lab-5-NFSU-classwork-questions

March 27, 2023

0.1 Sigmoid Function and Linear Classification

0.1.1 Classroom Task

The answers to these questions were discussed in today's lab. Hence you just have to repeat what ws done in the class. The difficulty level of this assignment is *Very easy*

[39]:
```
import numpy as np
import torch
import torch.nn.functional as F
import pandas as pd
import matplotlib.pyplot as plt
```

0.2 Prepare the data (10 Marks)

[28]:
```
# load the iris dataset: (10 marks)
df = #write your code here
df.columns = ['x1', 'x2', 'x3', 'x4', 'y']

# drop all rows with y = iris-versicolor
df = # complete the code

d = {'Iris-virginica': 1,
     'Iris-setosa': 0,}

df['y'] = df['y'].map(d)

# prepare data for ML by assigning features and target
X = torch.tensor(df[['x2', 'x4']].values, dtype=torch.float)
y = torch.tensor(df['y'].values, dtype=torch.int)

# shuffle the data to make train and test split
torch.manual_seed(123)
shuffle_idx = torch.randperm(y.size(0), dtype=torch.long)
percent80 = int(shuffle_idx.size(0)*0.8)

X_train, X_test = X[shuffle_idx[:percent80]], X[shuffle_idx[percent80:]]
y_train, y_test = y[shuffle_idx[:percent80]], y[shuffle_idx[percent80:]]
```

```
# Standardise (mean zero, unit variance)
mu, sigma = X_train.mean(dim=0), X_train.std(dim=0)
X_train = (X_train - mu) / sigma
X_test = (X_test - mu) / sigma
```

0.3 Theory Question (No code is required for such questions): (5 marks)

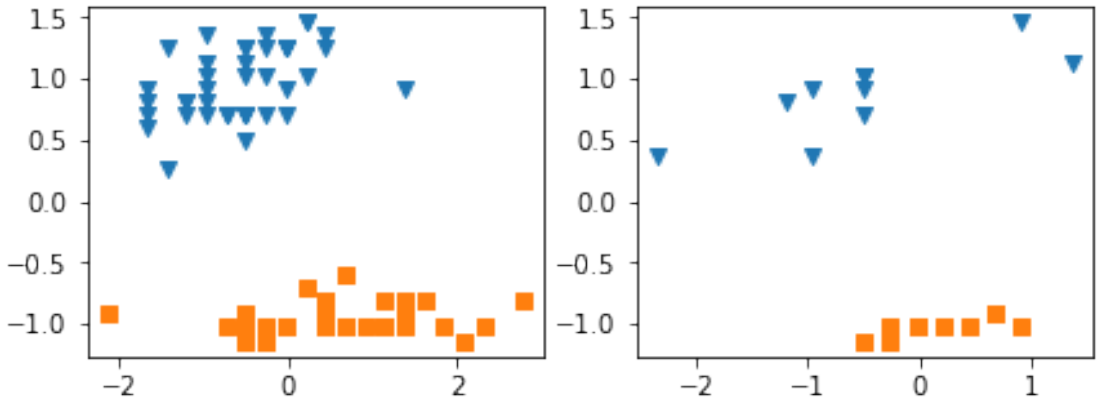
Q: What does the line `df['y'] = df['y'].map(d)` do in the above code. Write your answer below in one line. A: Replace this and write your answer here

0.4 Coding Question (10 marks)

[ ]:
```
# check if you data is standardised. (This was taught in Lab 2) [10 marks]
# Write your code here
```

0.4.1 Visualise the data

[29]:
```
fig, ax = plt.subplots(1, 2, figsize=(7, 2.5))
ax[0].scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], marker='v')
ax[0].scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], marker='s')
ax[1].scatter(X_test[y_test == 1, 0], X_test[y_test == 1, 1], marker='v')
ax[1].scatter(X_test[y_test == 0, 0], X_test[y_test == 0, 1], marker='s')
plt.show()
```



0.5 Theory Question (No code is required for such questions): (5 marks)

Q: What does the two plot in the above cell represent? (Answer in no more than one line) A: Replace this and write your answer here

0.6 Logistic Regression - The Signmoid Function

$$z = \mathbf{w}^T\mathbf{x} + b$$

$$a = \sigma(z)$$

0.7 Coding Question (10 marks)

[34]:
```
# write the function to implement sigmoid function (5 Marks)
```

[ ]:
```
# Use the sigmoid function to convert the x2 features between 0 and 1, show the␣
↪results using a plot (5 Marks)
```

The sigmoid function quashes everything between 0 and 1

0.8 Theory Question (No code is required for such questions): (5 marks)

Q:What will the sigmoid function return for an input value of 0.5? A: Replace this and write your answer here

0.9 Logistic Regression - Model Training With Pytorch

[37]:
```
device  = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
class LogisticRegression2(torch.nn.Module):
    def __init__(self, num_features):
        super(LogisticRegression2, self).__init__()
        self.linear = torch.nn.Linear(num_features, 1, dtype=torch.float32,␣
↪device = device)

        # initialize weights to zeros here,
        # since we used zero weights in the
        # manual approach
        self.linear.weight.detach().zero_()
        self.linear.bias.detach().zero_()

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

model = LogisticRegression2(2).to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
```

0.10 Theory Question (No code is required for such questions): (5 marks)

Q:What is the full name of the optimiser used for Logistic Regression? A: Replace this and write your answer here

