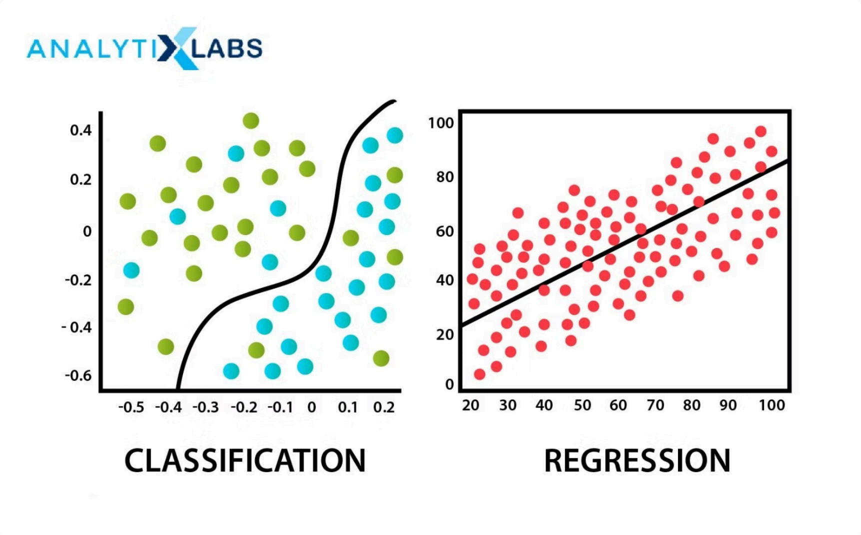
분류

- 분류는 데이터 항목에 개별 레이블을 할당.
- 분류의 기본 개념, 수학적 표기법, 다양한 분류기 유형 및 성능 측정 방법.
- 선형 회귀, 로지스틱 회귀 및 소프트맥스 회귀 분류 알고리즘 소개.



분류의 공식 표기법

- 분류기는 y = f(x) 함수로 표현되고, x는 입력 데이터 항목이고 y는 출력 범위. 입력 벡터 x는 독립 변수, 출력 y는 종속 변수라 함.
- 이산 변수를 처리할 때는 순서형과 명목형을 구분하고, 명목형 변수의 경우 원-핫 인코딩 기술을 사용하여 처리함.

1. 입력 데이터

   독립 변수 x로 표현

2. 분류 함수

   y = f(x) 형태의 함수

3. 출력 범주

   종속 변수 y로 표현

4. 이산 변수 처리

   순서형 또는 명목형으로 구분

이산 변수 처리

순서형 변수(ordinal)

- **명확한 순서**가 있지만, 그 간격이 일정하지 않은 범주형 변수.
- 예시) 매우 불만족, 불만족, 보통, 만족, 매우 만족
- 초등학교, 중학교, 고등학교, 대학교, 대학원

