CISC Class7 Code

We will practice using the functions in the scikit-learn (sklearn) to measure / plot the accuracy, precision, recall, f1 score, roc and confusion matrix.

Create synthetic samples

We create a 3-class toy dataset, each class is sampled from a 2D Gaussian distribution.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, roc_auc_score,
                             roc_curve, precision_recall_curve,
average_precision_score)
from sklearn.preprocessing import label_binarize
from sklearn.metrics import ConfusionMatrixDisplay

# Set random seed for reproducibility
np.random.seed(0)

# Parameters
n_train = 50
n_test = 30
n_classes = 3

# Class centers (make class 0 and class 1 closer)
mean_0 = [0, 0]
mean_1 = [1, 1]  # Close to class 0
mean_2 = [3, 3]  # Far from class 0 and 1

cov = [[1, 0], [0, 1]]  # Identity covariance matrix

# Generate training data
X_train_0 = np.random.multivariate_normal(mean_0, cov, n_train)
X_train_1 = np.random.multivariate_normal(mean_1, cov, n_train)
X_train_2 = np.random.multivariate_normal(mean_2, cov, n_train)
```

```
y_train_0 = np.zeros(n_train)
y_train_1 = np.ones(n_train)
y_train_2 = np.full(n_train, 2)

X_train = np.vstack((X_train_0, X_train_1, X_train_2))
y_train = np.concatenate((y_train_0, y_train_1, y_train_2))

# Generate testing data
X_test_0 = np.random.multivariate_normal(mean_0, cov, n_test)
X_test_1 = np.random.multivariate_normal(mean_1, cov, n_test)
X_test_2 = np.random.multivariate_normal(mean_2, cov, n_test)

y_test_0 = np.zeros(n_test)
y_test_1 = np.ones(n_test)
y_test_2 = np.full(n_test, 2)

X_test = np.vstack((X_test_0, X_test_1, X_test_2))
y_test = np.concatenate((y_test_0, y_test_1, y_test_2))
```

Train a logistic regression model

Sklearn provides a simple-to-use-class for logistic regression function.

```
# Initialize the classifier
clf = LogisticRegression(multi_class='multinomial', solver='lbfgs')

# Train the classifier
clf.fit(X_train, y_train)

# Predict on test data
y_pred = clf.predict(X_test)

# Predict probabilities for ROC and PR curves
y_score = clf.predict_proba(X_test)
```

Predict the accuracy, precision, recall and F1 score for each class

```
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Precision, Recall, F1 Score per class
precision = precision_score(y_test, y_pred, average=None, labels=[0, 1,
2])
recall = recall_score(y_test, y_pred, average=None, labels=[0, 1, 2])
f1 = f1_score(y_test, y_pred, average=None, labels=[0, 1, 2])

for i in range(n_classes):
   print(f"Class {i}: Precision: {precision[i]:.2f}, Recall:
{recall[i]:.2f}, F1 Score: {f1[i]:.2f}")
```

Plot the confusion matrix

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1,
2])
disp.plot(cmap=plt.cm.Blues)
plt.show()
```

Plot ROC curve and compute ROC AUC

To compute ROC, we need first to specify the positive class. We use label_binarize to convert labels to one-hot labels.

The roc_curve function returns the the false positive rate and true positive rate under different thresholds

```
# Binarize the output for ROC and PR curves
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])

# Compute ROC AUC for each class
roc_auc = dict()
for i in range(n_classes):
   roc_auc[i] = roc_auc_score(y_test_bin[:, i], y_score[:, i])
   print(f"Class {i}: ROC AUC: {roc_auc[i]:.2f}")

# Plot ROC curves
for i in range(n_classes):
   fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_score[:, i])
   plt.plot(fpr, tpr, label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='best')
plt.show()
```

Plot precision-recall curve

Same as the ROC curve, we need to compute the precision-recall curve for each class.

```
# Plot Precision-Recall curves
for i in range(n_classes):
   precision_i, recall_i, _ = precision_recall_curve(y_test_bin[:, i],
y_score[:, i])
   average_precision = average_precision_score(y_test_bin[:, i],
y_score[:, i])

   plt.step(recall_i, precision_i, where='post', label=f'Class {i} (AP =
{average_precision:.2f})')

plt.xlabel('Recall')
plt.ylabel('Precision')
```

```
plt.title('Precision-Recall Curve')
plt.legend(loc='best')
plt.show()
```

Plot the decision boundary

```
# Decision boundary plot
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure()
plt.contourf(xx, yy, Z, alpha=0.4, cmap=plt.cm.RdYlBu)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=20, edgecolor='k',
cmap=plt.cm.RdYlBu)
plt.title('Decision Boundary')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```