IE406 Machine Learning - Assignment 7

202201140 | Harsh Gajjar

1. Question 1

Task: Implement a simple artificial neural network (ANN) to classify the Iris dataset into its three species: Setosa, Versicolour, and Virginica. Evaluate the model's performance using accuracy, precision, recall, and F1-score.

Requirements:

- *Use TensorFlow/Keras or PyTorch to build the ANN.*
- *Split the dataset into training and testing sets.*
- *Use at least one hidden layer in the neural network.*
- *Apply the softmax function in the output layer for multi-class classifica-tion.*
- *After training the model, compute accuracy, precision, recall, and F1-score.*

Hints:

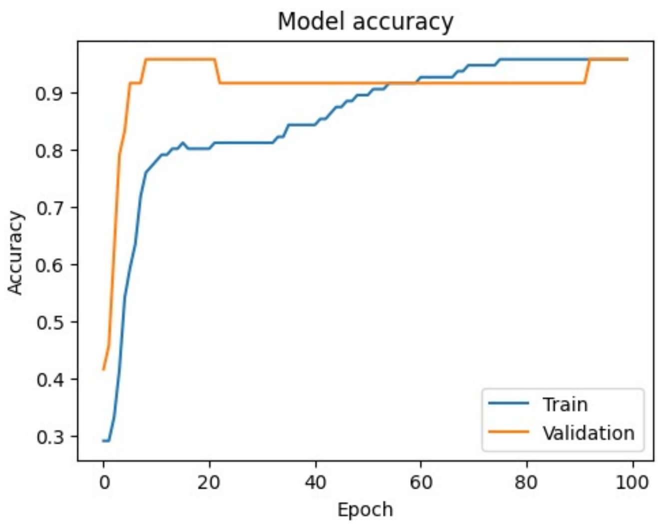
- *Use classification report from sklearn.metrics to compute preci-sion, recall, and F1-score.*
- *Use a small number of neurons in the hidden layer since the dataset is small.*
- *The output layer should have 3 neurons for the 3 classes.*

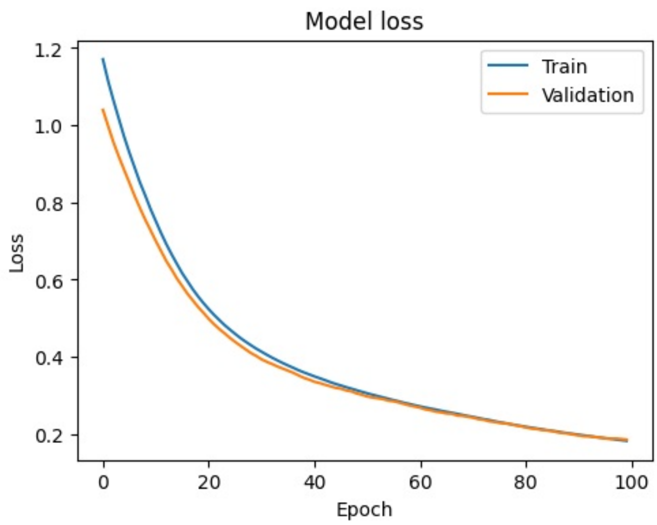
In [65]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split the dataset 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)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the ANN model
input_layer = Input(shape=(X_train.shape[1],))
hidden_layer = Dense(10, activation='relu')(input_layer)
output_layer = Dense(3, activation='softmax')(hidden_layer)
model = Model(inputs=input_layer, outputs=output_layer)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_split=0.2, verbose=0)

# Evaluate the model on the testing set
y_pred = np.argmax(model.predict(X_test), axis=1)

# Calculate accuracy, precision, recall, and F1-score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=data.target_names))

# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
```

```python
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')

plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1-score: 1.00

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```





2. Question 2

Task: Train a neural network classifier on the Iris dataset to predict the species of a flower based on its features (sepal length, sepal width, petal length, petal width). Evaluate the model using a confusion matrix and calculate accuracy, precision, recall, and F1-score.