명목형 변수

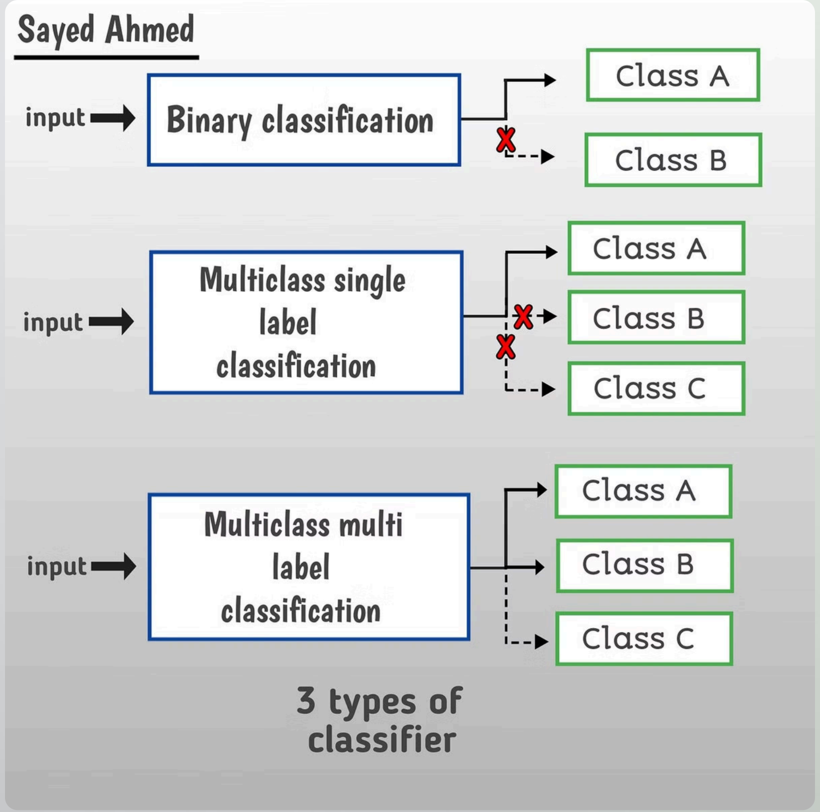
- 값들 간에 **순서**나 **크기**가 없는 **범주형 변수.**
- 예시) {바나나, 사과, 오렌지} 집합의 요소는 자연스러운 순서가 없음.
- **국적**: 미국, 한국, 일본, 독일 등

명목형 변수

- 데이터셋의 명목형 변수를 표현하는 방법은 각 레이블에 숫자를 할당.

명목형 변수 처리 방법

- **Label Encoding**: {바나나, 사과, 오렌지} 집합을 {0, 1, 2}로 처리함.
- **One-Hot Encoding**: 이진값(0 또는 1)을 가지는 **새로운 변수**로 변환
- **Dummy Encoding**: 모든 범주를 변환하지 않고 하나의 기준 범주를 제외하고 변환, 예시)국가: 미국 → [0, 0], 한국 → [1, 0], 일본 → [0, 1]여기서 미국이 기준범주가 됨.
- **Frequency Encoding**: 국가: 미국 → 500, 한국 → 300, 일본 → 200 (각 국가가 나타난 빈도)



분류기의 종류

유형	장점	단점
선형 회귀	구현하기 쉬움	성능이 보장되지 않음, 이진 레이블만 지원
로지스틱 회귀	매우 정확함, 사용자 정의 조정을 위한 유연한 정규화 방법	이진 레이블만 지원
소프트맥스 회귀	다중 클래스 분류 지원	구현하기 더 복잡함

원-핫 인코딩

원-핫 인코딩이란?

명목 변수의 각 값에 대해 더미 변수를 추가.

예시

과일 변수는 제거되고 바나나, 사과, 오렌지라는 세 개의 별도 변수로 대체. 각 변수는 해당 과일의 범주가 참인지 여부에 따라 0 또는 1 값.

장점

명목형 변수를 수치형으로 변환하여 많은 기계 학습 알고리즘에서 사용함.

		Banana	Apple	Orange
1	Banana	1.0	0.0	0.0
2	Apple	0.0	1.0	0.0
3	Orange	0.0	0.0	1.0

선형 회귀를 이용한 분류

선형 회귀 모델은 f(x) = wx+b 형태의 선형 함수.

