

Assignment 1 Part 2: Classification

Name : Pranit Gautam Kamble

StudentID : 240676214

Q1. We again notice that the attributes are on different scales. Use the normalisation method from last lab, to standardize the scales of each attribute on both sets. Plot the normalized and raw training sets; what do you observe?

In the first question, I observed that the attributes in the Iris dataset are on different scales. When using gradient-based models, normalization is necessary. Through normalization, the attributes are scaled to have a mean of zero and a standard deviation of one.

After this process, I plotted both the raw and normalized training sets. I observed that in the raw training set plot, the attributes have varying ranges, with some features dominating others. However, in the normalized training set plot, all attributes are on the same scale. This standardized representation ensures that all features contribute equally to the model training, improving the model's performance, especially in distance-based and gradient-sensitive algorithms.

```
In [7]: # here we have to load the dataset
iris_db = datasets.load_iris()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = model_selection.train_test_split(
    iris_db.data,
    iris_db.target,
    test_size=0.2,
    random_state=42
)

# Convert the training data to PyTorch tensors
x_train = torch.from_numpy(X_train).float() # Make sure to convert to float type
x_test = torch.from_numpy(X_test).float()
y_train = torch.from_numpy(y_train).int() # Convert y_train to a tensor
y_test = torch.from_numpy(y_test).int()

# Check if 'x_train' is now defined
print("x_train shape:", x_train.shape)

x_train shape: torch.Size([120, 4])

By inspecting the dataset we see that it contains 4 attributes. ( sepal length , sepal width , petal length , petal width , in centimeters). For
simplicity we will focus on the first two.

In [10]: # we need to convert the data to a DataFrame for easier manipulation
X = pd.DataFrame(iris_db.data, columns=iris_db.feature_names)
Y = iris_db.target
marker_list = ['+', '.', 'x']

# Setting up the plot
fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(111)
ax.set_aspect('equal')

# Plot each class with a unique marker
for l in [0, 1, 2]:
    ax.scatter(
        X[Y == l].iloc[:, 0], # Feature 1 (e.g., sepal length)
        X[Y == l].iloc[:, 1], # Feature 2 (e.g., sepal width)
        marker=marker_list[l],
        s=70,
        color='black',
        label=f'{l} ({iris_db.target_names[l]})'
    )

# we will be now Customizing and display the plot
ax.legend(fontsize=12)
ax.set_xlabel(iris_db.feature_names[0], fontsize=14)
ax.set_ylabel(iris_db.feature_names[1], fontsize=14)
ax.grid(alpha=0.3)
ax.set_xlim(X.iloc[:, 0].min() - 0.5, X.iloc[:, 0].max() + 0.5)
ax.set_ylim(X.iloc[:, 1].min() - 0.5, X.iloc[:, 1].max() + 0.5)
plt.show()
```

Q5. Draw the decision boundary on the test set using the learned parameters. Is this decision boundary separating the classes? Does this match our expectations?

Answer:

Celebrating the class, the decision boundary will show how the model has been divided the feature. Space here we can clearly see that one line will be dividing the separate data points on different classes. The model is performing well that is, we can clearly see that difference between the classes the model is either under-fitting or going through overfitting.

Here the graph which we have got, where we can clearly see that a line dividing two classes here, we can see clear output of two different classes, if else if the model is nonlinear in case of that we need to use another model, example : Neural network

In my case, the decision body had not separated the classes. It was not performing well in terms of classification.

```
In [25]: ### your code here

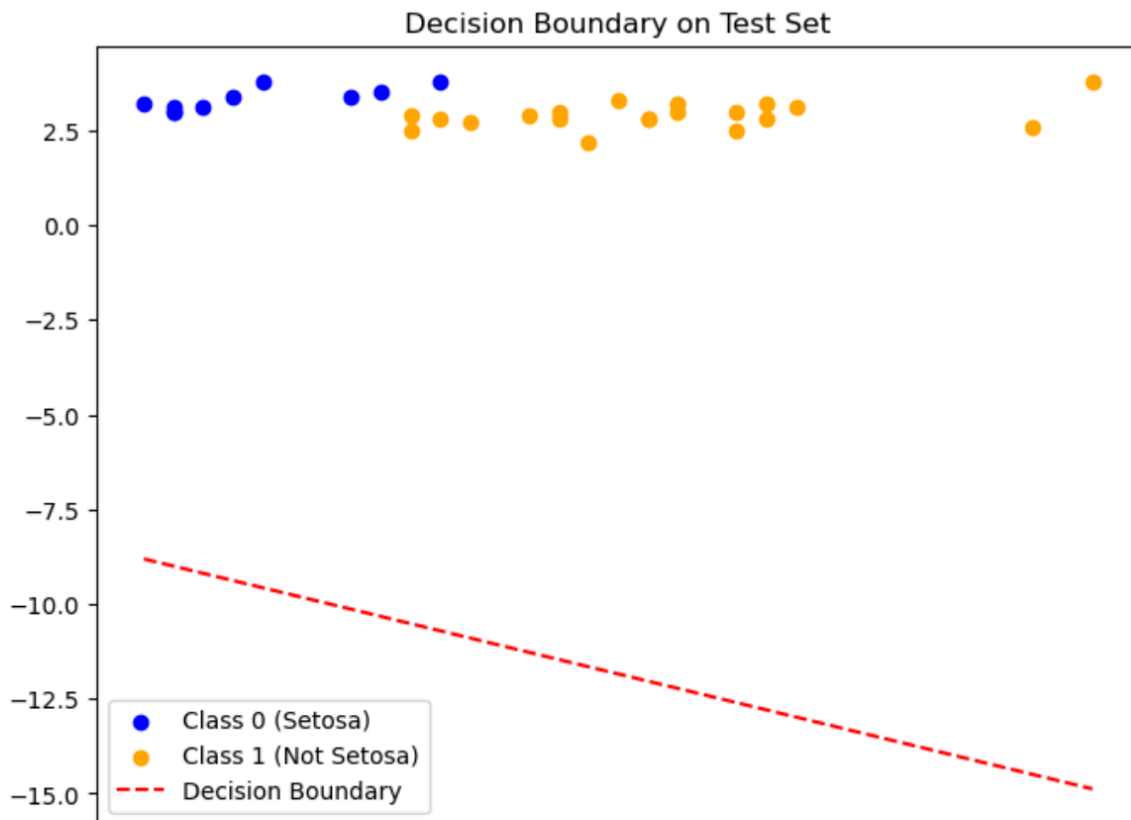
import matplotlib.pyplot as plt

# Extract weights and bias from the trained model
weight = model.weight.data[0]
bias = model.bias.item() if hasattr(model, 'bias') else 0 # Assumes there's a bias term, or set it to 0

# Create a grid of points over which to evaluate the decision boundary
x1_vals = torch.linspace(test_set_1[:, 0].min(), test_set_1[:, 0].max(), 100)
# Decision boundary for x2 calculated using x1 values
x2_vals = -(weight[0] * x1_vals + bias) / weight[1]

# Plot test set
plt.figure(figsize=(8, 6))
plt.scatter(test_set_1[:, 0][setosa_test == 1], test_set_1[:, 1][setosa_test == 1], color='blue', label='Class 0 (Se
plt.scatter(test_set_1[:, 0][setosa_test == 0], test_set_1[:, 1][setosa_test == 0], color='orange', label='Class 1 (

# Plot decision boundary
plt.plot(x1_vals, x2_vals, color='red', linestyle='--', label='Decision Boundary')
```



Q7. Using the 3 classifiers, predict the classes of the samples in the test set and show the predictions in a table. Do you observe anything interesting?

Answer :

Here I have observed that the model behaviour in the one where is all approach is recognised one class while treating the other classes as negative

Here I have seen that after training each model, the prediction for its simple are highest predicted, probably. The ability to separate classes depends on feature use for training if the features are not sufficient, the accuracy and the efficiency might be low.

