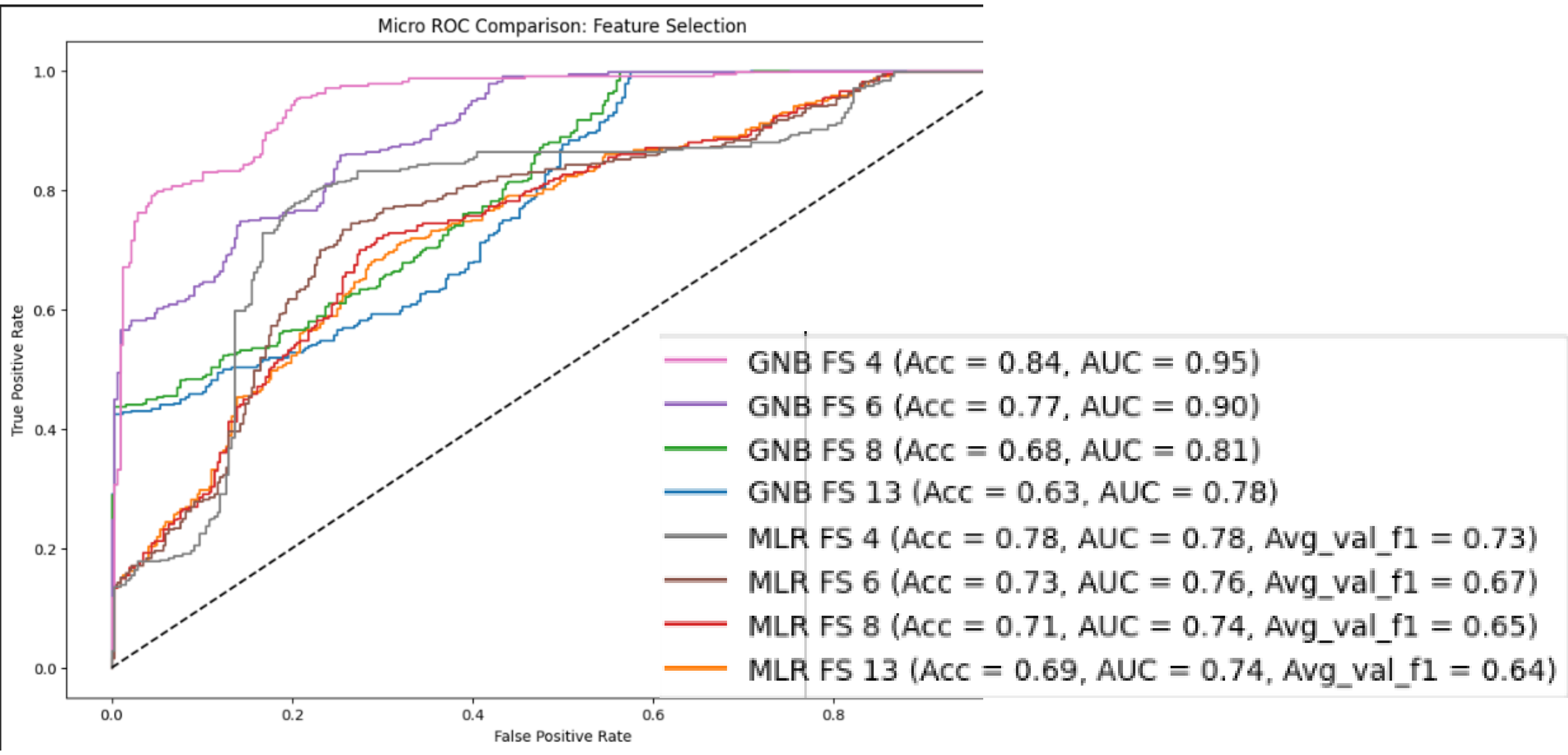
Wine:



Forest:



Churn:

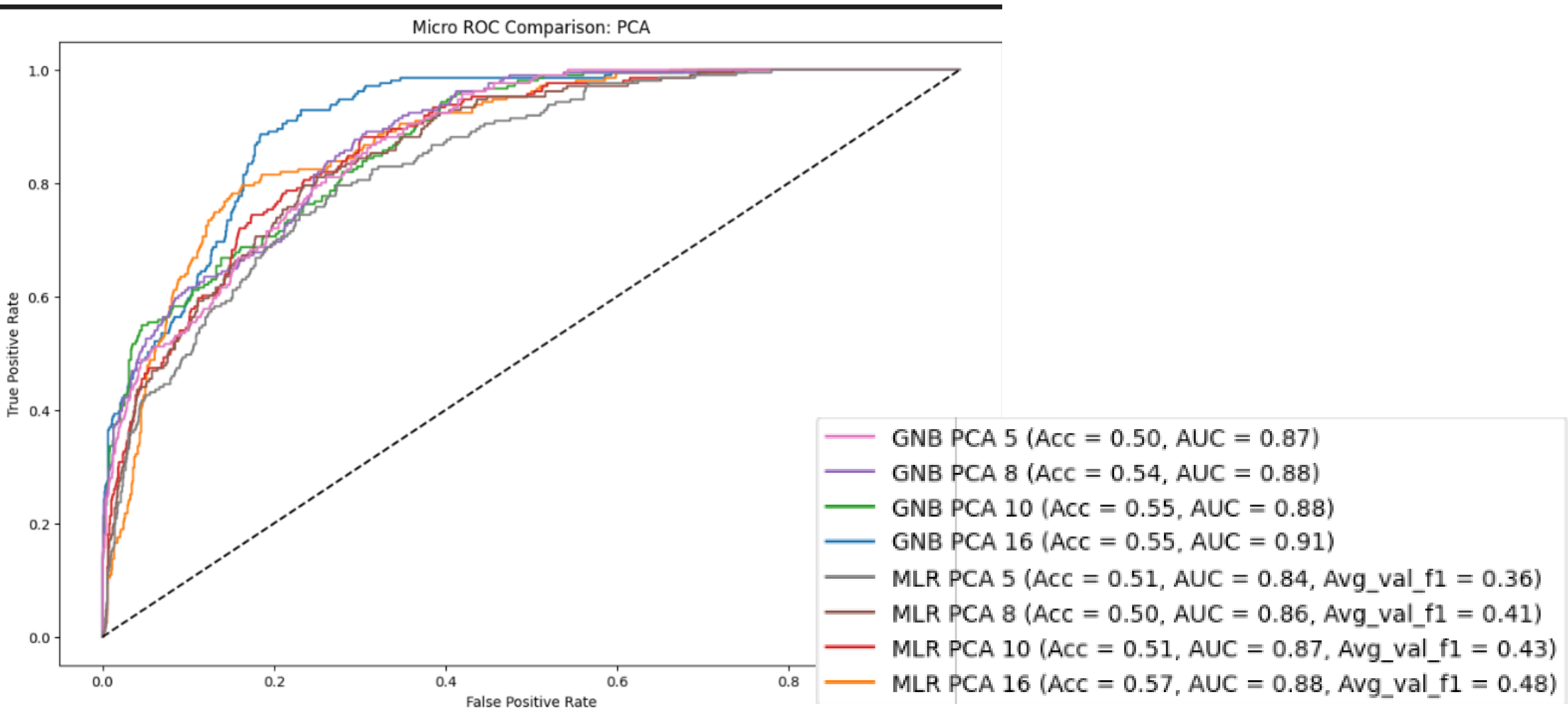


- ✧ Analysis:
  - ➢ On multi-class dataset, feature selection has slightly positive effect on GNB and causes harm to MLR. On the other hand, it significantly boosted performance on two-classes datasets.
  - ➢ My interpretation:
    For multi-class tasks, MLR requires more features to define the detail boundary between multiple classes. Features having lower mutual info with targeted output may provide decisive info between similar classes.
    For two-class tasks, we only need a couple crucial features to classify data. Feature selection helps both MLR and GNB prevent overfitting and dimensional curse.

