03.06 스터디

주제: 다중분류

다중분류(Multinomial Classification)

- 데이터를 정해진 클래스 중 하나로 분류
- 세가지 이상의 클래스
- 각 클래스에 대한 확률값을 출력하고, 가장 높은 확률을 가진 클래스를 선택

$$J(\theta) = -\sum_{i=1}^{n} y_i \log\left(h_\theta(x_i)\right)$$

$$where, h_\theta(x_i) = \frac{e^{zi}}{\sum_{j=1}^{k} e^{z_j}}, k = 1, 2, \dots M$$

소프트맥스 함수

이진 분류에서 출력층에 시그모이드 함수가 있었다면 다중분류에서는 소프트맥스 함수

각 클래스에 대한 확률을 제공함

예시) cat, dog, horse클래스 [5, 4, 2] → [0.71, 0.26, 0.04]

크로스엔트로피 손실함수

정답 라벨과 모델 예측 확률(소프트맥스 결과)를 비교하는 손실 함수

예시) 정답y를 원핫 벡터로 표현: [1, 0, 0]

J = -(1 * log0.71 + 0*log0.26 + 0*log0.04) = -log0.71 ≈0.34

▼ (참고) log(h(x))값의 범위는?

9. 다음은 이진 분류를 위한 비용함수이다. 빨간 색으로 표시된 부분이 가질 수 있는 최대값과 최소값을 쓰시오

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i log(h_\theta(x_i)) + (1-y_i)log(1-h_\theta(x_i))$$

$$where, h_\theta(x_i) = \frac{1}{1+e^{-\theta x}}, \quad y \in 0, 1$$

softmax함수값의 범위가 0~1일때, −∞< log(h(x)) ≤0

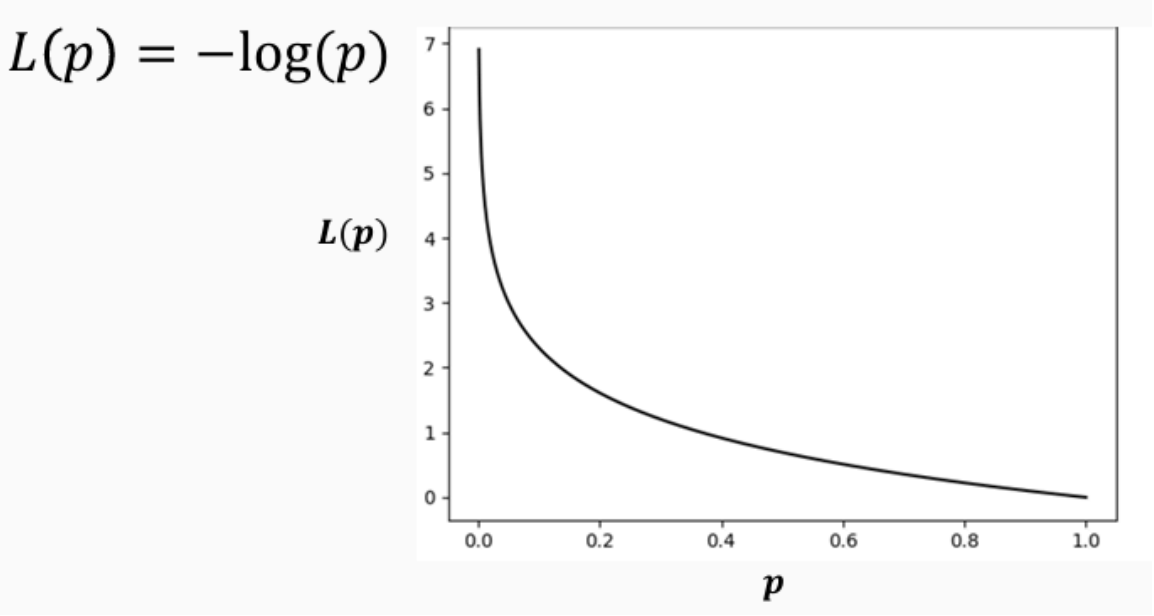
예시2) 정답이 [0,1,0]일 때

J = -(0 * log0.71 + 1*log0.26 + 0*log0.04) = -log0.26 ≈1.35

즉, 예측이 정답y를 많이 맞출수록, 정답 클래스의 확률값이 클 수록 손실함수값은 감소

NLL(Negative Log-Likelihood)

