training_logistic_regression

October 21, 2024

1 Implementing a Logistic Regression Classifier

1.1 1) Installing Libraries

[1]:
```
# !pip install numpy pandas matplotlib --yes
```

[2]:
```
#!pip install torch torchvision torchaudio
```

[3]:
```
#!pip install watermark
```

[4]:
```
#%load_ext watermark
#%watermark -v -p numpy,pandas,matplotlib,torch
```

1.2 2) Loading the Dataset

[5]:
```
import pandas as pd

df = pd.read_csv("perceptron_toydata-truncated.txt", sep="\t")
df
```

[5]:

	x1	x2	label
0	0.77	-1.14	0
1	-0.33	1.44	0
2	0.91	-3.07	0
3	-0.37	-1.91	0
4	-0.63	-1.53	0
5	0.39	-1.99	0
6	-0.49	-2.74	0
7	-0.68	-1.52	0
8	-0.10	-3.43	0
9	-0.05	-1.95	0
10	3.88	0.65	1
11	0.73	2.97	1
12	0.83	3.94	1
13	1.59	1.25	1
14	1.14	3.91	1
15	1.73	2.80	1

```
16  1.31  1.85       1
17  1.56  3.85       1
18  1.23  2.54       1
19  1.33  2.03       1
```

[6]:
```
X_train = df[["x1", "x2"]].values
y_train = df["label"].values
```

[7]:
```
X_train
```

[7]:
```
array([[ 0.77, -1.14],
       [-0.33,  1.44],
       [ 0.91, -3.07],
       [-0.37, -1.91],
       [-0.63, -1.53],
       [ 0.39, -1.99],
       [-0.49, -2.74],
       [-0.68, -1.52],
       [-0.1 , -3.43],
       [-0.05, -1.95],
       [ 3.88,  0.65],
       [ 0.73,  2.97],
       [ 0.83,  3.94],
       [ 1.59,  1.25],
       [ 1.14,  3.91],
       [ 1.73,  2.8 ],
       [ 1.31,  1.85],
       [ 1.56,  3.85],
       [ 1.23,  2.54],
       [ 1.33,  2.03]])
```

[8]:
```
X_train.shape
```

[8]:
```
(20, 2)
```

[9]:
```
y_train
```

[9]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

[10]:
```
y_train.shape
```

[10]:
```
(20,)
```

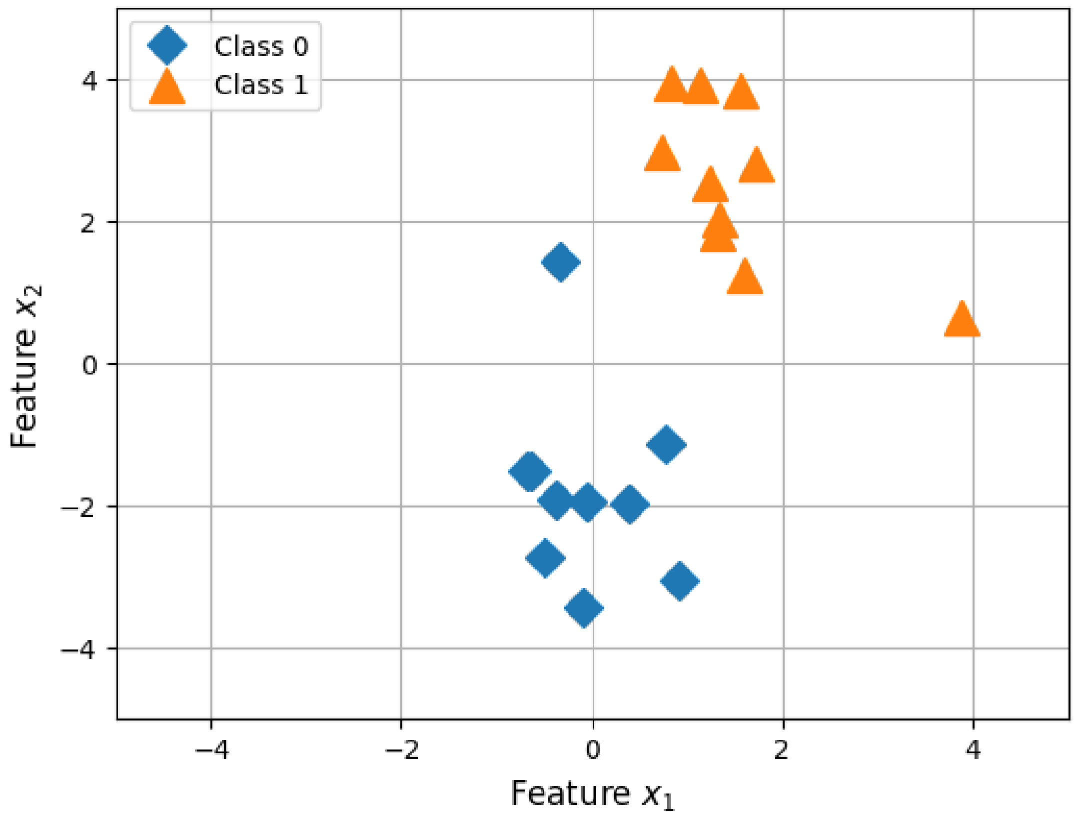
[11]:
```
import numpy as np

np.bincount(y_train)
```