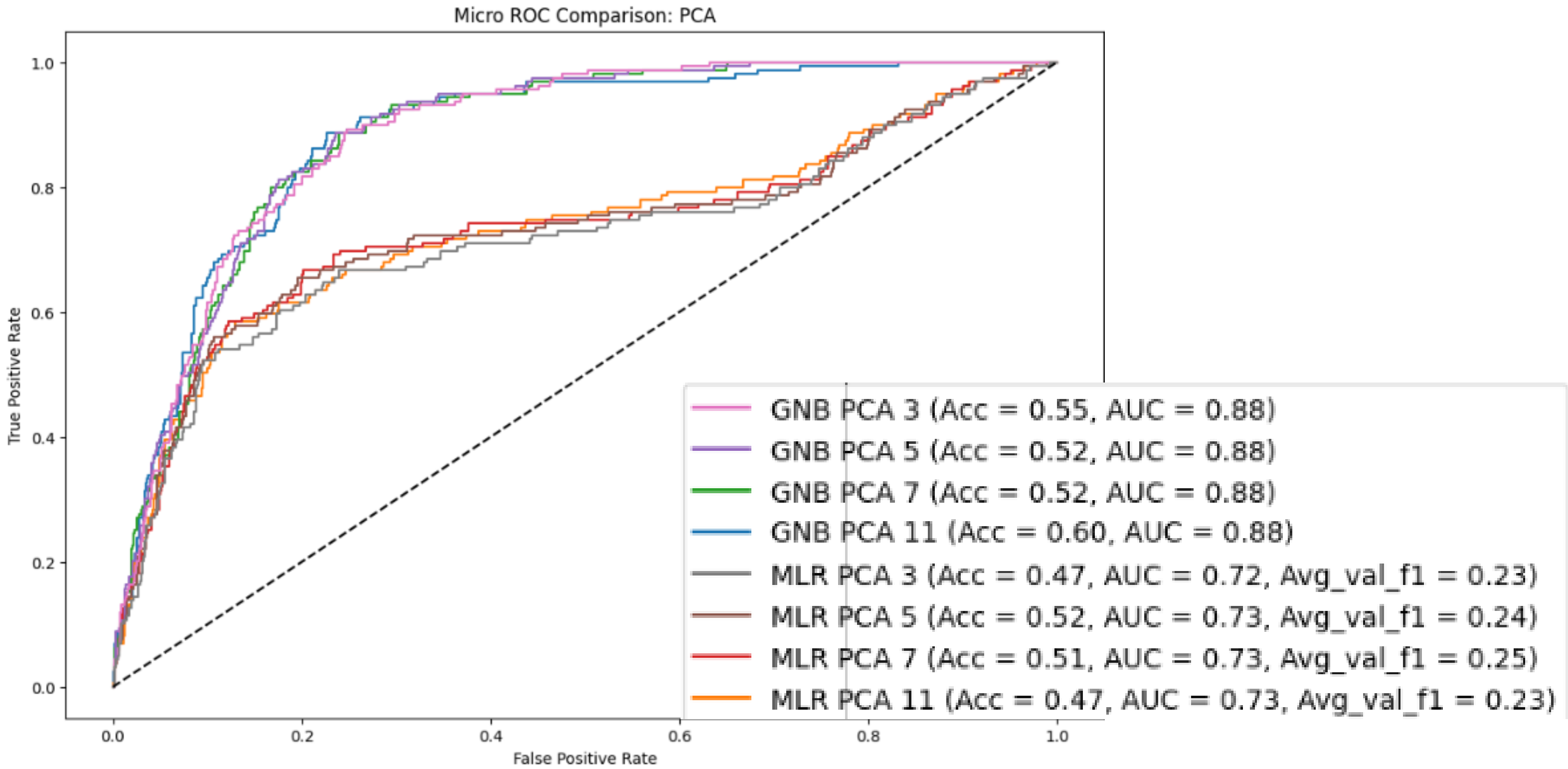
2. Primary Component Analysis

- ✧ Expectation:
  PCA will have a positive influence on GNB while having a negative influence on MLR.

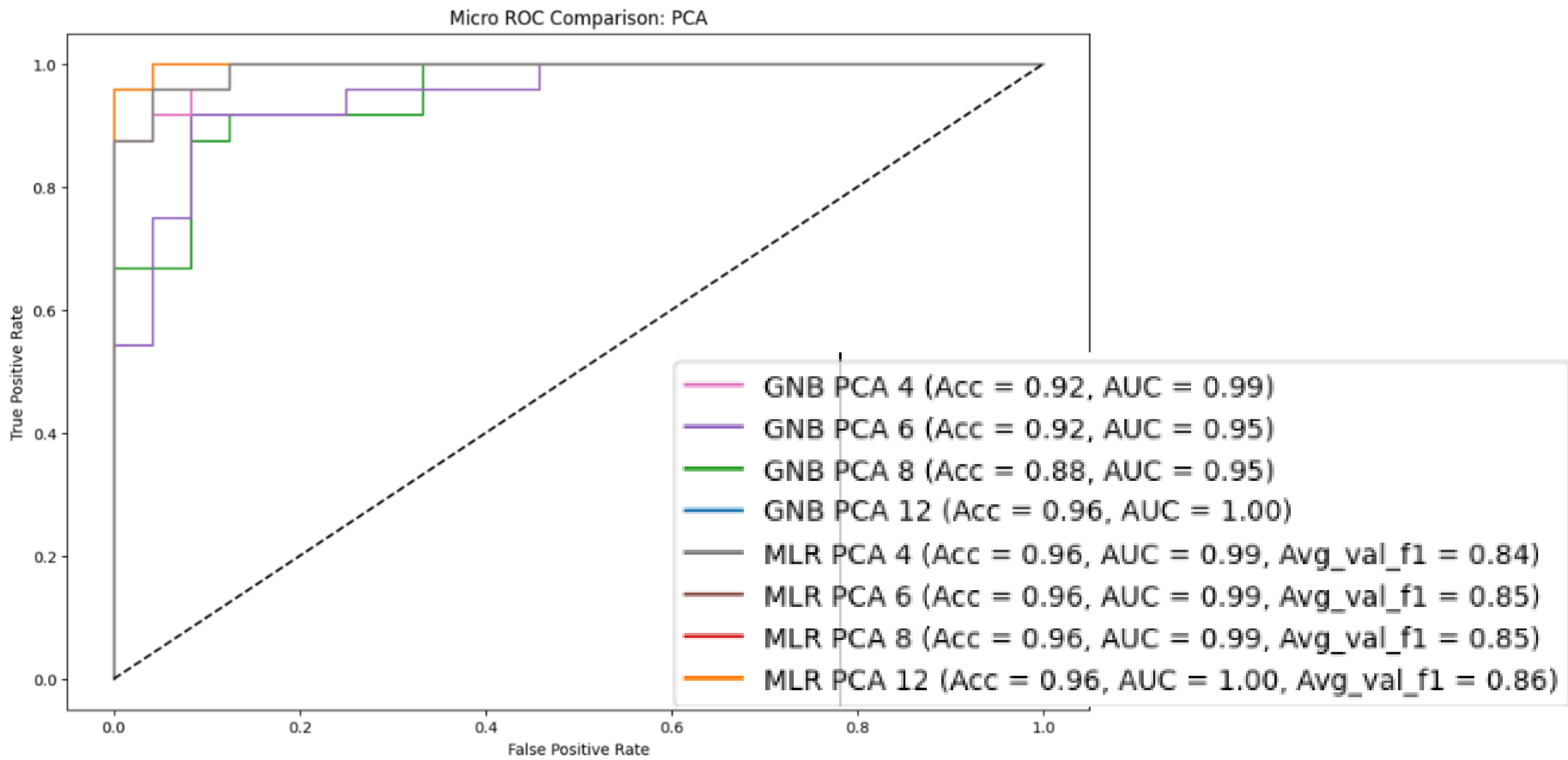
- ✧ Result
  Obesity:



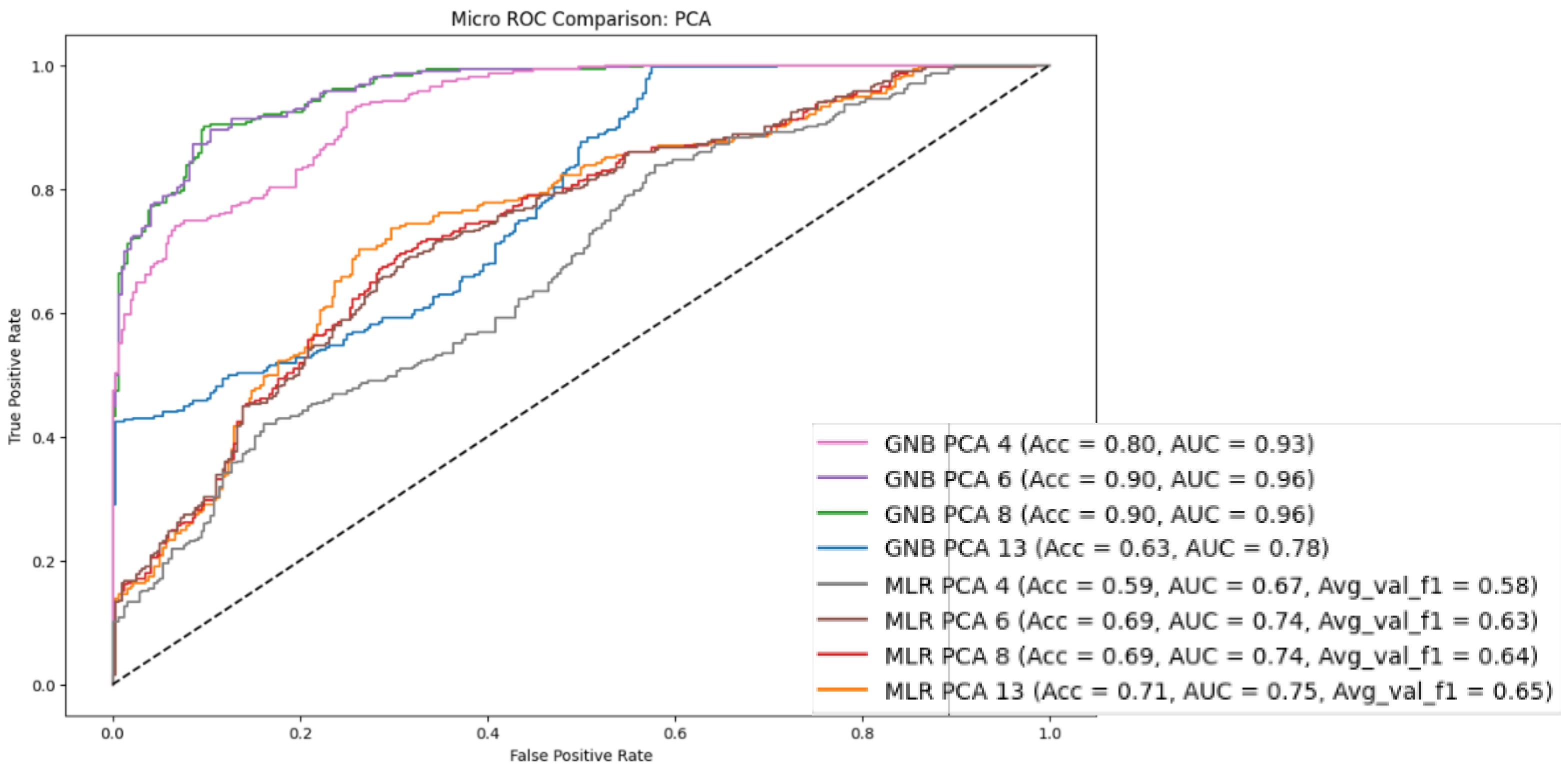
Wine:


Micro ROC Comparison: PCA
True Positive Rate
False Positive Rate
GNB PCA 3 (Acc = 0.55, AUC = 0.88)
GNB PCA 5 (Acc = 0.52, AUC = 0.88)
GNB PCA 7 (Acc = 0.52, AUC = 0.88)
GNB PCA 11 (Acc = 0.60, AUC = 0.88)
MLR PCA 3 (Acc = 0.47, AUC = 0.72, Avg_val_f1 = 0.23)
MLR PCA 5 (Acc = 0.52, AUC = 0.73, Avg_val_f1 = 0.24)
MLR PCA 7 (Acc = 0.51, AUC = 0.73, Avg_val_f1 = 0.25)
MLR PCA 11 (Acc = 0.47, AUC = 0.73, Avg_val_f1 = 0.23)

Forest:


Micro ROC Comparison: PCA
True Positive Rate
False Positive Rate
GNB PCA 4 (Acc = 0.92, AUC = 0.99)
GNB PCA 6 (Acc = 0.92, AUC = 0.95)
GNB PCA 8 (Acc = 0.88, AUC = 0.95)
GNB PCA 12 (Acc = 0.96, AUC = 1.00)
MLR PCA 4 (Acc = 0.96, AUC = 0.99, Avg_val_f1 = 0.84)
MLR PCA 6 (Acc = 0.96, AUC = 0.99, Avg_val_f1 = 0.85)
MLR PCA 8 (Acc = 0.96, AUC = 0.99, Avg_val_f1 = 0.85)
MLR PCA 12 (Acc = 0.96, AUC = 1.00, Avg_val_f1 = 0.86)

Churn:



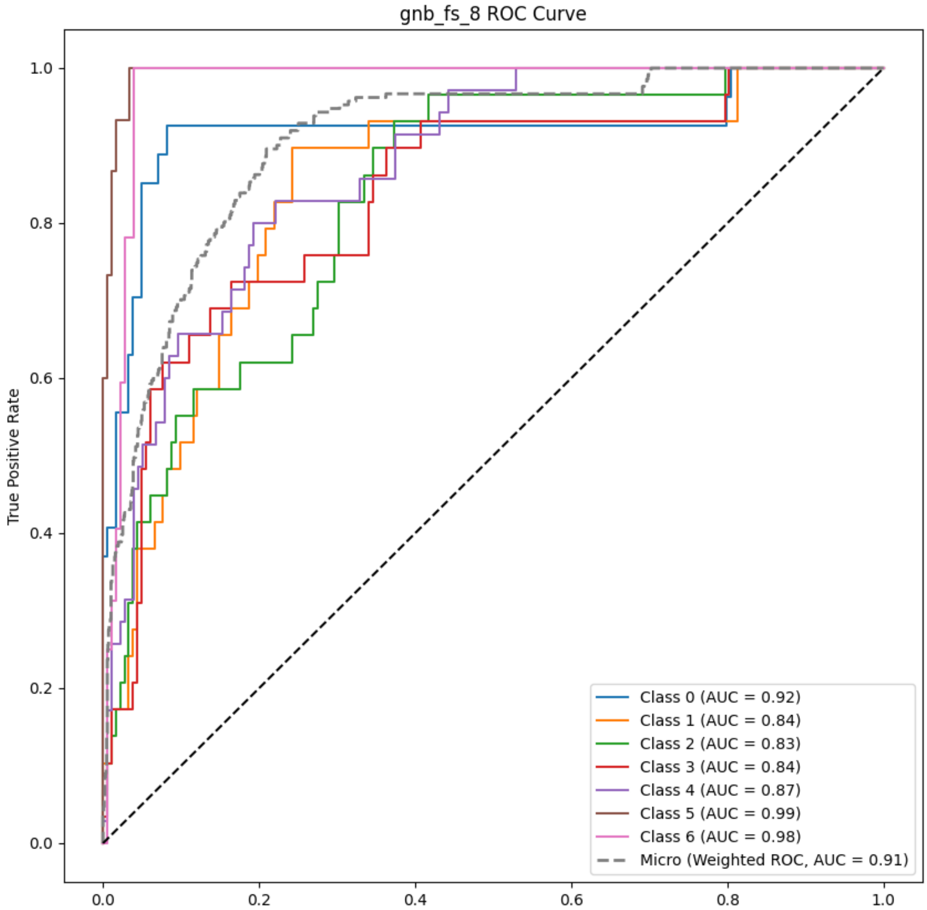
- Analysis: PCA has either no or negative effect on most of the datasets, except for GNB on the Churn dataset, the Acc and AUC reach the peak when PCA feature = 6~8

