Name: Rakshya Pandey

UID: C00580017

CSCE 509

Assignment 1

Github Link: https://github.com/rakshyaaa/PerceptronPattern/blob/main/Perceptron.ipynb

Part 0: Data creation

Generate a 2D, two-class data set. Class 0 are uniformly distributed in the (union of the) 3 squares [-1, 0]x[-1, 0] + [-1, 0]x[0, 1] + [0, 1]x[-1, 0]. Class 1 is Gaussian distributed centered at (0.5, 0.5) with variance (0.5, 0.5).

Generate 150 points in Class 0 and 50 points in Class 1. Plot the points in both classes, using a different color or shape for each class.

```
def generate_data_part0(n_class0=150, n_class1=50):
    """
    Returns:
    - X: feature matrix
    - y: class labels
    """
    # Class 0: Uniform distribution across 3 squares
    class0_points = []

    # Square 1: [-1, 0] x [-1, 0]
    square1 = np.random.uniform(low=[-1, -1], high=[0, 0], size=(n_class0//3,
2))

    # Square 2: [-1, 0] x [0, 1]
    square2 = np.random.uniform(low=[-1, 0], high=[0, 1], size=(n_class0//3,
2))

    # Square 3: [0, 1] x [-1, 0]
    square3 = np.random.uniform(low=[0, -1], high=[1, 0], size=(n_class0 -
2*(n_class0//3), 2))

    class0_points = np.vstack([square1, square2, square3])

    # Class 1: Gaussian distribution
    class1_points = np.random.normal(loc=[0.5, 0.5],
                                     scale=[0.5, 0.5],
                                     size=(n_class1, 2))

    # Combine data
    X = np.vstack([class0_points, class1_points])
```

```
    y = np.hstack([np.zeros(n_class0), np.ones(n_class1)])

    return X, y

X, y = generate_data_part0()


plt.figure(figsize=(8, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1],
                c='blue', label='Class 0', alpha=0.7)
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1],
                c='red', label='Class 1', alpha=0.7)
plt.title('Original Data Distribution')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```

```python
np.random.seed(42)
indices = np.random.permutation(len(X))
split_point = len(X) // 2

X_train, X_test = X[indices[:split_point]], X[indices[split_point:]]
y_train, y_test = y[indices[:split_point]], y[indices[split_point:]]

def evaluate_classifier(X, y, classifier, cv=4):
    """
    Evaluate classifier using 4-fold cross-validation

    Returns:
    - Metrics dictionary
    - Accuracy scores for variance calculation
    """
    # Stratified K-Fold for balanced splits
    skf = StratifiedKFold(n_splits=cv, shuffle=True, random_state=42)

    # Metrics to track
    accuracies = []
    precisions = []
    recalls = []
    aurocs = []

    for train_index, test_index in skf.split(X, y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Scale features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Fit classifier
        classifier.fit(X_train_scaled, y_train)
        y_pred = classifier.predict(X_test_scaled)

        # Compute metrics
        accuracies.append(accuracy_score(y_test, y_pred))
        precisions.append(precision_score(y_test, y_pred, zero_division=1))
        recalls.append(recall_score(y_test, y_pred, zero_division=1))

        # AUROC
        y_decision = classifier.decision_function(X_test_scaled)
        aurocs.append(roc_auc_score(y_test, y_decision))

    # Compute average metrics
    metrics = {
        'accuracy': np.mean(accuracies),
        'precision': np.mean(precisions),
        'recall': np.mean(recalls),
        'auroc': np.mean(aurocs),
        'accuracy_variance': np.var(accuracies)
    }

    return metrics, accuracies
```

Part 1: Perceptron

Implement a Perceptron classifier to separate the two classes. Use a test set of 50 Class 0 and 50 Class 1 points to evaluate the performance of your classifier. Use 4-fold cross validation to report the accuracy, precision, recall, AUROC metrics. What is the variance of the accuracy across the 4 runs?

```
perceptron = Perceptron(max_iter=1000, random_state=42)

perceptron_metrics, perceptron_accuracies = evaluate_classifier(X_train,
y_train, perceptron)

print("Perceptron Metrics:", perceptron_metrics)
```

Part 2: Linear SVM

A. Repeat Part 1 with a Linear SVM. Use a very small *C* hyperparameter.

```
svm_small_c = SVC(kernel='linear', C=0.01, random_state=42,
   probability=True)

svm_small_c_metrics, svm_small_c_accuracies = evaluate_classifier(X_train,
   y_train, svm_small_c)

      print("Linear SVM (Small C) Metrics:", svm_small_c_metrics)
```

B. Repeat Part 1 with a Linear SVM. Use a very large *C* hyperparameter

```
svm_large_c = SVC(kernel='linear', C=1000, random_state=42, probability=True)
svm_large_c_metrics, svm_large_c_accuracies = evaluate_classifier(X_train,
y_train, svm_large_c)
print("Linear SVM (Large C) Metrics:", svm_large_c_metrics)
```

Part 3: Nonlinear SVM

A. Repeat Part 1 with a nonlinear SVM. Use a very small *C* hyperparameter.