모델의 출력 확률과 실제 클래스 간의 차이를 측정하는 손실함수
예측이 정확할수록 NLL 값이 작아짐
(다중분류에서) 크로스 엔트로피와 수식적으로 동일



로그함수를 x축(p축)에 대칭한 형태

p는 확률이므로 p값의 범위는 0~1

다중 분류 정확도(Accuracy)

전체 샘플 중에서 정답을 맞춘 비율

		예측값		
		Category 1	Category 2	Category 3
실제값	Category 1	6	4	2
	Category 2	3	6	3
	Category 3	2	5	8

※(참고) 그 외 평가 지표는 구하는게 어려움

다중 분류에서 정밀도, 재현율을 구할 때는 보통 각 클래스에 대한 지표를 구하고 단순히 평균을 내거나 샘플 수 가중치를 곱해서 평균을 낸다.

$$Macro\ Average = \frac{1}{N}\sum_{k=1}^{N}(\text{각클래스의지표})$$

$$Weighted\ Average = \sum_{k=1}^{N}(\text{각클래스의지표} \times \text{각클래스데이터비율})$$

주요 메서드 & 속성

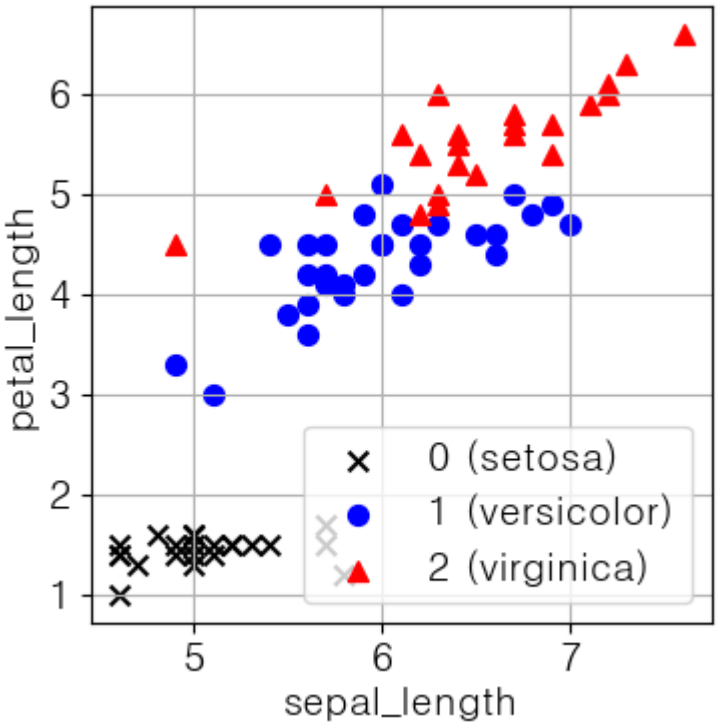
메서드/속성	설명
load_iris() iris.data iris.target	붓꽃(iris) 데이터셋을 불러오는 함수 각 데이터의 정답(품종 레이블, 0, 1, 2)
train_test_split()	학습 데이터와 테스트 데이터를 분할
nn.CrossEntropyLoss()	크로스 엔트로피 손실 함수 객체를 생성
nn.Softmax()	소프트맥스 함수 생성
nn.LogSoftmax()	로그소프트맥스 함수 생성 (Softmax값에 로그 적용)
nn.NLLLoss()	NLL 함수 생성
torch.log()	(자연)로그 함수 생성

붓꽃 데이터 산포도 (클래스3개, 변수2개)

▼ 전체 코드

```
from sklearn.datasets import load_iris

# 데이터 불러오기
iris = load_iris()

# 입력 데이터와 정답 데이터
x_org, y_org = iris.data, iris.target

# 입력 데이터로 sepal(꽃받침) length(0)와 petal(꽃잎) length(2)를 추출
x_select = x_org[:,[0,2]]

# 훈련 데이터와 검증 데이터로 분할(셔플도 동시에 실시함)
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(
    x_select, y_org, train_size=75, test_size=75,
    random_state=123)
```

```
x_t0 = x_train[y_train == 0]
x_t1 = x_train[y_train == 1]
x_t2 = x_train[y_train == 2]


plt.scatter(x_t0[:,0], x_t0[:,1], marker='x', c='k', s=50, label='0 (setosa)')
plt.scatter(x_t1[:,0], x_t1[:,1], marker='o', c='b', s=50, label='1 (versicolor)')
plt.scatter(x_t2[:,0], x_t2[:,1], marker='^', c='r', s=50, label='2 (virginica)')
plt.xlabel('sepal_length')
plt.ylabel('petal_length')
plt.legend()
plt.show()
```