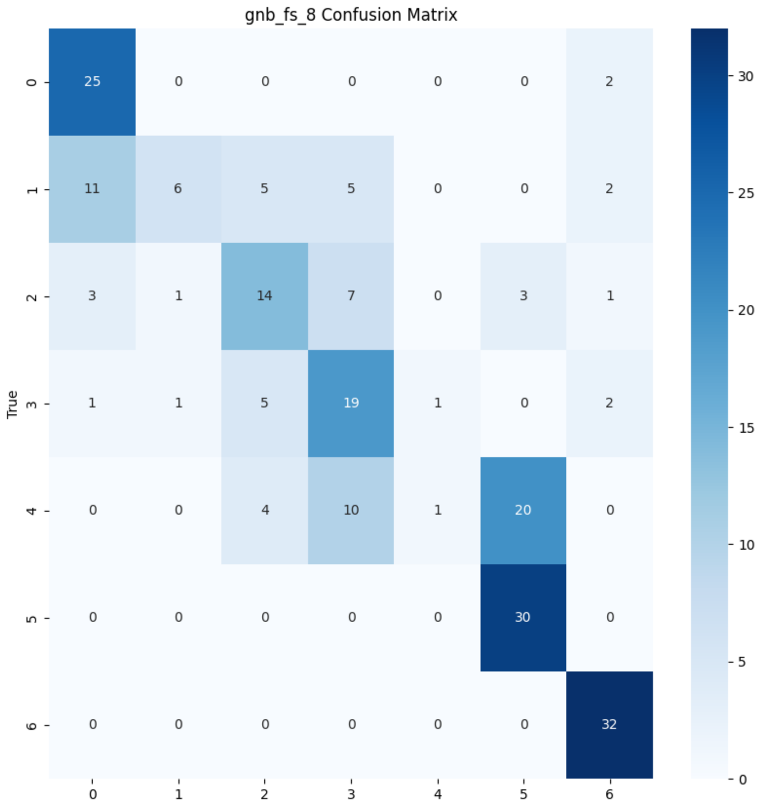
- My interpretation:
  Theoretically, GNB should benefit from PCA since its assumption of feature independence, which is aligned with PCA's orthogonal feature space.

  As a result, I printed the variance ratio in multi-class tasks, finding that the variance contribution from different features is similar, which means that the classifier loss lots of information when performing it, which may be one of the results.
  As for GNB in forest dataset, the original performance is good enough, the task may be too simple.

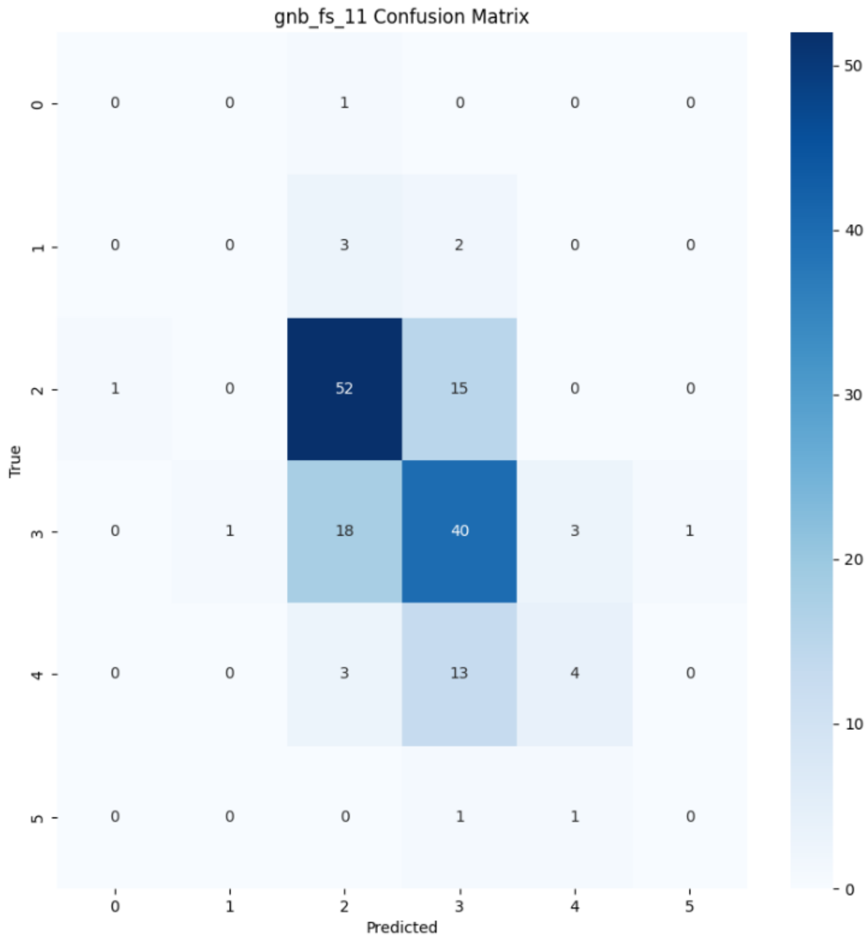
  On the other hand, MLR struggles in the PCA-transformed space since it relies on the linear relationships of the original features to define decision boundaries.

