NAME: Alex Friedrichsen
DATE: 4/6/2023
Homework: 8
CS 395

Problem 1

I created a data pipeline that was able to handle data from all four datasets. A function read in the dataset from CSV or scikit directly, and split into X and y. For the heart disease data, further preprocessing was required to one-hot encode several of the columns. Next, the X and y date were passed into my pipeline function, which performed a train/test split into x test, y test, x train, and y train. Next, missing values were imputed using the scikit SimpleImputer, as decision tree doesn't support missing values in the data. Next, a C4.5 decision tree, random forest, naive Bayes, and Support Vector Machine Model were created for all four data sets. After the models were each returned, I calculated the confusion matrices and associated metrics for each model, as well as ROC curves for the binary models (*cannot create ROC for non-binary classifiers*). Visualizations were saved and metrics saved to a .txt file. **Included figures and metrics in report are only for the wine dataset. All other data set figures and metrics are included in the attached zipped folder.**

model	sensitivity (recall)	specificity	balanced accuracy	precision	F1 Score
C4.5 Decision Tree	.947	.5	.333	.886	.975*
Naive Bayes	1	.525	.424	1	1*
Random Forest	1	.525	.424	1	1*
SVM	1	.513	.424	1	.975*

Table 1: Metrics for each model for the wine dataset

The calculated metrics indicate that several of the models had very high sensitivity, precision, and F1 score. The specificity is lower, and accordingly the balance accuracy is lower as well. All the models performed similarly. When we look at the confusion matrices for each model, we can see by the diagonal opacity that the models was classify most (or all) tuples correctly. Thusly, our true positive score was very high.

For the wine dataset, we do not have ROC curves as it is a multiclass (3-class) classification problem.