[11]: array([10, 10])

1.3 3) Visualizing the dataset

[12]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

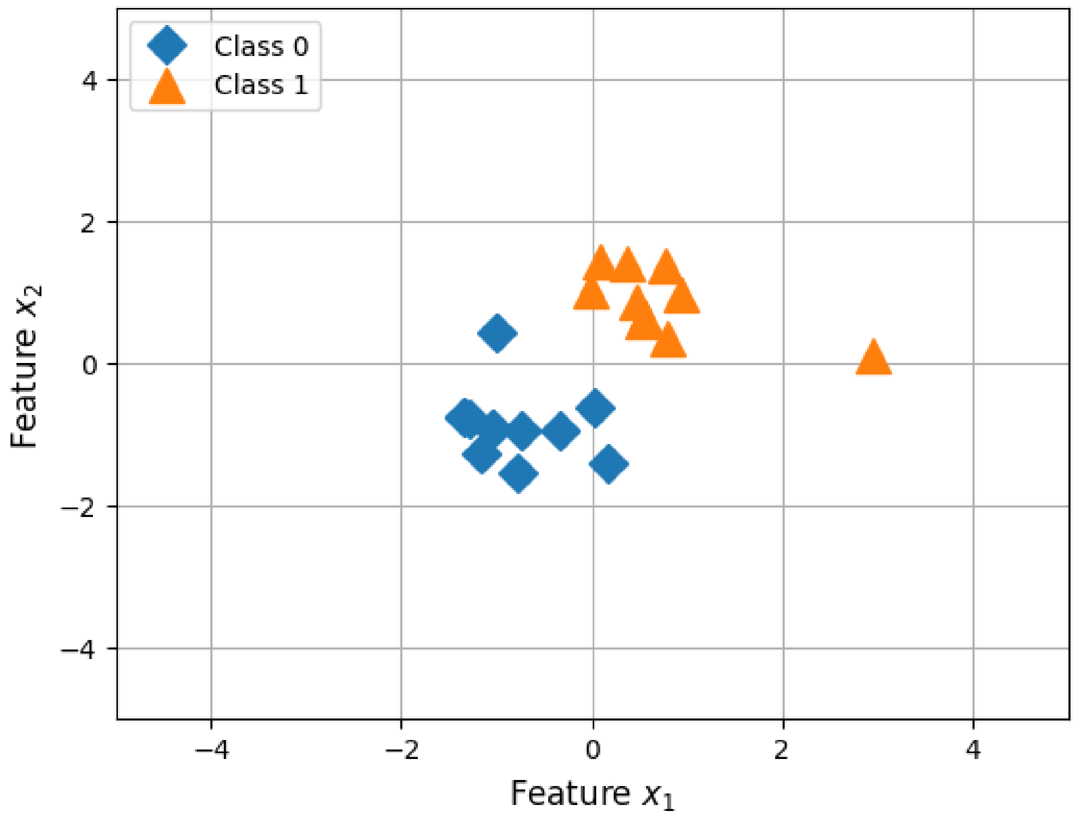
[13]:
```
plt.plot(
    X_train[y_train == 0, 0],
    X_train[y_train == 0, 1],
    marker="D",
    markersize=10,
    linestyle="",
    label="Class 0",
)

plt.plot(
    X_train[y_train == 1, 0],
    X_train[y_train == 1, 1],
    marker="^",
    markersize=13,
    linestyle="",
    label="Class 1",
)

plt.legend(loc=2)

plt.xlim([-5, 5])
plt.ylim([-5, 5])

plt.xlabel("Feature $x_1$", fontsize=12)
plt.ylabel("Feature $x_2$", fontsize=12)

plt.grid()
plt.show()
```