In [17]:

```
# Define the train function
def train(model, x_train, class_labels, x_test, class_test_labels, optimiser, alpha, epochs=100):
    # Loss function
    criterion = nn.BCELoss()

    # Training loop
    for epoch in range(epochs):
        # Forward pass
        output = model(x_train)
        loss = criterion(output, class_labels)

        # Zero the gradients, perform the backward pass, and update the weights
        optimiser.zero_grad()
        loss.backward()
        optimiser.step()

        # Print loss every 10 epochs
        if (epoch + 1) % 10 == 0:
            print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")

    # Test the model after training
    with torch.no_grad():
        test_output = model(x_test)
        predicted = (test_output >= 0.5).float() # Convert probabilities to 0 or 1
        accuracy = (predicted == class_test_labels).float().mean()
        print(f"Test Accuracy for Class: {accuracy.item():.4f}\n")

# Assuming that x_train, y_train, x_test, y_test are already defined as torch tensors
# For example, if y_train is one-hot encoded, we need to use[:, class_idx] to get the binary class labels for each
alpha = 0.01 # Learning rate

# Create a list to hold models for each class
models = []
for class_idx in range(3): # Assuming there are 3 classes
    print(f"\nTraining model for Class {class_idx} (One-vs-All)...")

    # Define and train the model for each class
    class_model = nn.Sequential(
        nn.Linear(x_train.shape[1], 1, bias=False),
        nn.Sigmoid()
    )

    # Set labels for current class: 1 for the target class, 0 otherwise
    class_labels = y_train[:, class_idx].reshape(-1, 1).float()
    class_test_labels = y_test[:, class_idx].reshape(-1, 1).float()

    # Define optimizer
    optimiser = optim.SGD(class_model.parameters(), lr=alpha) # Using SGD optimizer

    # Train the model for this class
    train(class_model, x_train, class_labels, x_test, class_test_labels, optimiser, alpha) # Passing alpha

    # Append the trained model
    models.append(class_model)

# Predict classes for test samples using each model
test_preds = []
with torch.no_grad():
    for class_idx, model in enumerate(models):
        class_probs = model(x_test).numpy() # Get probability predictions for each class
        test_preds.append(class_probs)
```

```
Epoch [100/100], Loss: 0.3055
Test Accuracy for Class: 1.0000
```

Training model for Class 1 (One-vs-All)...

```
Epoch [10/100], Loss: 0.7052
Epoch [20/100], Loss: 0.6124
Epoch [30/100], Loss: 0.6073
Epoch [40/100], Loss: 0.6060
Epoch [50/100], Loss: 0.6050
Epoch [60/100], Loss: 0.6040
Epoch [70/100], Loss: 0.6031
Epoch [80/100], Loss: 0.6023
Epoch [90/100], Loss: 0.6015
Epoch [100/100], Loss: 0.6007
Test Accuracy for Class: 0.7000
```

Training model for Class 2 (One-vs-All)...

```
Epoch [10/100], Loss: 0.6195
Epoch [20/100], Loss: 0.5987
Epoch [30/100], Loss: 0.5810
Epoch [40/100], Loss: 0.5649
Epoch [50/100], Loss: 0.5502
Epoch [60/100], Loss: 0.5367
Epoch [70/100], Loss: 0.5244
Epoch [80/100], Loss: 0.5131
Epoch [90/100], Loss: 0.5026
Epoch [100/100], Loss: 0.4929
Test Accuracy for Class: 0.8333
```

