Intro to ML - Exercise 3 Naama Avni 208523456

Outputs and solutions

Question 1

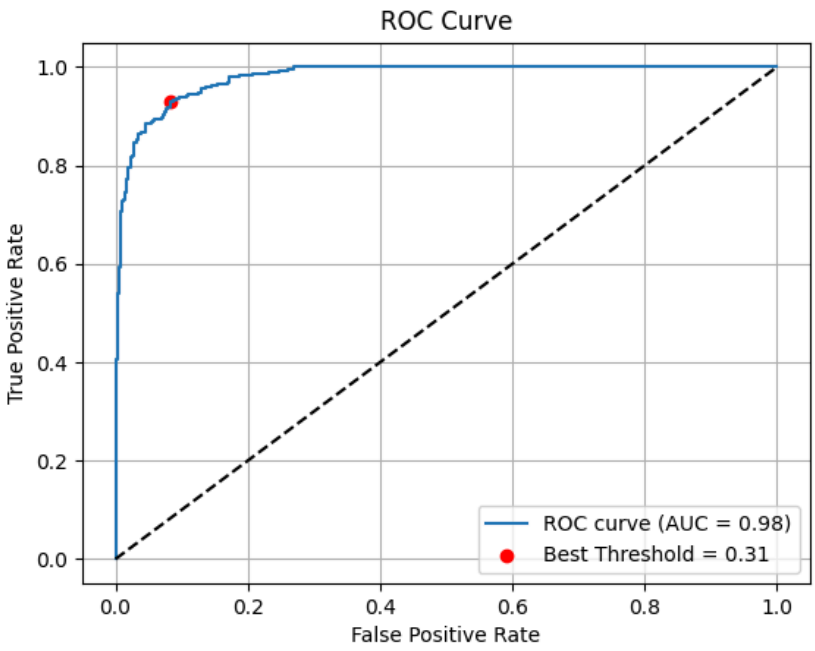
No output - solution in the code

Question 2

Model Score: 0.84
Model weights:
Too many weights to print them all, example of 6 weights was printed
[-7.38065369e-01 -3.26496472e-02 -7.16581154e-02 ... 1.74701561e-04
0.00000000e+00 3.80863506e-05]

Question 3

ROC CURVE:



The best threshold as determined by the point on the ROC curve that maximizes Youden's J statistic is 0.31.
This threshold used in the predict method and plotted in the ROC curve.

Intro to ML - Exercise 3 Naama Avni 208523456

Question 4

OVR model Score: 0.87
Model weights per class:
[[1.70476003e-01 2.54659015e-01 9.29626527e-01 -1.47018452e+00
  -6.57081405e-01]
 [2.62926270e-02 -9.98563309e-04 -6.09605533e-01 3.24313910e-01
  -1.15271197e-01]
 [-2.97949635e-01 -6.81820563e-01 -7.43236966e-01 1.11867453e+00
   7.63564251e-01]]

Full output

Question 2
Model weights: [-7.38065369e-01 -3.26496472e-02 -7.16581154e-02 ...
1.74701561e-04
  0.00000000e+00 3.80863506e-05]
Score: 0.84

Question 3
Best Threshold: 0.31
The best threshold as determined by the point on the ROC curve that maximizes Youden's J statistic is 0.31.
This threshold used in the predict method and plotted in the ROC curve.

Question 4
OVR Model weights: [[1.70476003e-01 2.54659015e-01 9.29626527e-01
-1.47018452e+00
  -6.57081405e-01]
 [2.62926270e-02 -9.98563309e-04 -6.09605533e-01 3.24313910e-01
  -1.15271197e-01]
 [-2.97949635e-01 -6.81820563e-01 -7.43236966e-01 1.11867453e+00
   7.63564251e-01]]
OVR Score: 0.87

Intro to ML - Exercise 3 Naama Avni 208523456

Full code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.feature_extraction.text import TfidfVectorizer



class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self._learning_rate = learning_rate
        self._num_iterations = num_iterations
        self.weights = None

    def fit(self, X, y):
        """
        Fit the model using the provided data.
        """
        # Add intercept term to X
        X = np.c_[np.ones(X.shape[0]), X]

        # Initialize weights
        self.weights = np.zeros(X.shape[1])

        # Gradient descent loop
        for _ in range(self._num_iterations):
            predictions = self._sigmoid(np.dot(X, self.weights))
            errors =  y - predictions
            gradient = np.dot(X.T, errors) / len(y)
            self.weights += self._learning_rate * gradient


    def predict(self, X):
        """
        Predict the class labels for the provided data.
        """
        # Add intercept term to X
```

```
        X = np.c_[np.ones(X.shape[0]), X]

        # Compute predictions
        probabilities = self._sigmoid(np.dot(X, self.weights))
        # Convert probabilities to class labels based on the best
threshold
        return (probabilities >= 0.31).astype(int)

    def predict_proba(self, X):
        """
        Predict the class probabilities for the provided data.
        """
        # Add intercept term to X
        X = np.c_[np.ones(X.shape[0]), X]

        # Compute probabilities
        return self._sigmoid(np.dot(X, self.weights))

    def score(self, X, y):
        """
        Compute the accuracy of the model on the provided data.
        """
        predictions = self.predict(X)
        return round(np.mean(predictions == y), 2)

    def _sigmoid(self, z):
        """
        Compute the sigmoid function.
        """
        return 1 / (1 + np.exp(-z))

    def plot_roc_auc(self, X, y):
        """
        Plot the ROC curve and compute the AUC score.
        Works only for binary classification.
        """
        probabilities = self.predict_proba(X)

        fpr, tpr, thresholds = roc_curve(y, probabilities)
        roc_auc = roc_auc_score(y, probabilities)
```

Intro to ML - Exercise 3 Naama Avni 208523456

```
        # Compute Youden's J statistic
        j_scores = tpr - fpr
        best_idx = np.argmax(j_scores)
        best_threshold = thresholds[best_idx]
        print(f"Best Threshold: {best_threshold:.2f}")

        plt.figure()
        plt.plot(fpr, tpr, label='ROC curve (AUC = %0.2f)' %
roc_auc)
        plt.plot([0, 1], [0, 1], 'k--')
        plt.scatter(fpr[best_idx], tpr[best_idx], color='red',
label=f'Best Threshold = {best_threshold:.2f}')
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('ROC Curve')
        plt.legend(loc='lower right')
        plt.grid()
        plt.show()


class LogisticRegressionOVR:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.weights = None
        self._n_classes = None
        self._learning_rate = learning_rate
        self._num_iterations = num_iterations


    def fit(self, X, y):
        X = np.c_[np.ones(X.shape[0]), X]  # Add intercept
        self._n_classes = np.unique(y).size
        self.weights = np.zeros((self._n_classes, X.shape[1]))

        for class_idx in range(self._n_classes):
            binary_y = (y == class_idx).astype(int)
            weights = np.zeros(X.shape[1])

            for _ in range(self._num_iterations):
                predictions = self._sigmoid(np.dot(X, weights))
                errors = binary_y - predictions
                gradient = np.dot(X.T, errors) / len(y)
```

Intro to ML - Exercise 3 Naama Avni 208523456

```
                weights += self._learning_rate * gradient

            self.weights[class_idx] = weights

    def predict(self, X):
        probs = self.predict_proba(X)
        return np.argmax(probs, axis=1)

    def predict_proba(self, X):
        X = np.c_[np.ones(X.shape[0]), X]
        logits = np.dot(X, self.weights.T)  # shape: (n_samples,
n_classes)
        probs = self._sigmoid(logits)
        return probs

    def score(self, X, y):
        predictions = self.predict(X)
        return np.mean(predictions == y)

    def _sigmoid(self, z):
        z = np.clip(z, -500, 500)
        return 1 / (1 + np.exp(-z))



def main_question_2(X_train, y_train, X_test, y_test):
    print("\nQuestion 2")
    model = LogisticRegression()
    model.fit(X_train, y_train)
    print("Model weights:", model.weights)
    score = model.score(X_test, y_test)
    print("Score:", score)
    return model

def main_question_3(model, X_test, y_test):
    print("\nQuestion 3")
    model.plot_roc_auc(X_test, y_test)
    print("The best threshold as determined by the point on the
ROC curve that maximizes Youden's J statistic is 0.31.")
    print("This threshold used in the predict method and plotted
in the ROC curve.")
```

Intro to ML - Exercise 3 Naama Avni 208523456

```
def main_question_4():
    print("\nQuestion 4")
    iris = datasets.load_iris()
    X_iris = iris.data
    y_iris = iris.target
    X_train_iris, X_test_iris, y_train_iris, y_test_iris =
train_test_split(X_iris, y_iris, test_size=0.2, random_state=42)
    model_ovr = LogisticRegressionOVR()
    model_ovr.fit(X_train_iris, y_train_iris)
    print("OVR Model weights:", model_ovr.weights)
    score_ovr = model_ovr.score(X_test_iris, y_test_iris)
    print("OVR Score:", round(score_ovr, 2))

if __name__ == "__main__":
    df = pd.read_csv("./spam_ham_dataset.csv")
    df = df[["text", "label_num"]]

    # Convert text to vector
    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(df['text']).toarray()
    print("X shape:", X.shape)
    y = df['label_num'].values

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Question 2
    model = main_question_2(X_train, y_train, X_test, y_test)

    # Question 3
    main_question_3(model, X_test, y_test)

    # Question 4
    main_question_4()
```