lab-5-NFSU

March 27, 2023

0.1 Sigmoid Function and Linear Classification

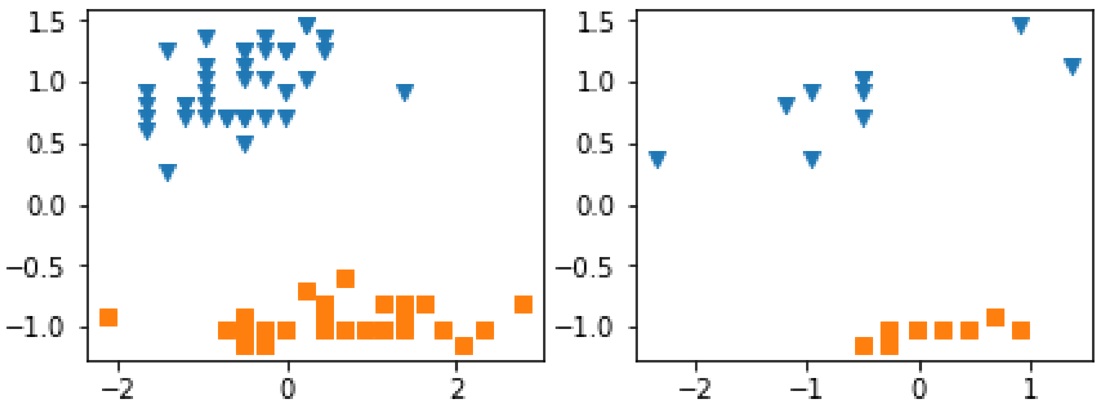
[1]:
```
import numpy as np
import torch
import torch.nn.functional as F
import pandas as pd
import matplotlib.pyplot as plt
```

0.1.1 Prepare the data

[49]:
```
df = pd.read_csv('./data/iris.data', index_col=None, header=None)
df.columns = ['x1', 'x2', 'x3', 'x4', 'y']

# drop all rows with y = iris-versicolor
df = df[df['y'] != 'Iris-versicolor']

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
```