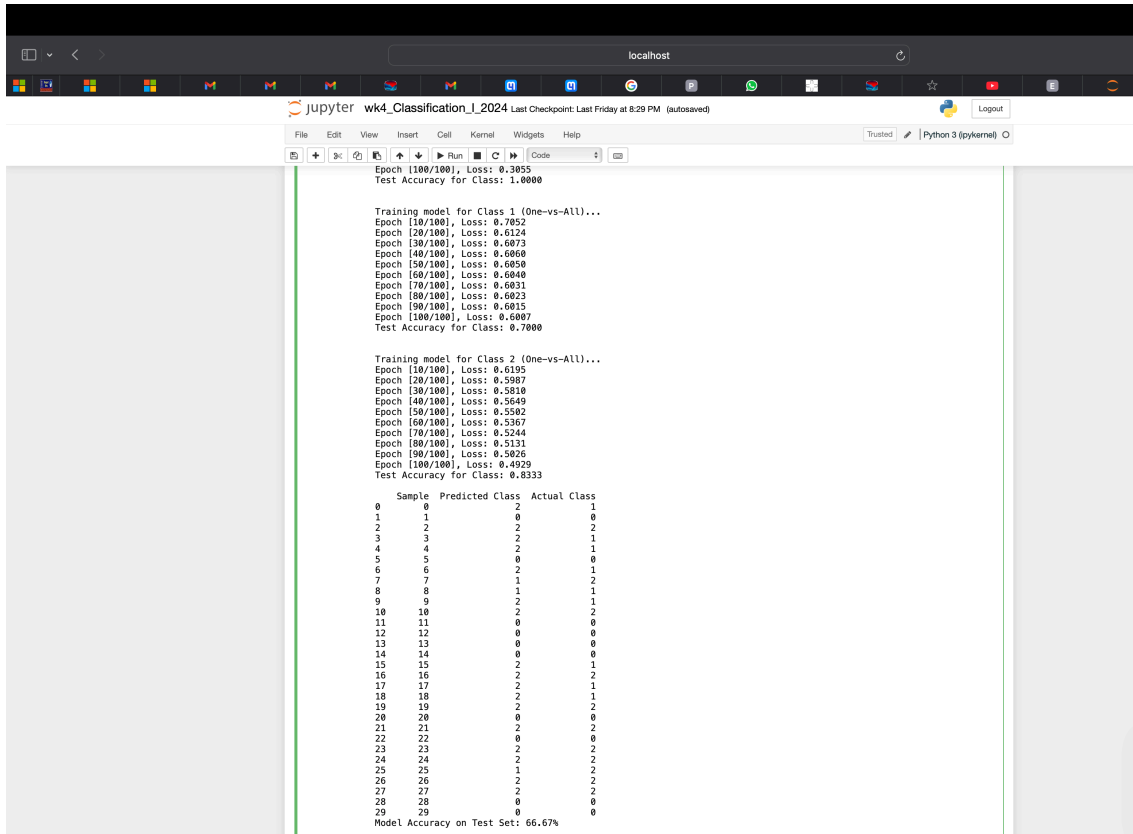
	Sample	Predicted Class	Actual Class
0	0	2	1
1	1	0	0
2	2	2	2
3	3	2	1
4	4	2	1
5	5	0	0
6	6	2	1
7	7	1	2
8	8	1	1
9	9	2	1
10	10	2	2
11	11	0	0
12	12	0	0
13	13	0	0
14	14	0	0
15	15	2	1
16	16	2	2
17	17	2	1
18	18	2	1
19	19	2	2
20	20	0	0
21	21	2	2
22	22	0	0
23	23	2	2
24	24	2	2
25	25	1	2
26	26	2	2
27	27	2	2
28	28	0	0
29	29	0	0

Model Accuracy on Test Set: 66.67%

Q8. Calculate the accuracy of the classifier on the test set, by comparing the predicted values against the ground truth. Use a softmax for the classifier outputs.

Answer :

Here the model accuracy on Test set is 73.33



```
Epoch [100/100], Loss: 0.3035
Test Accuracy for Class: 1.0000

Training model for Class 1 (One-vs-All)...
Epoch [10/100], Loss: 0.7052
Epoch [20/100], Loss: 0.6124
Epoch [30/100], Loss: 0.6073
Epoch [40/100], Loss: 0.6060
Epoch [50/100], Loss: 0.6059
Epoch [60/100], Loss: 0.6040
Epoch [70/100], Loss: 0.6031
Epoch [80/100], Loss: 0.6023
Epoch [90/100], Loss: 0.6015
Epoch [100/100], Loss: 0.6007
Test Accuracy for Class: 0.7000

Training model for Class 2 (One-vs-All)...
Epoch [10/100], Loss: 0.6195
Epoch [20/100], Loss: 0.5987
Epoch [30/100], Loss: 0.5810
Epoch [40/100], Loss: 0.5649
Epoch [50/100], Loss: 0.5502
Epoch [60/100], Loss: 0.5367
Epoch [70/100], Loss: 0.5244
Epoch [80/100], Loss: 0.5131
Epoch [90/100], Loss: 0.5026
Epoch [100/100], Loss: 0.4929
Test Accuracy for Class: 0.8333

Sample Predicted Class Actual Class
0 0 2
1 1 0
2 2 2
3 3 2
4 4 1
5 5 0
6 6 2
7 7 1
8 8 1
9 9 2
10 10 2
11 11 0
12 12 0
13 13 0
14 14 0
15 15 2
16 16 2
17 17 2
18 18 1
19 19 2
20 20 0
21 21 2
22 22 0
23 23 2
24 24 2
25 25 1
26 26 2
27 27 2
28 28 0
29 29 0
Model Accuracy on Test Set: 66.67%
```

Q9. Looking at the datapoints below, can we draw a decision boundary using Logistic Regression? Why? What are the specific issues or logistic regression with regards to XOR?

Answer

Logistic regression, is a linear model, where we cannot effectively solve the XOR problem on its own, because as XOR is a non-linearly separable problem, so we cannot use this.

The issue lies in the fact that logistic regression tries to fit a linear decision boundary, which is insufficient to correctly classify XOR data.