[14]:
```
X_train = (X_train - X_train.mean(axis=0)) / X_train.std(axis=0)
```

[15]:
```
plt.plot(
    X_train[y_train == 0, 0],
    X_train[y_train == 0, 1],
    marker="D",
    markersize=10,
    linestyle="",
    label="Class 0",
)

plt.plot(
    X_train[y_train == 1, 0],
    X_train[y_train == 1, 1],
    marker="^",
    markersize=13,
    linestyle="",
    label="Class 1",
)

plt.legend(loc=2)
```

```
plt.xlim([-5, 5])
plt.ylim([-5, 5])

plt.xlabel("Feature $x_1$", fontsize=12)
plt.ylabel("Feature $x_2$", fontsize=12)

plt.grid()
plt.show()
```



1.4 4) Implementing the model

[16]:
```
import torch

class LogisticRegression(torch.nn.Module):

    def __init__(self, num_features):
        super().__init__()
        self.linear = torch.nn.Linear(num_features, 1)
```

```
    def forward(self, x):
        logits = self.linear(x)
        probas = torch.sigmoid(logits)
        return probas
```

[17]:
```
torch.manual_seed(1)

model = LogisticRegression(num_features=2)
```

[18]:
```
x = torch.tensor([1.1, 2.1])

with torch.no_grad():
    proba = model(x)

print(proba)
```

```
tensor([0.4033])
```

1.5 5) Defining a DataLoader

[19]:
```
from torch.utils.data import Dataset, DataLoader


class MyDataset(Dataset):
    def __init__(self, X, y):

        self.features = torch.tensor(X, dtype=torch.float32)
        self.labels = torch.tensor(y, dtype=torch.float32)

    def __getitem__(self, index):
        x = self.features[index]
        y = self.labels[index]
        return x, y

    def __len__(self):
        return self.labels.shape[0]


train_ds = MyDataset(X_train, y_train)

train_loader = DataLoader(
    dataset=train_ds,
    batch_size=10,
    shuffle=True,
)
```

[20]:
```
X_train.shape
```

[20]: (20, 2)