0.1.2 Visualise the data

[57]:
```
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()
```



0.2 Logistic Regression - The Sigmoid Function

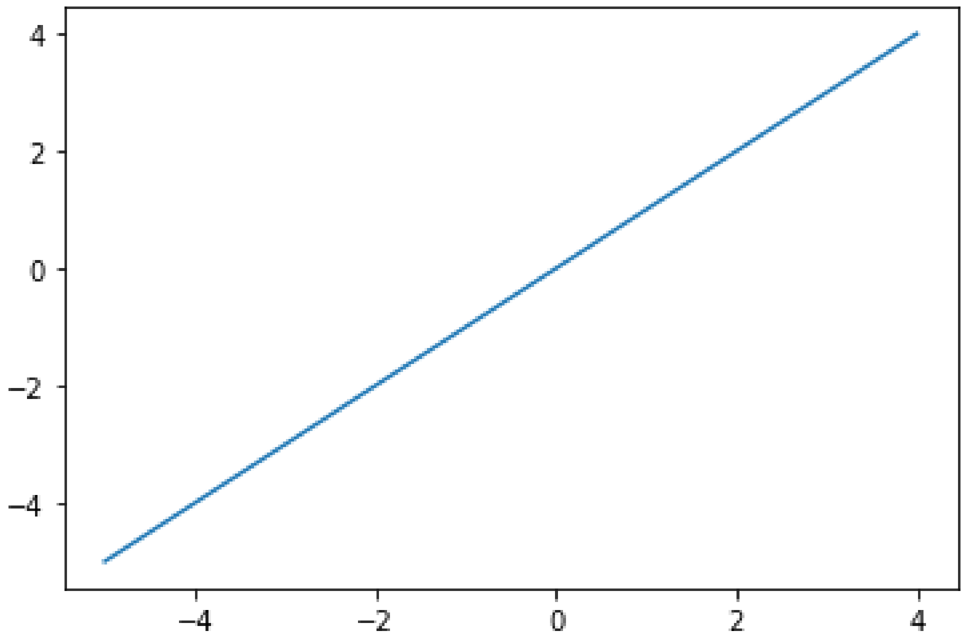
$$z = \mathbf{w}^T\mathbf{x} + b$$

$$a = \sigma(z)$$

[58]:
```
def sigmoid(x):
    return 1/(1+torch.exp(-x))
```

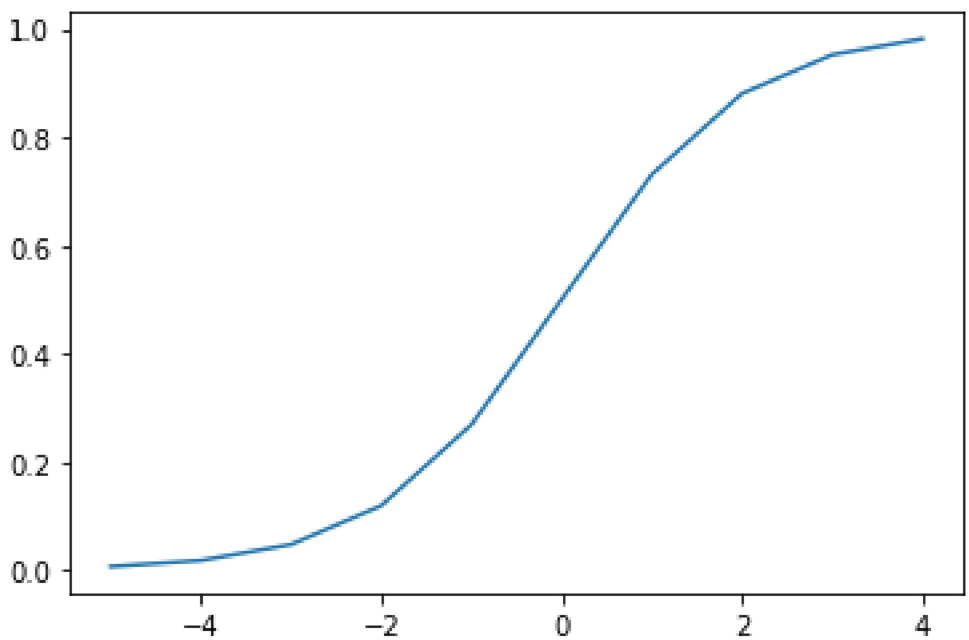
[77]:
```
examl = [torch.tensor(i) for i in range(-5, 5, 1)]
plt.plot(examl,examl)
```

[77]:
```
[<matplotlib.lines.Line2D at 0x7fd485c5af40>]
```



[78]:
```
plt.plot(examl,sigmoid(torch.tensor(examl)))
```

[78]: [<matplotlib.lines.Line2D at 0x7fd485edf190>]

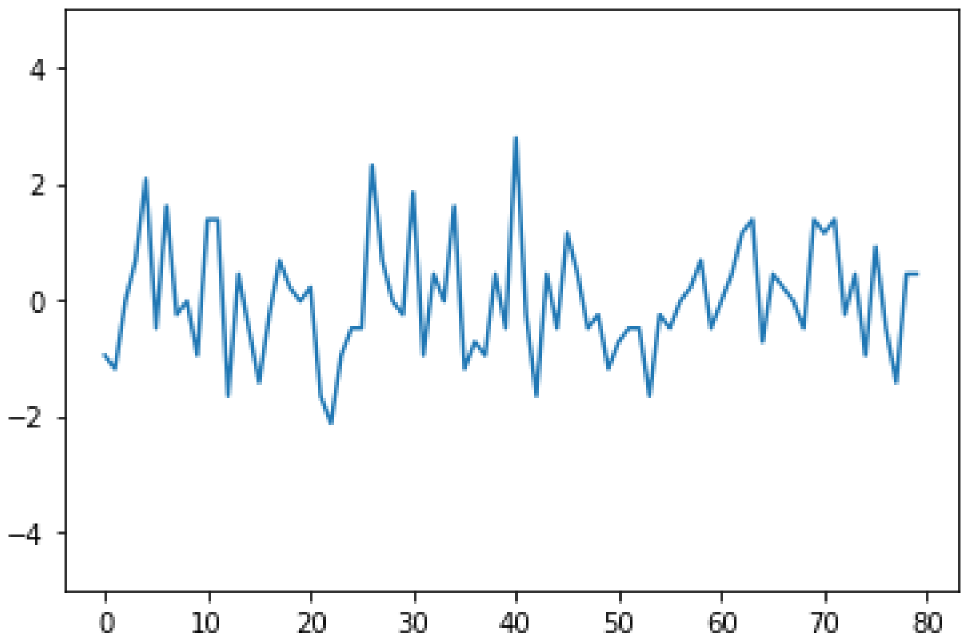


[82]:
```
sum(sigmoid(torch.tensor(examl)))
```

[82]: tensor(4.5067)