3. Other observation:





- For obesity dataset, if we want to get better performance, we need to collect more data about overweight level 1 and overweight level 2 to let the classifer learn more about their chracteristics.



- Severe data unbalance exists in the wine dataset.
  We need greater diversity in the representation of both poor quality and excellent wines.

IV. Appendix

1. Use map function to clean and load the data.

```python
def load_and_preprocess_data(file_path):
    df = pd.read_csv(file_path)
    df.columns = df.columns.str.strip()
    print("Columns:", df.columns.tolist())
    if file_path == "data/obesity.csv":
        df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
        df['family_history_with_overweight'] = df['family_history_with_overweight'].map({'no': 0, 'yes': 1})
        df['FAVC'] = df['FAVC'].map({'no': 0, 'yes': 1})
        df['CAEC'] = df['CAEC'].map({'no': 0, 'Sometimes': 1, 'Frequently': 2, 'Always': 3})
        df['SMOKE'] = df['SMOKE'].map({'no': 0, 'yes': 1})
        df['SCC'] = df['SCC'].map({'no': 0, 'yes': 1})
        df['CALC'] = df['CALC'].map({'no': 0, 'Sometimes': 1, 'Frequently': 2, 'Always': 3})
        df['MTRANS'] = df['MTRANS'].map({'Walking': 0, 'Bike': 1, 'Motorbike': 2, 'Public_Transportation': 3, 'Automobile': 4})
        df['NObeyesdad'] = df['NObeyesdad'].map({'Insufficient_Weight': 0, 'Normal_Weight': 1, 'Overweight_Level_I': 2, 'Overweight_Lev

        X = df.drop(columns=['NObeyesdad'], axis=1).values
        Y = df['NObeyesdad'].values
        num_classes = len(np.unique(Y))
```

2. Split the data into train, validation and test for the following process, also make sure the distribution are similar.

```python
for cls in classes:
    if test_samples_per_class[cls] == 0 and counts[classes == cls][0] > 0:
        test_samples_per_class[cls] = 1
current_test_size = sum(test_samples_per_class.values())
if current_test_size < test_size:
```

```python
        test_size = n_samples // k
        test_indices = indices[:test_size]
        train_val_indices = indices[test_size:]

    X_test, Y_test = X[test_indices], Y[test_indices]
    train_val_split = k_fold_cross_validation(len(train_val_indices), k=k, shuffle=True, random_seed=random_seed)

    print("Class distribution in full dataset:", np.bincount(Y))
    print("Class distribution in test set:", np.bincount(Y_test))
    print("Class distribution in train/val set:", np.bincount(Y[train_val_indices]))

    return X_test, Y_test, train_val_indices, train_val_split
```

```python
def k_fold_cross_validation(n_samples, k=5, shuffle=True, random_seed = 42):
    indices = np.arange(n_samples)
    if shuffle:
        np.random.seed(random_seed)
        np.random.shuffle(indices)

    fold_sizes = np.full(k, n_samples // k, dtype=int)
    fold_sizes[:n_samples % k] += 1

    folds = []
    current = 0
    for fold_size in fold_sizes:
        start, stop = current, current + fold_size
        val_indices = indices[start:stop]
        train_indices = np.concatenate([indices[:start], indices[stop:]])
        folds.append((train_indices, val_indices))
        current = stop

    return folds
```