다중분류 모델(입력변수 2개)

▼ 전체코드

```
## 모델 정의 및 생성
class Net(nn.Module):
```

```
    def __init__(self, n_input, n_output):
        super().__init__()
        self.l1 = nn.Linear(n_input, n_output)

    def forward(self, x):
        x1 = self.l1(x)
        return x1

n_input = x_train.shape[1]
n_output = len(list(set(y_train)))

net = Net(n_input, n_output)

## 변수 텐서화
inputs = torch.tensor(x_train).float()
labels = torch.tensor(y_train).long()

inputs_test = torch.tensor(x_test).float()
labels_test = torch.tensor(y_test).long()


lr = 0.01
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=lr)
num_epochs = 10000
history = np.zeros((0,5))

for epoch in range(num_epochs):

    optimizer.zero_grad()
    outputs = net(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()

    # 예측 라벨 산출
    predicted = torch.max(outputs, 1)[1]
```

```
    # 손실과 정확도 계산
    train_loss = loss.item()
    train_acc = (predicted == labels).sum()  / len(labels)

    outputs_test = net(inputs_test)
    loss_test = criterion(outputs_test, labels_test)
    predicted_test = torch.max(outputs_test, 1)[1]
    # predicted_test= outputs_test.argmax(-1)   # 이렇게 써도 됨

    val_loss =  loss_test.item()
    val_acc =  (predicted_test == labels_test).sum() / len(labels_test)

    if ((epoch) % 10 == 0):
        print (f'Epoch [{epoch}/{num_epochs}], loss: {train_loss:.5f} acc: {tra
        item = np.array([epoch, train_loss, train_acc, val_loss, val_acc])
        history = np.vstack((history, item))
```

```
## 각 행에서 가장 높은 점수를 가진 클래스의 인덱스
# torch.max(outputs, 1) 각 행에서 가장 큰 값과 그 인덱스
predicted = torch.max(outputs, 1)[1]

## 손실과 정확도 계산
train_loss = loss.item()
train_acc = (predicted == labels).sum()  / len(labels)

   ## 테스트 데이터에 대한 예측과 평가
outputs_test = net(inputs_test)
loss_test = criterion(outputs_test, labels_test)
predicted_test = torch.max(outputs_test, 1)[1]

   ## 테스트 데이터 손실과 정확도 계산
val_loss =  loss_test.item()
val_acc =  (predicted_test == labels_test).sum() / len(labels_test)
```

outputs: 각 클래스에 대한 점수를 나타내는 텐서

`torch.max(outputs, 1)`는 "각 행에서 가장 큰 값과 그 인덱스

predicted == labels

```
predicted = torch.tensor([1, 0, 2, 1, 0])
labels = torch.tensor([1, 0, 1, 1, 2])

print(predicted == labels)  # tensor([True, True, False, True, False])
print((predicted == labels).sum())  # 3
print((predicted == labels).sum() / len(labels))  # 0.6 (정확도)
```

다중분류 모델(입력변수 4개)

*입력 변수가 두개 일때 → x_select

```
# 입력 데이터와 정답 데이터
x_org, y_org = iris.data, iris.target

# 입력 데이터로 sepal(꽃받침) length(0)와 petal(꽃잎) length(2)를 추출
x_select = x_org[:,[0,2]]

x_train, x_test, y_train, y_test = train_test_split(
    x_select, y_org, train_size=75, test_size=75,
    random_state=123)
```