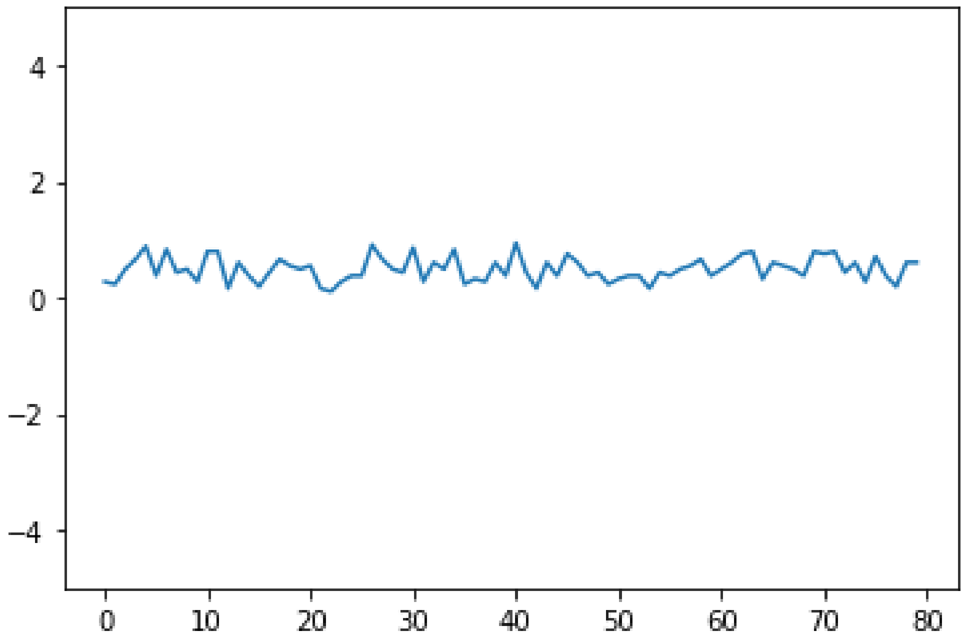
[80]:
```
plt.plot(X_train[:,0])
plt.ylim(-5,5)
```

[80]: (-5.0, 5.0)



[81]:
```
plt.plot(sigmoid(X_train[:,0]))
plt.ylim(-5,5)
```

[81]: (-5.0, 5.0)



The sigmoid function quashes eberything between 0 and 1

0.2.1 Softmax

[87]:
```
m = torch.nn.Softmax()
rnd = torch.rand(10)
print("Random 10 numbers = {}".format(rnd))
print("Sum of random 10 numbers = {}".format(sum(rnd)))
print("Softmax of 10 numbers = {}".format(m(rnd)))
print("Sum of the softmax of random 10 numbers = {}".format(sum(m(rnd))))
```

```
Random 10 numbers = tensor([0.8954, 0.2979, 0.6314, 0.5028, 0.1239, 0.3786,
0.1661, 0.7211, 0.5449,
        0.5490])
Sum of random 10 numbers = 4.811153411865234
Softmax of 10 numbers = tensor([0.1474, 0.0811, 0.1132, 0.0995, 0.0681, 0.0879,
0.0711, 0.1238, 0.1038,
        0.1042])
Sum of the softmax of random 10 numbers = 1.0
```

```
/var/folders/_3/x_hy8vf90v93s9rdb_r5pj140000gn/T/ipykernel_14971/257327191.py:5:
UserWarning: Implicit dimension choice for softmax has been deprecated. Change
the call to include dim=X as an argument.
  print("Softmax of 10 numbers = {}".format(m(rnd)))
/var/folders/_3/x_hy8vf90v93s9rdb_r5pj140000gn/T/ipykernel_14971/257327191.py:6:
UserWarning: Implicit dimension choice for softmax has been deprecated. Change
the call to include dim=X as an argument.
```

```
print("Sum of the softmax of random 10 numbers = {}".format(sum(m(rnd))))
```

0.3 Logistic Regression - Model Training With Pytorch

[88]:
```
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)

    def forward(self, x):
        logits = self.linear(x); # y = Wx+b
        probas = torch.sigmoid(logits) # y^ = sigmoid(y)
        return probas;

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

[89]:
```
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)
```