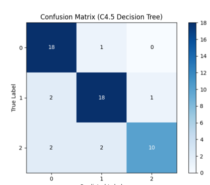

Figure 1: Wine dataset decision tree confusion matrix

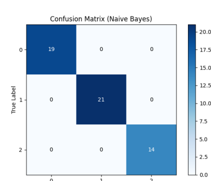

Figure 2: Wine dataset naive Bayes confusion matrix

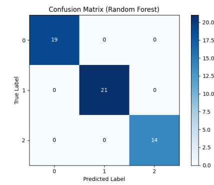

Figure 3: Wine dataset random forest confusion matrix

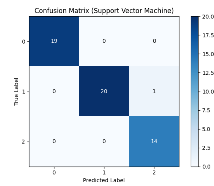

Figure 4: Wine dataset support vector machine confusion matrix

HW8.py

```python
#Disclaimer: parts or the whole of functions were generated using generative AI tools, then checked and modified for precision.

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, roc_curve, auc
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.datasets import load_wine, load_breast_cancer, load_iris

def train_test_split_data(X, y, test_size=0.3, random_state=42):
    """Split the dataset into training and testing sets."""
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)
    return X_train, X_test, y_train, y_test

def c45_decision_tree(X_train, y_train, X_test):
    """Train and evaluate a C4.5 decision tree model."""
    model = DecisionTreeClassifier(criterion='entropy')
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return y_pred, model

def naive_bayes(X_train, y_train, X_test):
    """Train and evaluate a naive Bayes model."""
    model = GaussianNB()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return y_pred, model

def random_forest(X_train, y_train, X_test):
    """Train and evaluate a random forest model."""
    model = RandomForestClassifier(n_estimators=100)
    model.fit(X_train, y_train)
```

```
    y_pred = model.predict(X_test)
    return y_pred, model

def support_vector_machine(X_train, y_train, X_test):
    """Train and evaluate a support vector machine model."""
    model = SVC(kernel='linear')
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return y_pred, model

def calculate_metrics(y_true, y_pred):
    """Calculate the confusion matrix and various classification metrics."""
    cm = confusion_matrix(y_true, y_pred)
    if cm.shape == (2, 2):
        tn, fp, fn, tp = cm.ravel()
        sensitivity = tp / (tp + fn)
        specificity = tn / (tn + fp)
        balanced_accuracy = (sensitivity + specificity) / 2
        precision = tp / (tp + fp)
        recall = sensitivity
        f1_score = 2 * precision * recall / (precision + recall)
        return cm, sensitivity, specificity, balanced_accuracy, precision, recall, f1_score
    elif cm.shape == (3, 3):
        # Calculate metrics for the three-class case
        tp0 = cm[0, 0]
        tp1 = cm[1, 1]
        tp2 = cm[2, 2]
        fp0 = np.sum(cm[1:, 0])
        fp1 = np.sum(np.concatenate((cm[:1, 1], cm[2:, 1])))
        fp2 = np.sum(cm[:2, 2])
        fn0 = np.sum(cm[0, 1:])
        fn1 = np.sum(np.concatenate((cm[0, :1], cm[0, 2:])))
        fn2 = np.sum(cm[:2, 0])
        tn0 = np.sum(cm[1:, 1:])  # includes class 1, class 2
        tn1 = np.sum(np.concatenate((cm[:1, :1], cm[:1, 2:], cm[2:, :1], cm[2:, 2:])))
        tn2 = np.sum(cm[:2, :2])
        sensitivity0 = tp0 / (tp0 + fn0)
        sensitivity1 = tp1 / (tp1 + fn1)
        sensitivity2 = tp2 / (tp2 + fn2)
        specificity0 = tn0 / (tn0 + fp0)
        specificity1 = tn1 / (tn1 + fp1)
        specificity2 = tn2 / (tn2 + fp2)
        balanced_accuracy = (sensitivity0 + sensitivity1 + sensitivity2) / 3
        precision0 = tp0 / (tp0 + fp0)
        precision1 = tp1 / (tp1 + fp1)
        precision2 = tp2 / (tp2 + fp2)
        precision = (precision0 + precision1 + precision2) / 3
        recall = (sensitivity0 + sensitivity1 + sensitivity2) / 3
        f1_score = 2 * precision * recall / (precision + recall)
        return cm, sensitivity0, sensitivity1, sensitivity2, specificity0, specificity1, specificity2, balanced_accuracy, precis
    else:
        print('Error: Confusion matrix has unexpected shape')
        return None, None, None, None, None, None, None, None, None, None, None


def plot_confusion_matrix(cm, model_name, output_dir):
    """Plot the confusion matrix."""
    plt.figure()
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(f'Confusion Matrix ({model_name})')
    plt.colorbar()
    tick_marks = np.arange(cm.shape[0])
    plt.xticks(tick_marks, tick_marks)
    plt.yticks(tick_marks, tick_marks)
    plt.xlabel('Predicted Label')
```

```
    plt.ylabel('True Label')
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'), ha="center", va="center", color="white" if cm[i, j] > cm.max() / 2 else "black
    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, 'confusion_matrix.png'))
    plt.close()

def plot_roc_curve(y_true, y_score, model_name, output_dir):
    """Plot the ROC curve."""
    fpr, tpr, _ = roc_curve(y_true, y_score)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC Curve (AUC = {roc_auc:.3f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve ({model_name})')
    plt.legend(loc="lower right")
    plt.savefig(os.path.join(output_dir, 'roc_curve.png'))
    plt.close()


def pipeline(X, y, dataset_name, output_dir='output'):
    """Run the classification pipeline on the specified dataset."""
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    # Create a separate output folder for this dataset
    dataset_output_dir = os.path.join(output_dir, dataset_name)
    if not os.path.exists(dataset_output_dir):
        os.makedirs(dataset_output_dir)

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split_data(X, y)

    # Impute missing values
    imputer = SimpleImputer(strategy='mean')
    X_train = imputer.fit_transform(X_train)
    X_test = imputer.transform(X_test)

    # Train and evaluate the C4.5 decision tree model
    y_pred, model = c45_decision_tree(X_train, y_train, X_test)
    cm, *metrics = calculate_metrics(y_test, y_pred)
    if cm is not None:
        model_name = 'C4.5 Decision Tree'
        model_output_dir = os.path.join(output_dir, dataset_name, model_name)
        if not os.path.exists(model_output_dir):
            os.makedirs(model_output_dir)
        with open(os.path.join(model_output_dir, 'metrics.txt'), 'w') as f:
            f.write(f'Confusion Matrix ({model_name}):\n{cm}\n\n')
            f.write(f'Sensitivity: {metrics[0]:.3f}\n')
            f.write(f'Specificity: {metrics[1]:.3f}\n')
            f.write(f'Balanced Accuracy: {metrics[2]:.3f}\n')
            f.write(f'Precision: {metrics[3]:.3f}\n')
            f.write(f'Recall: {metrics[4]:.3f}\n')
            f.write(f'F1 Score: {metrics[5]:.3f}\n')
        y_score = model.predict_proba(X_test)[:, 1]
        plot_confusion_matrix(cm, model_name, model_output_dir)
        if cm.shape == (2, 2):
            plot_roc_curve(y_test, y_score, model_name, model_output_dir)