1.6 6) The training loop

[21]:
```python
import torch.nn.functional as F


torch.manual_seed(1)
model = LogisticRegression(num_features=2)
optimizer = torch.optim.SGD(model.parameters(), lr=0.05)

num_epochs = 20

for epoch in range(num_epochs):

    model = model.train()
    for batch_idx, (features, class_labels) in enumerate(train_loader):

        probas = model(features)

        loss = F.binary_cross_entropy(probas, class_labels.view(probas.shape))

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        ### LOGGING
        print(f'Epoch: {epoch+1:03d}/{num_epochs:03d}'
               f' | Batch {batch_idx:03d}/{len(train_loader):03d}'
               f' | Loss: {loss:.2f}')
```

```
Epoch: 001/020 | Batch 000/002 | Loss: 0.67
Epoch: 001/020 | Batch 001/002 | Loss: 0.73
Epoch: 002/020 | Batch 000/002 | Loss: 0.67
Epoch: 002/020 | Batch 001/002 | Loss: 0.67
Epoch: 003/020 | Batch 000/002 | Loss: 0.60
Epoch: 003/020 | Batch 001/002 | Loss: 0.68
Epoch: 004/020 | Batch 000/002 | Loss: 0.69
Epoch: 004/020 | Batch 001/002 | Loss: 0.54
Epoch: 005/020 | Batch 000/002 | Loss: 0.61
Epoch: 005/020 | Batch 001/002 | Loss: 0.57
Epoch: 006/020 | Batch 000/002 | Loss: 0.59
Epoch: 006/020 | Batch 001/002 | Loss: 0.54
Epoch: 007/020 | Batch 000/002 | Loss: 0.51
Epoch: 007/020 | Batch 001/002 | Loss: 0.58
Epoch: 008/020 | Batch 000/002 | Loss: 0.51
```

```
Epoch: 008/020 | Batch 001/002 | Loss: 0.54
Epoch: 009/020 | Batch 000/002 | Loss: 0.51
Epoch: 009/020 | Batch 001/002 | Loss: 0.49
Epoch: 010/020 | Batch 000/002 | Loss: 0.53
Epoch: 010/020 | Batch 001/002 | Loss: 0.44
Epoch: 011/020 | Batch 000/002 | Loss: 0.42
Epoch: 011/020 | Batch 001/002 | Loss: 0.52
Epoch: 012/020 | Batch 000/002 | Loss: 0.46
Epoch: 012/020 | Batch 001/002 | Loss: 0.44
Epoch: 013/020 | Batch 000/002 | Loss: 0.46
Epoch: 013/020 | Batch 001/002 | Loss: 0.42
Epoch: 014/020 | Batch 000/002 | Loss: 0.41
Epoch: 014/020 | Batch 001/002 | Loss: 0.43
Epoch: 015/020 | Batch 000/002 | Loss: 0.43
Epoch: 015/020 | Batch 001/002 | Loss: 0.39
Epoch: 016/020 | Batch 000/002 | Loss: 0.41
Epoch: 016/020 | Batch 001/002 | Loss: 0.39
Epoch: 017/020 | Batch 000/002 | Loss: 0.38
Epoch: 017/020 | Batch 001/002 | Loss: 0.39
Epoch: 018/020 | Batch 000/002 | Loss: 0.44
Epoch: 018/020 | Batch 001/002 | Loss: 0.31
Epoch: 019/020 | Batch 000/002 | Loss: 0.34
Epoch: 019/020 | Batch 001/002 | Loss: 0.39
Epoch: 020/020 | Batch 000/002 | Loss: 0.33
Epoch: 020/020 | Batch 001/002 | Loss: 0.38
```

1.7 7) Evaluating the results

[22]:
```
probas
```

[22]:
```
tensor([[0.6687],
        [0.6810],
        [0.2162],
        [0.7398],
        [0.2126],
        [0.5744],
        [0.3209],
        [0.6503],
        [0.4276],
        [0.2598]], grad_fn=<SigmoidBackward0>)
```

[23]:
```
pred = torch.where(probas > 0.5, 1, 0)
pred
```

[23]:
```
tensor([[1],
        [1],
        [0],
```

```
         [1],
         [0],
         [1],
         [0],
         [1],
         [0],
         [0]])
```

[24]:
```
class_labels.view(pred.shape).to(pred.dtype)
```

[24]:
```
tensor([[1],
         [1],
         [0],
         [1],
         [0],
         [1],
         [0],
         [1],
         [0],
         [0]])
```

[25]:
```
def compute_accuracy(model, dataloader):

    model = model.eval()

    correct = 0.0
    total_examples = 0

    for idx, (features, class_labels) in enumerate(dataloader):

        with torch.no_grad():
            probas = model(features)

        pred = torch.where(probas > 0.5, 1, 0)
        lab = class_labels.view(pred.shape).to(pred.dtype)

        compare = lab == pred
        correct += torch.sum(compare)
        total_examples += len(compare)

    return correct / total_examples
```