▼ 전체코드

```
x_train, x_test, y_train, y_test = train_test_split(
    x_org, y_org, train_size=75, test_size=75,
    random_state=123)

n_input = x_train.shape[1]

inputs = torch.tensor(x_train).float()
```

```
labels = torch.tensor(y_train).long()
inputs_test = torch.tensor(x_test).float()
labels_test = torch.tensor(y_test).long()

lr = 0.01

net = Net(n_input, n_output)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=lr)
num_epochs = 10000

history = np.zeros((0,5))

for epoch in range(num_epochs):


    optimizer.zero_grad()
    outputs = net(inputs)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()

    # 예측 라벨 산출
    predicted = torch.max(outputs, 1)[1]

    # 손실과 정확도 계산
    train_loss = loss.item()
    train_acc = (predicted == labels).sum()  / len(labels)

    # 예측 계산
    outputs_test = net(inputs_test)
    loss_test = criterion(outputs_test, labels_test)
    predicted_test = torch.max(outputs_test, 1)[1]

    val_loss =  loss_test.item()
    val_acc =  (predicted_test == labels_test).sum() / len(labels_test)

    if ( epoch % 10 == 0):
```

```
        print (f'Epoch [{epoch}/{num_epochs}], loss: {train_loss:.5f} acc: {tra
        item = np.array([epoch , train_loss, train_acc, val_loss, val_acc])
        history = np.vstack((history, item))
```

NLL Loss함수

입력데이터 / 정답레이블 생성 - 텐서화 - NLL손실함수 정의 -

```
outputs_np = np.array(range(1, 13)).reshape((4,-1))

# 더미 정답 데이터(샘플의 정답 클래스 인덱스)
labels_np = np.array([0, 1, 2, 0])

# 텐서화
outputs_dummy = torch.tensor(outputs_np).float()    #강의자료는 .없이 출력됨(버
labels_dummy = torch.tensor(labels_np).long()

# 결과 확인
print(outputs_dummy.data)
print(labels_dummy.data)

nllloss = nn.NLLLoss()
loss = nllloss(outputs_dummy, labels_dummy) # -(1 + 5 + 9 + 10)/4 = -6.25
print(loss.item())

<출력>
tensor([[ 1.,  2.,  3.],
        [ 4.,  5.,  6.],
        [ 7.,  8.,  9.],
        [10., 11., 12.]])
tensor([0, 1, 2, 0])
-6.25
```

NLL함수면 코드에서 로그연산이 있어야하지 않나?

원래 `NLLLoss`는 입력으로 로그 확률을 받아야 함

모델 클래스측에 LogSoftmax 함수를 포함

```
class Net(nn.Module):
    def __init__(self, n_input, n_output):
        super().__init__()
        self.l1 = nn.Linear(n_input, n_output)

        # logsoftmax 함수 정의
        self.logsoftmax = nn.LogSoftmax(dim=1) # 열 방향으로 연산)


    def forward(self, x):
        x1 = self.l1(x)
        x2 = self.logsoftmax(x1)
        return x2
```

모델 클래스측에 소프트맥스 함수만 포함된 경우

```
class Net(nn.Module):
    def __init__(self, n_input, n_output):
        super().__init__()
        self.l1 = nn.Linear(n_input, n_output)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x1 = self.l1(x)
        x2 = self.softmax(x1)
        return x2

lr = 0.01
net = Net(n_input, n_output)
```

```
criterion = nn.NLLLoss()
optimizer = optim.SGD(net.parameters(), lr=lr)
num_epochs = 10000
history = np.zeros((0,5))

for epoch in range(num_epochs):


    optimizer.zero_grad()
    outputs = net(inputs)

    # 여기서 로그 함수를 적용함
    outputs2 = torch.log(outputs)

    loss = criterion(outputs2, labels)
    loss.backward()
    optimizer.step()
    predicted = torch.max(outputs, 1)[1]

    train_loss = loss.item()
    train_acc = (predicted == labels).sum()  / len(labels)

    outputs_test = net(inputs_test)

    # 여기서 로그 함수를 적용함
    outputs2_test = torch.log(outputs_test)

    loss_test = criterion(outputs2_test, labels_test)
    predicted_test = torch.max(outputs_test, 1)[1]

    val_loss =  loss_test.item()
    val_acc =  (predicted_test == labels_test).sum() / len(labels_test)

    if epoch % 10 == 0:
        print (f'Epoch [{epoch}/{num_epochs}], loss: {train_loss:.5f} acc: {train_a
        item = np.array([epoch , train_loss, train_acc, val_loss, val_acc])
        history = np.vstack((history, item))
```