Q:In the line `model = LogisticRegression2(2).to(device)` what does 2 represent? A: Replace this and write your answer here

[42]:
```
def comp_accuracy(label_var, pred_probas):
    pred_labels = torch.where((pred_probas > 0.5), 1, 0).view(-1)
    acc = torch.sum(pred_labels == label_var.view(-1)).float() / label_var.
 ↪size(0)
    return acc

num_epochs = 30

X_train_tensor = torch.tensor(X_train, dtype=torch.float32, device=device)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32, device=device).
 ↪view(-1, 1)


for epoch in range(num_epochs):

    #### Compute outputs ####
    out = model(X_train_tensor)

    #### Compute gradients ####
    loss = F.binary_cross_entropy(out, y_train_tensor, reduction='sum')
    optimizer.zero_grad()
    loss.backward()

    #### Update weights ####
    optimizer.step()

    #### Logging ####
    pred_probas = model(X_train_tensor)
    acc = comp_accuracy(y_train_tensor, pred_probas)
    print('Epoch: %03d' % (epoch + 1), end="")
    print(' | Train ACC: %.3f' % acc, end="")
    print(' | Cost: %.3f' % F.binary_cross_entropy(pred_probas, y_train_tensor))


print('\nModel parameters:')
print('  Weights: %s' % model.linear.weight)
print('  Bias: %s' % model.linear.bias)
```

```
Epoch: 001 | Train ACC: 1.000 | Cost: 0.005
Epoch: 002 | Train ACC: 1.000 | Cost: 0.005
Epoch: 003 | Train ACC: 1.000 | Cost: 0.004
Epoch: 004 | Train ACC: 1.000 | Cost: 0.004
Epoch: 005 | Train ACC: 1.000 | Cost: 0.004
Epoch: 006 | Train ACC: 1.000 | Cost: 0.004
Epoch: 007 | Train ACC: 1.000 | Cost: 0.004
Epoch: 008 | Train ACC: 1.000 | Cost: 0.004
Epoch: 009 | Train ACC: 1.000 | Cost: 0.004
```

```
Epoch: 010 | Train ACC: 1.000 | Cost: 0.004
Epoch: 011 | Train ACC: 1.000 | Cost: 0.004
Epoch: 012 | Train ACC: 1.000 | Cost: 0.004
Epoch: 013 | Train ACC: 1.000 | Cost: 0.004
Epoch: 014 | Train ACC: 1.000 | Cost: 0.004
Epoch: 015 | Train ACC: 1.000 | Cost: 0.004
Epoch: 016 | Train ACC: 1.000 | Cost: 0.003
Epoch: 017 | Train ACC: 1.000 | Cost: 0.003
Epoch: 018 | Train ACC: 1.000 | Cost: 0.003
Epoch: 019 | Train ACC: 1.000 | Cost: 0.003
Epoch: 020 | Train ACC: 1.000 | Cost: 0.003
Epoch: 021 | Train ACC: 1.000 | Cost: 0.003
Epoch: 022 | Train ACC: 1.000 | Cost: 0.003
Epoch: 023 | Train ACC: 1.000 | Cost: 0.003
Epoch: 024 | Train ACC: 1.000 | Cost: 0.003
Epoch: 025 | Train ACC: 1.000 | Cost: 0.003
Epoch: 026 | Train ACC: 1.000 | Cost: 0.003
Epoch: 027 | Train ACC: 1.000 | Cost: 0.003
Epoch: 028 | Train ACC: 1.000 | Cost: 0.003
Epoch: 029 | Train ACC: 1.000 | Cost: 0.003
Epoch: 030 | Train ACC: 1.000 | Cost: 0.003

Model parameters:
  Weights: Parameter containing:
tensor([[-1.2253,  6.4618]], requires_grad=True)
  Bias: Parameter containing:
tensor([0.0646], requires_grad=True)

/var/folders/_3/x_hy8vf90v93s9rdb_r5pj140000gn/T/ipykernel_71013/1144892227.py:8
: UserWarning: To copy construct from a tensor, it is recommended to use
sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  X_train_tensor = torch.tensor(X_train, dtype=torch.float32, device=device)
/var/folders/_3/x_hy8vf90v93s9rdb_r5pj140000gn/T/ipykernel_71013/1144892227.py:9
: UserWarning: To copy construct from a tensor, it is recommended to use
sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  y_train_tensor = torch.tensor(y_train, dtype=torch.float32,
device=device).view(-1, 1)
```

[44]:
```
X_test_tensor = torch.tensor(X_test, dtype=torch.float32, device=device)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32, device=device)

pred_probas = model(X_test_tensor)
test_acc = comp_accuracy(y_test_tensor, pred_probas)
```

```
print('Test set accuracy: %.2f%%' % (test_acc*100))
```

```
Test set accuracy: 100.00%
```

```
/var/folders/_3/x_hy8vf90v93s9rdb_r5pj140000gn/T/ipykernel_71013/1323677059.py:1
: UserWarning: To copy construct from a tensor, it is recommended to use
sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  X_test_tensor = torch.tensor(X_test, dtype=torch.float32, device=device)
/var/folders/_3/x_hy8vf90v93s9rdb_r5pj140000gn/T/ipykernel_71013/1323677059.py:2
: UserWarning: To copy construct from a tensor, it is recommended to use
sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  y_test_tensor = torch.tensor(y_test, dtype=torch.float32, device=device)
```