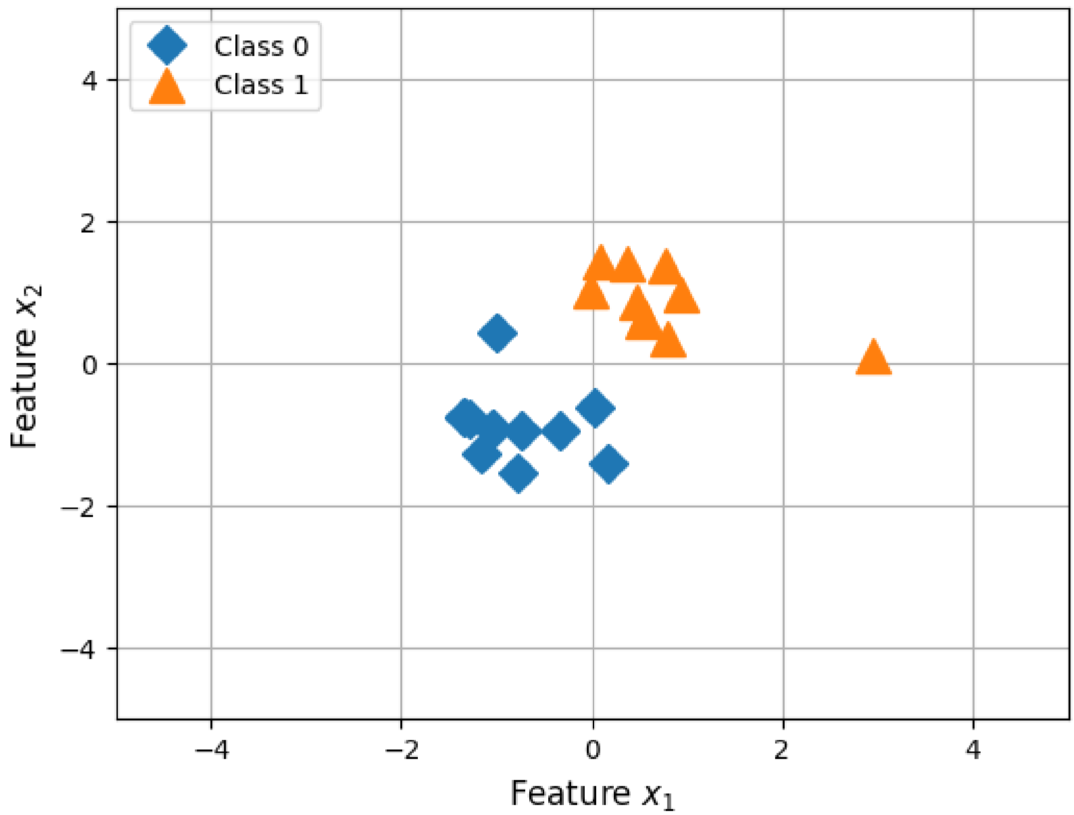
[26]:
```
train_acc = compute_accuracy(model, train_loader)
```