```python
    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: 0.988 | Cost: 0.038
Epoch: 002 | Train ACC: 0.988 | Cost: 0.028
Epoch: 003 | Train ACC: 1.000 | Cost: 0.022
Epoch: 004 | Train ACC: 1.000 | Cost: 0.018
Epoch: 005 | Train ACC: 1.000 | Cost: 0.015
Epoch: 006 | Train ACC: 1.000 | Cost: 0.014
Epoch: 007 | Train ACC: 1.000 | Cost: 0.012
Epoch: 008 | Train ACC: 1.000 | Cost: 0.011
Epoch: 009 | Train ACC: 1.000 | Cost: 0.011
Epoch: 010 | Train ACC: 1.000 | Cost: 0.010
Epoch: 011 | Train ACC: 1.000 | Cost: 0.009
Epoch: 012 | Train ACC: 1.000 | Cost: 0.009
Epoch: 013 | Train ACC: 1.000 | Cost: 0.008
Epoch: 014 | Train ACC: 1.000 | Cost: 0.008
Epoch: 015 | Train ACC: 1.000 | Cost: 0.008
Epoch: 016 | Train ACC: 1.000 | Cost: 0.007
Epoch: 017 | Train ACC: 1.000 | Cost: 0.007
Epoch: 018 | Train ACC: 1.000 | Cost: 0.007
Epoch: 019 | Train ACC: 1.000 | Cost: 0.007
Epoch: 020 | Train ACC: 1.000 | Cost: 0.006
Epoch: 021 | Train ACC: 1.000 | Cost: 0.006
Epoch: 022 | Train ACC: 1.000 | Cost: 0.006
Epoch: 023 | Train ACC: 1.000 | Cost: 0.006
Epoch: 024 | Train ACC: 1.000 | Cost: 0.006
Epoch: 025 | Train ACC: 1.000 | Cost: 0.006
Epoch: 026 | Train ACC: 1.000 | Cost: 0.005
Epoch: 027 | Train ACC: 1.000 | Cost: 0.005
Epoch: 028 | Train ACC: 1.000 | Cost: 0.005
Epoch: 029 | Train ACC: 1.000 | Cost: 0.005
Epoch: 030 | Train ACC: 1.000 | Cost: 0.005

Model parameters:
  Weights: Parameter containing:
tensor([[-1.2392,  5.7726]], requires_grad=True)
  Bias: Parameter containing:
tensor([-0.0394], requires_grad=True)
```

```
/var/folders/_3/x_hy8vf90v93s9rdb_r5pj140000gn/T/ipykernel_14971/1144892227.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_14971/1144892227.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)
```

[58]:
```
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))
```

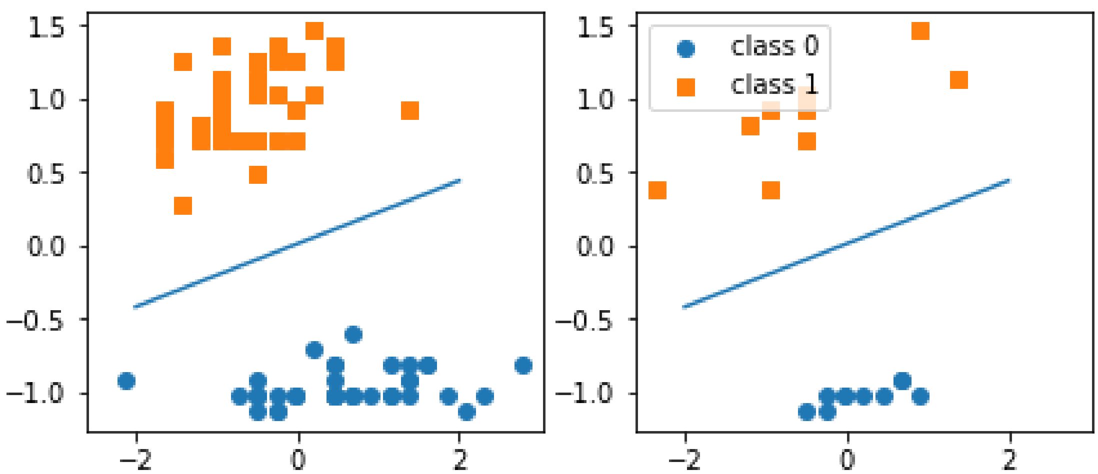
```
Test set accuracy: 100.00%
```

```
/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)
```

0.4 Decision Boundary

[90]:
```
w, b = model.linear.weight.detach().view(-1), model.linear.bias.detach()

x_min = -2
y_min = ( (-(w[0] * x_min) - b[0])
          / w[1] )

x_max = 2
y_max = ( (-(w[0] * x_max) - b[0])
          / w[1] )
```

```python
fig, ax = plt.subplots(1, 2, sharex=True, figsize=(7, 3))
ax[0].plot([x_min, x_max], [y_min, y_max])
ax[1].plot([x_min, x_max], [y_min, y_max])

ax[0].scatter(X_train[y_train==0, 0], X_train[y_train==0, 1], label='class 0', marker='o')
ax[0].scatter(X_train[y_train==1, 0], X_train[y_train==1, 1], label='class 1', marker='s')

ax[1].scatter(X_test[y_test==0, 0], X_test[y_test==0, 1], label='class 0', marker='o')
ax[1].scatter(X_test[y_test==1, 0], X_test[y_test==1, 1], label='class 1', marker='s')

ax[1].legend(loc='upper left')
plt.show()
```



[ ]: