Assignment Week 4

Student Name: Patrick Farmer Student Number: 20331828

Date: 2024-07-10 Dataset1: 12-24-12-0 Dataset2: 12-12-12-0

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

I(a)

Appendix

I(a)

```
for index in range(1, 3):
    data = pd.read_csv(f'week4_{index}.csv', skiprows=1)
    X = data.iloc[:, :-1].values
    y = data.iloc[:, -1].values

    if np.any(pd.isnull(y)):
        raise ValueError("The target variable y contains NaN values. Please clean the data before pro

    plt.figure(figsize=(10, 6))
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
    plt.title(f'Scatter plot of the data (week4_{index}.csv)')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    i_string = 'i' * index
    plt.savefig(f'Images/{i_string}(a(1)).png')
    plt.close()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    log_reg_param_grid = {'poly__degree': list(range(1, 6)), 'log_reg__C': np.logspace(-3, 3, 7)}
    log_reg_runner = model_runner.ModelRunner('log_reg', log_reg_param_grid, use_pipeline=True, i_str

    log_reg_runner.perform_grid_search(X_train, y_train)
    log_reg_runner.train_model(X_train, y_train)
    y_prob_log_reg = log_reg_runner.evaluate_model(X_test, y_test)

    log_reg_runner.plot_decision_boundary(X_train, y_train, index)
    log_reg_runner.plot_cross_validation_results(index)

    log_reg_runner.plot_roc_curve(y_test, y_prob_log_reg, index)

class ModelRunner:
    def __init__(self, model_name, param_grid, use_pipeline=False, i_string=''):
        self.model_name = model_name
        self.param_grid = param_grid
        self.use_pipeline = use_pipeline
        self.grid_search = None
        self.final_model = None
        self.i_string = i_string

    def perform_grid_search(self, X_train, y_train):
        if self.use_pipeline:
            pipeline = Pipeline([
                ('poly', PolynomialFeatures()),
                (self.model_name, LogisticRegression(max_iter=1000, penalty='l2') if self.model_name
            ])
            self.grid_search = GridSearchCV(pipeline, self.param_grid, cv=5, scoring='neg_mean_square
        else:
            model = LogisticRegression(max_iter=1000, penalty='l2') if self.model_name == 'log_reg' e
            self.grid_search = GridSearchCV(model, self.param_grid, cv=5, scoring='neg_mean_squared_e
        self.grid_search.fit(X_train, y_train)
        return self.grid_search

    def train_model(self, X_train, y_train):
        self.final_model = self.grid_search.best_estimator_
        self.final_model.fit(X_train, y_train)

    def plot_decision_boundary(self, X, y, index):
        xx, yy = np.meshgrid(np.linspace(X[:, 0].min() - 1, X[:, 0].max() + 1, 100),
                             np.linspace(X[:, 1].min() - 1, X[:, 1].max() + 1, 100))
        Z = self.final_model.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)

        plt.figure(figsize=(10, 6))
        plt.contourf(xx, yy, Z, alpha=0.3)
        plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
        plt.title(f'Decision boundary of {self.model_name} (week4_{index}.csv)')
        plt.xlabel('Feature 1')
        plt.ylabel('Feature 2')
        if self.model_name == 'log_reg':
            plt.savefig(f'Images/{self.i_string}(a(2)).png')
        else:
            plt.savefig(f'Images/{self.i_string}(b(2)).png')
        plt.close()

    def plot_cross_validation_results(self, index):
        results = self.grid_search.cv_results_
        mean_test_scores = -results['mean_test_score']
        std_test_scores = results['std_test_score']

        plt.figure(figsize=(10, 6))
        if self.model_name == 'log_reg':
            poly_degrees = self.param_grid['poly__degree']
            C_values = self.param_grid['log_reg__C']
            for degree in poly_degrees:
                mask = results['param_poly__degree'] == degree
                plt.errorbar(C_values, mean_test_scores[mask], yerr=std_test_scores[mask], label=f'De
            plt.xscale('log')
            plt.xlabel('C value (log scale)')
        else:
            k_values = self.param_grid['n_neighbors']
            plt.errorbar(k_values, mean_test_scores, yerr=std_test_scores, fmt='o-', capsize=5)
            plt.xlabel('Number of Neighbors (k)')

        plt.ylabel('Mean cross-validation mean squared error')
        plt.title(f'Cross-validation mean squared error for different parameters (week4_{index}.csv)'
        plt.legend()
        if self.model_name == 'log_reg':
            plt.savefig(f'Images/{self.i_string}(a(3)).png')
        else:
            plt.savefig(f'Images/{self.i_string}(b(3)).png')
        plt.close()
```

I(b)

```
for index in range(1, 3):
    knn_param_grid = {'n_neighbors': list(range(1, 21))}
    knn_runner = model_runner.ModelRunner('knn', knn_param_grid, i_string=i_string)

    knn_runner.perform_grid_search(X_train, y_train)
    knn_runner.train_model(X_train, y_train)
    y_prob_knn = knn_runner.evaluate_model(X_test, y_test)

    knn_runner.plot_decision_boundary(X_train, y_train, index)
    knn_runner.plot_cross_validation_results(index)
```

I(c)

```
log_reg_runner.plot_confusion_matrix(y_test, index)
knn_runner.plot_confusion_matrix(y_test, index)
```

I(d)

```
log_reg_runner.plot_roc_curve(y_test, y_prob_log_reg, index)
knn_runner.plot_roc_curve(y_test, y_prob_knn, index)
```

```
def plot_roc_curve(self, y_test, y_prob, index):
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    auc_score = auc(fpr, tpr)

    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, label=f'{self.model_name} (AUC = {auc_score:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve (week4_{index}.csv)')
    plt.legend(loc='best')
    plt.savefig(f'Images/{self.i_string}(d).png')
    plt.close()
```

II(a)

The code is generic and runs for other dataset automatically

```
for index in range(1, 3):
```