[27]:
```
print(f"Accuracy: {train_acc*100}%")
```

```
Accuracy: 100.0%
```

1.8 8) Optional: visualizing the decision boundary

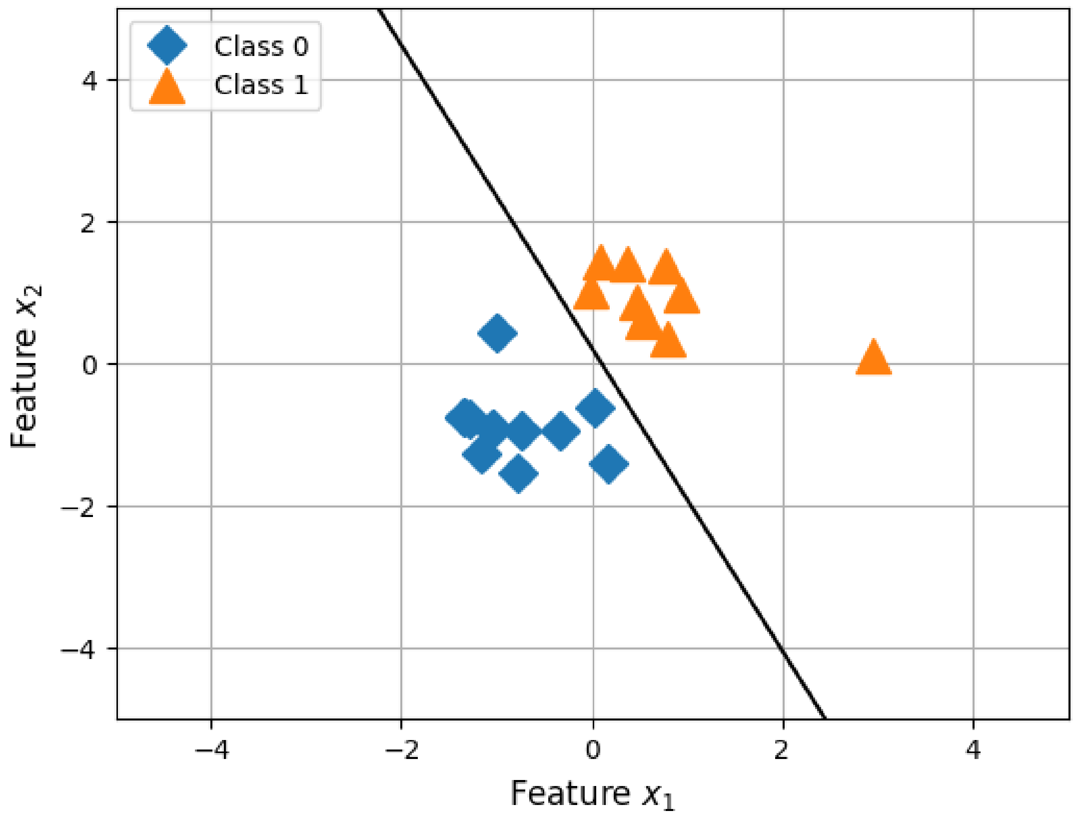
[28]:
```
plt.plot(
    X_train[y_train == 0, 0],
    X_train[y_train == 0, 1],
    marker="D",
    markersize=10,
    linestyle="",
    label="Class 0",
)

plt.plot(
    X_train[y_train == 1, 0],
    X_train[y_train == 1, 1],
    marker="^",
    markersize=13,
    linestyle="",
    label="Class 1",
)

plt.legend(loc=2)

plt.xlim([-5, 5])
plt.ylim([-5, 5])

plt.xlabel("Feature $x_1$", fontsize=12)
plt.ylabel("Feature $x_2$", fontsize=12)

plt.grid()
plt.show()
```



[29]:
```
def plot_boundary(model):

    w1 = model.linear.weight[0][0].detach()
    w2 = model.linear.weight[0][1].detach()
    b = model.linear.bias[0].detach()

    x1_min = -20
    x2_min = (-(w1 * x1_min) - b) / w2

    x1_max = 20
    x2_max = (-(w1 * x1_max) - b) / w2

    return x1_min, x1_max, x2_min, x2_max
```

[30]:
```
x1_min, x1_max, x2_min, x2_max = plot_boundary(model)


plt.plot(
    X_train[y_train == 0, 0],
    X_train[y_train == 0, 1],
    marker="D",
```

```
    markersize=10,
    linestyle="",
    label="Class 0",
)

plt.plot(
    X_train[y_train == 1, 0],
    X_train[y_train == 1, 1],
    marker="^",
    markersize=13,
    linestyle="",
    label="Class 1",
)

plt.plot([x1_min, x1_max], [x2_min, x2_max], color="k")

plt.legend(loc=2)

plt.xlim([-5, 5])
plt.ylim([-5, 5])

plt.xlabel("Feature $x_1$", fontsize=12)
plt.ylabel("Feature $x_2$", fontsize=12)

plt.grid()
plt.show()
```



Class 0
Class 1
4
2
0
−2
−4
Feature x_2
−4
−2
0
2
4
Feature x_1