```

```python
    # Train and evaluate the naive Bayes model
    y_pred, model = naive_bayes(X_train, y_train, X_test)
    cm, *metrics = calculate_metrics(y_test, y_pred)
    if cm is not None:
        model_name = 'Naive Bayes'
        model_output_dir = os.path.join(output_dir, dataset_name, model_name)
        if not os.path.exists(model_output_dir):
            os.makedirs(model_output_dir)
        with open(os.path.join(model_output_dir, 'metrics.txt'), 'w') as f:
            f.write(f'Confusion Matrix ({model_name}):\n{cm}\n\n')
            f.write(f'Sensitivity: {metrics[0]:.3f}\n')
            f.write(f'Specificity: {metrics[1]:.3f}\n')
            f.write(f'Balanced Accuracy: {metrics[2]:.3f}\n')
            f.write(f'Precision: {metrics[3]:.3f}\n')
            f.write(f'Recall: {metrics[4]:.3f}\n')
            f.write(f'F1 Score: {metrics[5]:.3f}\n')
        y_score = model.predict_proba(X_test)[:, 1]
        plot_confusion_matrix(cm, model_name, model_output_dir)
        if cm.shape == (2, 2):
            plot_roc_curve(y_test, y_score, model_name, model_output_dir)

    # Train and evaluate the random forest model
    y_pred, model = random_forest(X_train, y_train, X_test)
    cm, *metrics = calculate_metrics(y_test, y_pred)
    if cm is not None:
        model_name = 'Random Forest'
        model_output_dir = os.path.join(output_dir, dataset_name, model_name)
        if not os.path.exists(model_output_dir):
            os.makedirs(model_output_dir)
        with open(os.path.join(model_output_dir, 'metrics.txt'), 'w') as f:
            f.write(f'Confusion Matrix ({model_name}):\n{cm}\n\n')
            f.write(f'Sensitivity: {metrics[0]:.3f}\n')
            f.write(f'Specificity: {metrics[1]:.3f}\n')
            f.write(f'Balanced Accuracy: {metrics[2]:.3f}\n')
            f.write(f'Precision: {metrics[3]:.3f}\n')
            f.write(f'Recall: {metrics[4]:.3f}\n')
            f.write(f'F1 Score: {metrics[5]:.3f}\n')
        y_score = model.predict_proba(X_test)[:, 1]
        plot_confusion_matrix(cm, model_name, model_output_dir)
        if cm.shape == (2, 2):
            plot_roc_curve(y_test, y_score, model_name, model_output_dir)

    # Train and evaluate the support vector machine model
    y_pred, model = support_vector_machine(X_train, y_train, X_test)
    cm, *metrics = calculate_metrics(y_test, y_pred)
    if cm is not None:
        model_name = 'Support Vector Machine'
        model_output_dir = os.path.join(output_dir, dataset_name, model_name)
        if not os.path.exists(model_output_dir):
            os.makedirs(model_output_dir)
        with open(os.path.join(model_output_dir, 'metrics.txt'), 'w') as f:
            f.write(f'Confusion Matrix ({model_name}):\n{cm}\n\n')
            f.write(f'Sensitivity: {metrics[0]:.3f}\n')
            f.write(f'Specificity: {metrics[1]:.3f}\n')
            f.write(f'Balanced Accuracy: {metrics[2]:.3f}\n')
            f.write(f'Precision: {metrics[3]:.3f}\n')
            f.write(f'Recall: {metrics[4]:.3f}\n')
            f.write(f'F1 Score: {metrics[5]:.3f}\n')
        y_score = model.decision_function(X_test)
        plot_confusion_matrix(cm, model_name, model_output_dir)
        if cm.shape == (2, 2):
            plot_roc_curve(y_test, y_score, model_name, model_output_dir)

def load_data(filename):
    """Load the heart disease dataset."""
```

```python
    df = pd.read_csv(filename, usecols=["age", "sex", "cp", "trestbps", "chol", "fbs", "restecg", "thalch", "exang", "oldpeak",

    # Replace the target variable values
    df["num"] = df["num"].replace({1: 0, 2: 1, 3: 1, 4: 1})

    # One-hot encode categorical variables
    df = pd.get_dummies(df, columns=["sex", "cp", "fbs", "restecg", "exang", "slope", "ca", "thal"])

    # Split the data into X and y
    X = df.drop("num", axis=1)
    y = df["num"]

    return X, y

def main():
    sklearn_datasets = [('wine', load_wine()), ('breast_cancer', load_breast_cancer()), ('iris', load_iris())]
    for dataset_name, dataset in sklearn_datasets:
        X = dataset.data
        y = dataset.target
        pipeline(X, y, dataset_name)
    X, y = load_data('heart_disease_uci.csv')
    pipeline(X, y, 'heart_disease_uci')


if __name__ == '__main__':
    main()
```