3. Implement the GNB and MLR classifier

```python
class GaussianNaiveBayes:
    def __init__(self):
        self.classes = None
        self.means = None
        self.variances = None
        self.priors = None

    def fit(self, X, Y):
        n_sample, n_features = X.shape
        self.classes = np.unique(Y)
        n_classes = len(self.classes)

        self.means = np.zeros((n_classes, n_features))
        self.variances = np.zeros((n_classes, n_features))
        self.priors = np.zeros(n_classes)

        for idx, c in enumerate(self.classes):
            X_c = X[Y == c]
            self.means[idx] = np.mean(X_c, axis=0)
            self.variances[idx] = np.var(X_c, axis=0)
            self.priors[idx] = len(X_c) / n_sample

    def predict(self, X):
        log_probs = np.zeros((X.shape[0], len(self.classes)))
        for idx, c in enumerate(self.classes):
            log_prob = -0.5 * np.sum(np.log(2 * np.pi * self.variances[idx]) + \
                        ((X - self.means[idx]) ** 2) / (2 * self.variances[idx]), axis=1)
            log_probs[:, idx] = np.log(self.priors[idx]) + log_prob

        probs = np.exp(log_probs - np.max(log_probs, axis=1, keepdims=True))
        probs /= np.sum(probs, axis=1, keepdims=True)
        return probs, np.argmax(log_probs, axis=1)
```

```python
class MultinomialLogisticRegression:
    def __init__(self, lr=0.01, epochs=1000):
        self.lr = lr
        self.epochs = epochs
        self.weights = None

    def softmax(self, z):
        exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
        return exp_z / np.sum(exp_z, axis=1, keepdims=True)

    def fit(self, X, Y):
        n_samples, n_features = X.shape
        n_classes = len(np.unique(Y))

        self.weights = np.zeros((n_features, n_classes))
        y_one_hot = np.eye(n_classes)[Y]

        for epoch in range(self.epochs):
            z = np.dot(X, self.weights)
            probs = self.softmax(z)
            error = probs - y_one_hot
            gradient = np.dot(X.T, error) / n_samples
            self.weights -= self.lr * gradient
            self.lr = self.lr * 0.95
            # if epoch % 100 == 0:
            #     loss = -np.mean(np.sum(y_one_hot * np.log(probs + 1e-10), axis=1))
            #     print(f"Epoch {epoch}, Loss: {loss:.4f}")

    def predict(self, X):
        z = np.dot(X, self.weights)
        probs = self.softmax(z)
        return probs, np.argmax(probs, axis=1)
```

4. Find eigenvalues and eigenvector to calculate primary components.

```python
def compute_pca(X, n_components):
    X_centered = X - np.mean(X, axis=0)
    covariance_matrix = np.cov(X_centered, rowvar=False)
    eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)

    sorted_indices = np.argsort(eigenvalues)[::-1]
    sorted_eigenvalues = eigenvalues[sorted_indices]
    sorted_eigenvectors = eigenvectors[:, sorted_indices]

    W = sorted_eigenvectors[:, :n_components]
    X_pca = np.dot(X_centered, W)

    # U, S, Vt = np.linalg.svd(X_centered, full_matrices=False)
    # sorted_eigenvalues = S**2 / (X.shape[0] - 1)
    # W = Vt.T[:, :n_components]
    # X_pca = U * S

    return X_pca, W, sorted_eigenvalues
```

5. Compute mutual information to serve as the criteria of 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)  # 限制範圍為 0 到 14

        joint_hist = np.zeros((n_classes, len(bins) - 1))  # 形狀為 (n_classes, 15)
        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
```

```python
def select_features(X, Y, n_features):
    mi = compute_mutual_information(X, Y)
    top_indices = np.argsort(mi)[::-1][:n_features]
    X_selected = X[:, top_indices]
    return X_selected, top_indices
```

6. Compute metrics – Calculate TP, FP, FN ro get precision, recall, F1…

```python
def compute_metrics(y_true, y_pred, num_classes):
    cm = confusion_matrix(y_true, y_pred, num_classes)
    accuracy = np.trace(cm) / np.sum(cm)
    precision = np.zeros(num_classes)
    recall = np.zeros(num_classes)
    f1_score = np.zeros(num_classes)

    for i in range(num_classes):
        tp = cm[i, i]
        fp = np.sum(cm[:, i]) - tp
        fn = np.sum(cm[i, :]) - tp

        precision[i] = tp / (tp + fp) if (tp + fp) > 0 else 0
        recall[i] = tp / (tp + fn) if (tp + fn) > 0 else 0
        f1_score[i] = 2 * (precision[i] * recall[i]) / (precision[i] + recall[i]) if (precision[i] + recall[i]) > 0 else 0

    total_precision = np.mean(precision)
    total_recall = np.mean(recall)
    total_f1_score = np.mean(f1_score)


    return accuracy, total_precision, total_recall, total_f1_score
```

7. Compute metrics – Calculate ROC and AUC

```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-
        y_true_sorted = y_true_one_hot[:, i][sorted_indices] # rearranging the ord
        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
```

```python
# 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.   :)