Logistic regression

```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
```

```
x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]
```

```
x_train = torch.FloatTensor(x_data)
y_train = torch.FloatTensor(y_data)
```

Hypothesis

```
from IPython.display import Image
image_url = 'https://bitbucket.org/hyuk125/lg_dic/raw/99785e9d01523e8bb6bf78d1
Image(image_url)
```

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

cost

```
from IPython.display import Image
image_url = 'https://bitbucket.org/hyuk125/lg_dic/raw/99785e9d01523e8bb6bf78d1
Image(image_url)
```

$$cost(W) = -\frac{1}{m}\sum y\log(H(x)) + (1-y)(\log(1-H(x))$$

Pytorch has binary cross entropy function:
torch.nn.functional.binary_cross_entropy(input, target, ...)

```
predict = torch.FloatTensor(np.array([[0.1], [0.2], [0.9]]))
label = torch.FloatTensor(np.array([[0], [0], [1]]))
```

```
F.binary_cross_entropy(predict, label)
```

```
tensor(0.1446)
```

Training

```
class my_logistic(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(2, 1)
        self.sigmoid = nn.Sigmoid() # Just add sigmoid function!

    def forward(self, x):
        return self.sigmoid(self.linear(x))
```

```
model = my_logistic()
```

```
# optimizer 설정
optimizer = optim.SGD(model.parameters(), lr=1)

nb_epochs = 100
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    hypothesis = model(x_train)

    # cost 계산
    cost = F.binary_cross_entropy(hypothesis, y_train)

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 20번마다 로그 출력
    if epoch % 10 == 0:
        prediction = hypothesis >= torch.FloatTensor([0.5])
        correct_prediction = prediction.float() == y_train
        accuracy = correct_prediction.sum().item() / len(correct_prediction)
        print('Epoch {:4d}/{} Cost: {:.6f} '.format(
            epoch, nb_epochs, cost.item(),
        ))
```

```
Epoch    0/100 Cost: 2.302668
Epoch   10/100 Cost: 0.546537
Epoch   20/100 Cost: 0.407635
Epoch   30/100 Cost: 0.344355
Epoch   40/100 Cost: 0.291087
Epoch   50/100 Cost: 0.243137
Epoch   60/100 Cost: 0.200380
Epoch   70/100 Cost: 0.167988
Epoch   80/100 Cost: 0.150016
Epoch   90/100 Cost: 0.139079
Epoch  100/100 Cost: 0.129961
```

결과 출력

```
from sklearn.decomposition import PCA
```

```
x1 = torch.linspace(0, 10, 100).reshape(-1, 1)
x2 = torch.linspace(0, 10, 100).reshape(-1, 1)
```

```
pca = PCA(n_components=1)
x_pca_100 = pca.fit_transform(torch.cat((x1, x2),axis=1))
```

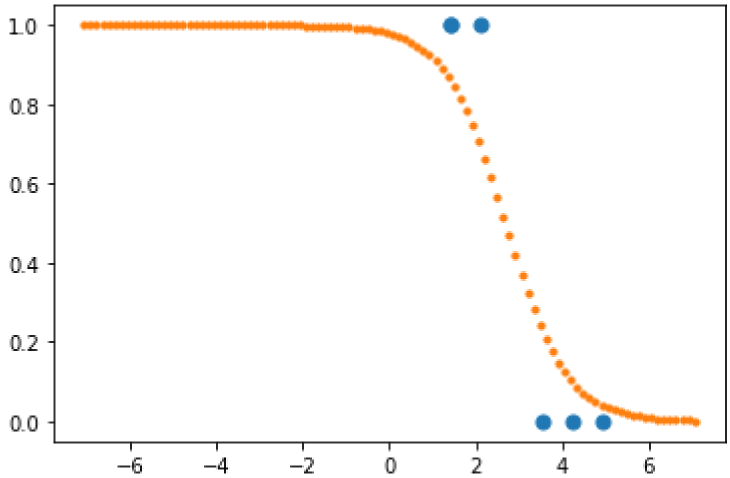
```
x_pca = pca.transform(x_train)
```

```
hypothesis = model(torch.cat((x1, x2),axis=1))
```

```
hx = hypothesis.detach().numpy()
```

```
plt.scatter(x_pca, y_train, s=50)
plt.scatter(x_pca_100, hx, s=10)
```

```
<matplotlib.collections.PathCollection at 0x7f43b2bc0350>
```



Accuracy

```
hypothesis = model(x_train)

prediction = hypothesis >= torch.FloatTensor([0.5])

correct_prediction = prediction.float() == y_train

accuracy = correct_prediction.sum().item() / correct_prediction.shape[0]

accuracy * 100
```

```
100.0
```

```
1
```