Requirements:

- *Build a neural network using TensorFlow/Keras or PyTorch with at least one hidden layer.*
- *Use the ReLU activation function for hidden layers and softmax for the output layer.*
- *After training, create a confusion matrix to visualize the performance.*
- *Compute accuracy, precision, recall, and F1-score for each class (Setosa, Versicolour, Virginica).*

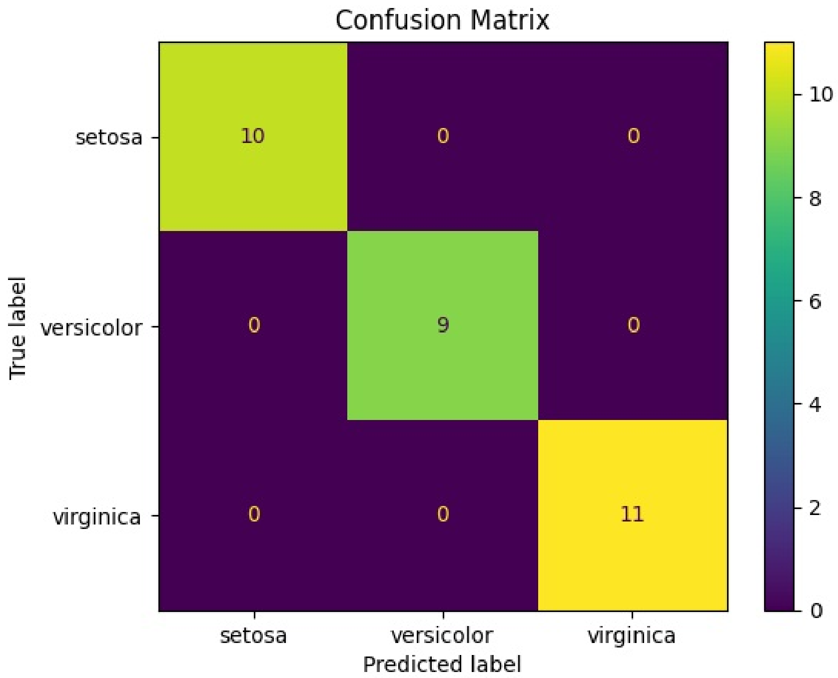
Hints:

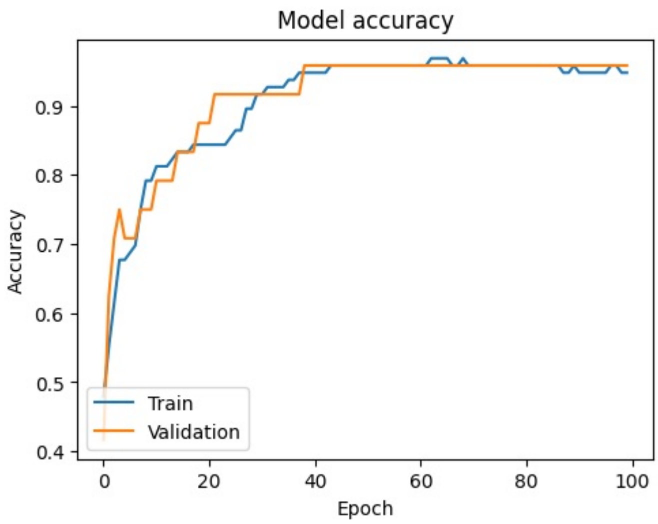
- *Use ConfusionMatrixDisplay from sklearn.metrics for visualization.*
- *Use the Adam optimizer to train the model.*
- *Apply cross-entropy loss function for multi-class classification.*

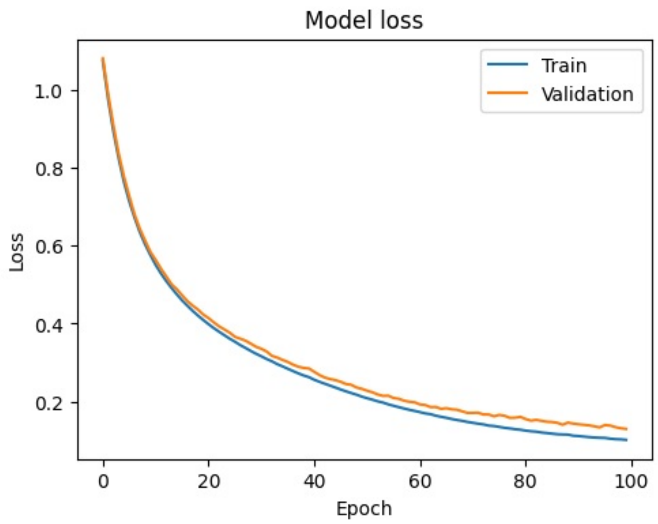
In [66]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, Con
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split the dataset 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)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the ANN model
input_layer = Input(shape=(X_train.shape[1],))
hidden_layer = Dense(32, activation='relu')(input_layer)
output_layer = Dense(3, activation='softmax')(hidden_layer)
model = Model(inputs=input_layer, outputs=output_layer)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_split=0.2, verbose=0)

# Evaluate the model on the testing set
y_pred = np.argmax(model.predict(X_test), axis=1)

# Calculate accuracy, precision, recall, and F1-score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-score: {f1:.2f}")

print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=data.target_names))

# Plot confusion matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, display_labels=data.target_names)
plt.title('Confusion Matrix')
plt.show()

# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')

plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1-score: 1.00

Classification Report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```







3. Question 3

Task: Perform multi-class classification on the Iris dataset using a neural network. Implement cross-validation to evaluate the model's performance and report accuracy, precision, recall, and F1-score.

Requirements:

- *Use TensorFlow/Keras or PyTorch to build the ANN.*
- *Apply k-fold cross-validation (e.g., 5-fold) to evaluate the model's performance.*
- *Use at least one hidden layer with an appropriate activation function.*
- *After performing cross-validation, compute the average accuracy, precision, recall, and F1-score across all folds.*

Hints:

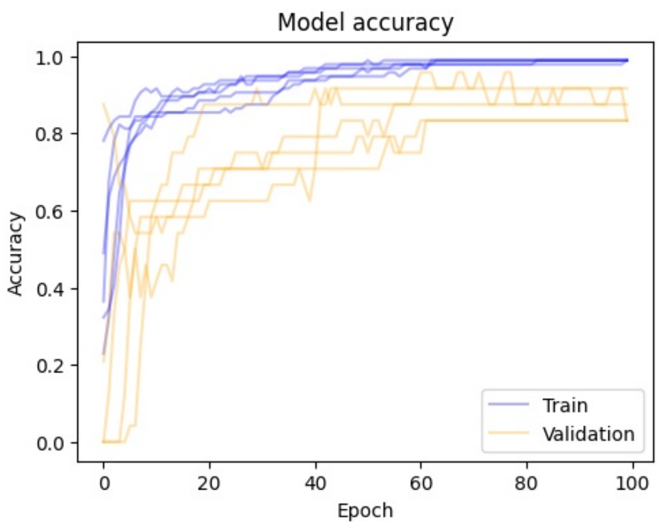
- *Use KFold or StratifiedKFold from sklearn.model_selection for cross-validation.*
- *Use classification_report from sklearn.metrics to compute metrics on each fold.*
- *Aggregate the results of each fold to calculate average performance.*

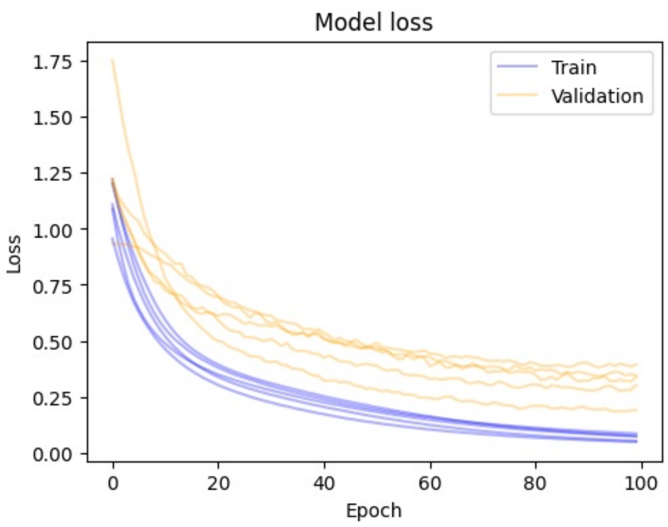
In [61]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Standardize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Define the model creation function
def create_model():
    input_layer = Input(shape=(X.shape[1],))
    hidden_layer = Dense(32, activation='relu')(input_layer)
    output_layer = Dense(3, activation='softmax')(hidden_layer)
    model = Model(inputs=input_layer, outputs=output_layer)
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

accuracy_list = []
precision_list = []
recall_list = []
f1_list = []
history_list = []

# Perform cross-validation
for train_index, test_index in kf.split(X, y):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model = create_model()
    history = model.fit(X_train, y_train, epochs=100, batch_size=10, validation_split=0.2, verbose=0)
    history_list.append(history)

    y_pred = np.argmax(model.predict(X_test), axis=1)

    accuracy_list.append(accuracy_score(y_test, y_pred))
    precision_list.append(precision_score(y_test, y_pred, average='weighted'))
    recall_list.append(recall_score(y_test, y_pred, average='weighted'))
    f1_list.append(f1_score(y_test, y_pred, average='weighted'))

# Calculate average metrics
average_accuracy = np.mean(accuracy_list)
average_precision = np.mean(precision_list)
average_recall = np.mean(recall_list)
average_f1 = np.mean(f1_list)

print(f"Average Accuracy: {average_accuracy:.2f}")
print(f"Average Precision: {average_precision:.2f}")
print(f"Average Recall: {average_recall:.2f}")
print(f"Average F1-score: {average_f1:.2f}")

print("\nClassification Report for the last fold:\n", classification_report(y_test, y_pred, target_names=data.ta

# Plot average training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
for history in history_list:
    plt.plot(history.history['accuracy'], color='blue', alpha=0.3)
    plt.plot(history.history['val_accuracy'], color='orange', alpha=0.3)
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='lower right')

# Plot average training & validation loss values
plt.subplot(1, 2, 2)
```

```
for history in history_list:
    plt.plot(history.history['loss'], color='blue', alpha=0.3)
    plt.plot(history.history['val_loss'], color='orange', alpha=0.3)
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')

plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
Average Accuracy: 0.95
Average Precision: 0.96
Average Recall: 0.95
Average F1-score: 0.95

Classification Report for the last fold:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.83      1.00      0.91        10
   virginica       1.00      0.80      0.89        10

    accuracy                           0.93        30
   macro avg       0.94      0.93      0.93        30
weighted avg       0.94      0.93      0.93        30
```





4. Logistic Regression from Scratch

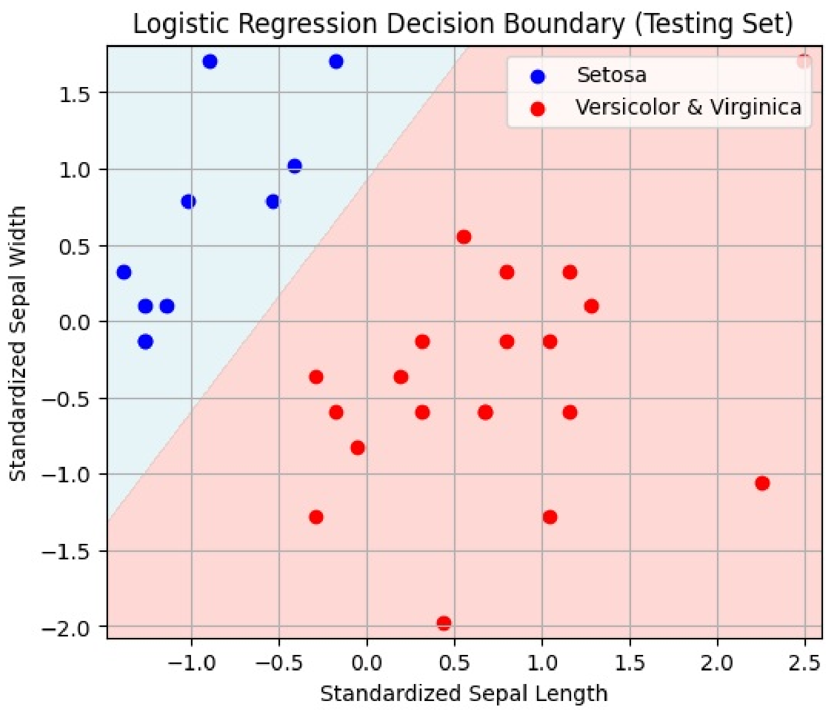
Consider the Iris dataset with sepal length and sepal width as the attributes, and Iris-Setosa as class c1, and the Virginica as class c2. There are n1 = 50 points in c1 and n2 = 100 points in c2. Task: Train the logistic regression model and find the separating decision boundary and plot it. Do it from scratch without using a library.

In [59]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Load the Iris dataset
data = load_iris()
X = data.data[:, :2]  # using sepal length and sepal width
y = data.target

# Filter the classes 0 (Setosa), 1 (Versicolor), and 2 (Virginica)
X = X[(y == 0) | (y == 1) | (y == 2)]
y = y[(y == 0) | (y == 1) | (y == 2)]

# Convert classes: c1: {Setosa (0)} -> 0, and c2: {Versicolor (1) & Virginica (2)} -> 1
y = np.where(y == 0, 0, 1)

scaler = StandardScaler()
X = scaler.fit_transform(X)

# Add a bias term to the feature matrix
X = np.hstack([np.ones((X.shape[0], 1)), X])
```

```python
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def cost_function(X, y, theta):
    h = sigmoid(X @ theta)
    m = len(y)
    cost = -(1/m) * (y.T @ np.log(h) + (1 - y).T @ np.log(1 - h))
    return cost

def gradient_descent(X, y, theta, learning_rate, iterations):
    m = len(y)
    cost_history = np.zeros(iterations)

    for i in range(iterations):
        gradient = (1/m) * X.T @ (sigmoid(X @ theta) - y)
        theta -= learning_rate * gradient
        cost_history[i] = cost_function(X, y, theta)

    return theta, cost_history

# Split the dataset 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)

theta = np.zeros(X_train.shape[1])
learning_rate = 0.1
iterations = 1000

# Train the model
theta, cost_history = gradient_descent(X_train, y_train, theta, learning_rate, iterations)

# Accuracy on training set
y_pred_train = (sigmoid(X_train @ theta) >= 0.5).astype(int)
accuracy_train = (y_pred_train == y_train).mean() * 100
print(f"Training Accuracy: {accuracy_train:.2f}%")

# Accuracy on testing set
y_pred_test = (sigmoid(X_test @ theta) >= 0.5).astype(int)
accuracy_test = (y_pred_test == y_test).mean() * 100
print(f"Testing Accuracy: {accuracy_test:.2f}%")

# Plot the decision boundary (testing set)
x_min, x_max = X_test[:, 1].min() - 0.1, X_test[:, 1].max() + 0.1
y_min, y_max = X_test[:, 2].min() - 0.1, X_test[:, 2].max() + 0.1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min, y_max, 100))
grid = np.c_[np.ones((xx.ravel().shape[0], 1)), xx.ravel(), yy.ravel()]
Z = sigmoid(grid @ theta).reshape(xx.shape)

plt.figure(figsize=(6, 5))
plt.contourf(xx, yy, Z, alpha=0.3, levels=[-np.inf, 0.5, np.inf], colors=['lightblue', 'salmon'])
plt.scatter(X_test[y_test == 0, 1], X_test[y_test == 0, 2], color='blue', label='Setosa')
plt.scatter(X_test[y_test == 1, 1], X_test[y_test == 1, 2], color='red', label='Versicolor & Virginica')
plt.xlabel('Standardized Sepal Length')
plt.ylabel('Standardized Sepal Width')
plt.title('Logistic Regression Decision Boundary (Testing Set)')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```

```
Training Accuracy: 99.17%
Testing Accuracy: 100.00%
```


Logistic Regression Decision Boundary (Testing Set)
Setosa
Versicolor & Virginica
Standardized Sepal Width
Standardized Sepal Length
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5