```
svm_nonlinear_small_c = SVC(kernel='rbf', C=0.01, random_state=42,
   probability=True)

svm_nonlinear_small_c_metrics, svm_nonlinear_small_c_accuracies =
   evaluate_classifier(X_train, y_train, svm_nonlinear_small_c)

     print("Nonlinear SVM (Small C) Metrics:",
svm_nonlinear_small_c_metrics)
```

B. Repeat Part 1 with a nonlinear SVM. Use a very large *C* hyperparameter.

```
  svm_nonlinear_large_c = SVC(kernel='rbf', C=1000, random_state=42,
probability=True)
svm_nonlinear_large_c_metrics, svm_nonlinear_large_c_accuracies =
evaluate_classifier(X_train, y_train, svm_nonlinear_large_c)
print("Nonlinear SVM (Large C) Metrics:", svm_nonlinear_large_c_metrics)
```

C. Increase the Class 1 variance to (1.5, 1.5). Plot the data. Repeat Part 1 with a reasonable choice of *C*.

```
X_high_var, y_high_var = generate_data_part0(n_class0=150, n_class1=50)
np.random.seed(42)
indices_high_var = np.random.permutation(len(X_high_var))
split_point_high_var = len(X_high_var) // 2
X_train_high_var = X_high_var[indices_high_var[:split_point_high_var]]
y_train_high_var = y_high_var[indices_high_var[:split_point_high_var]]

svm_reasonable_c = SVC(kernel='rbf', C=10, random_state=42,
probability=True)
svm_reasonable_c_metrics, svm_reasonable_c_accuracies =
evaluate_classifier(X_train_high_var, y_train_high_var,
svm_reasonable_c)
print("Nonlinear SVM (Reasonable C) Metrics:",
svm_reasonable_c_metrics)
```

Part 4: Mismatch between training and test data

For test data, use the same distribution of Class 0 as before, such as in Part 3C. Similar to Part 3C, set the Class 1 variance to (1.5, 1.5).

For training data, change the Class 0 samples to be uniformly distributed in the (union of the) 3 squares [-0.5, 0.5]x[-0.5, 0.5] + [-0.5, 0.5]x[0.5, 0.5] + [0, 1]x[-0.5, 0.5]. Class 1 is Gaussian distributed centered at (0.5, 0.5) with variance (0.5, 0.5).

Implement a nonlinear SVM. Train using the training data. Use 4-fold cross validation and the test data to report the accuracy, precision, recall, AUROC metrics. What is the variance of the accuracy across the 4 runs?

```
def generate_mismatched_data(n_class0=150, n_class1=50):
    # Modified Class 0 distribution
    class0_points = np.random.uniform(low=[-0.5, -0.5], high=[1, 0.5],
size=(n_class0, 2))

    # Gaussian Class 1 with increased variance
    class1_points = np.random.normal(loc=[0.5, 0.5],
                                     scale=[1.5, 1.5],
                                     size=(n_class1, 2))

    X = np.vstack([class0_points, class1_points])
    y = np.hstack([np.zeros(n_class0), np.ones(n_class1)])

    return X, y
 X_mismatch, y_mismatch = generate_mismatched_data()

svm_mismatch = SVC(kernel='rbf', C=10, random_state=42, probability=True)
svm_mismatch_metrics, svm_mismatch_accuracies =
evaluate_classifier(X_mismatch, y_mismatch, svm_mismatch)
print("Nonlinear SVM (Mismatched Data) Metrics:", svm_mismatch_metrics)
```

Part 5: High variance

Generate the data sets as in Part 3C (i.e., no mismatch between training and test data sets). Reduce the number of samples in the training set to 60 samples in Class 0 and 20 samples in Class 1. Implement a nonlinear SVM. Train using the training data. Use 4-fold cross validation and the test data to report the accuracy, precision, recall, AUROC metrics. What is the variance of the accuracy across the 4 runs?

```
X_reduced, y_reduced = generate_data_part0(n_class0=60, n_class1=20)


svm_reduced = SVC(kernel='rbf', C=10, random_state=42, probability=True)

svm_reduced_metrics, svm_reduced_accuracies = evaluate_classifier(X_reduced,
y_reduced, svm_reduced)

print("Nonlinear SVM (Reduced Training Set) Metrics:", svm_reduced_metrics)
```