데이터 준비

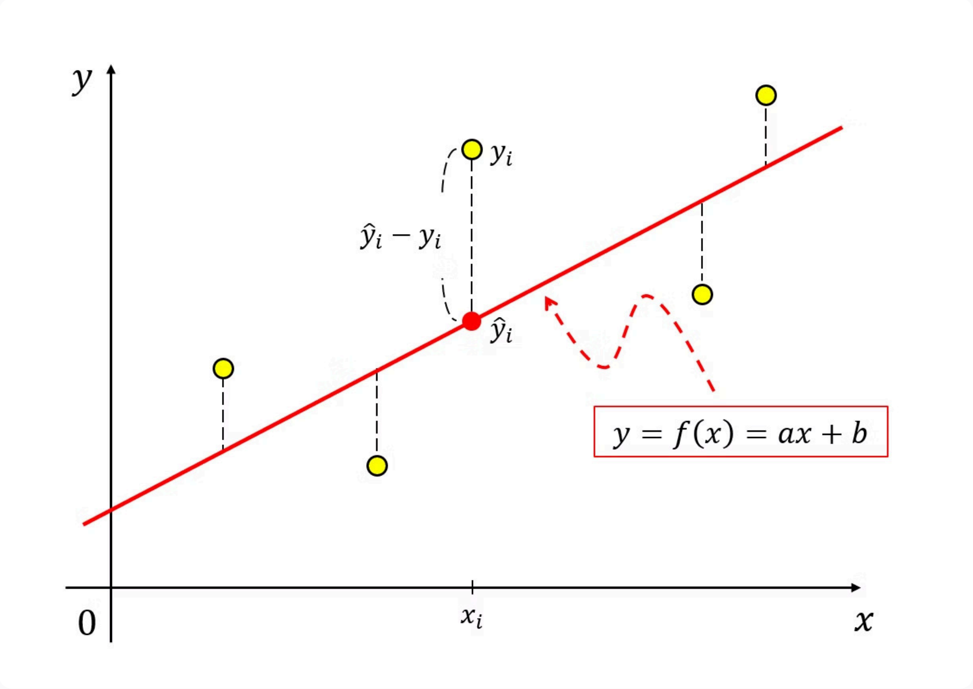
이진 클래스로 레이블된 데이터셋을 준비.

선형 모델 적용

데이터에 선형 회귀 모델을 적용. 활성함수는 선형함수(linear) 사용. 비용함수는 mean_squared_error임

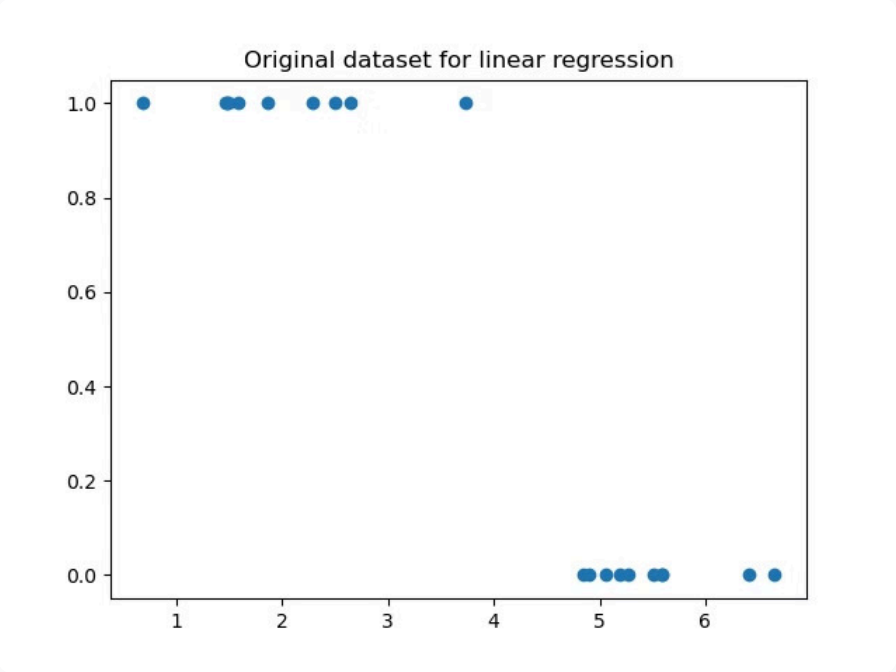
임계값 설정

분류를 위한 임계값을 결정.



선형 회귀 파이썬 코드-입력데이터

```
x_label0 = np.random.normal(5, 1, 10)#평균 5, 표준편차 1, 데이터수 10
x_label1 = np.random.normal(2, 1, 10)
x_train = np.append(x_label0, x_label1)
y_train = [0.] * len(x_label0) + [1.] * len(x_label1)
plt.scatter(x_train, y_train)
```



선형 회귀 파이썬 코드-모델 구축

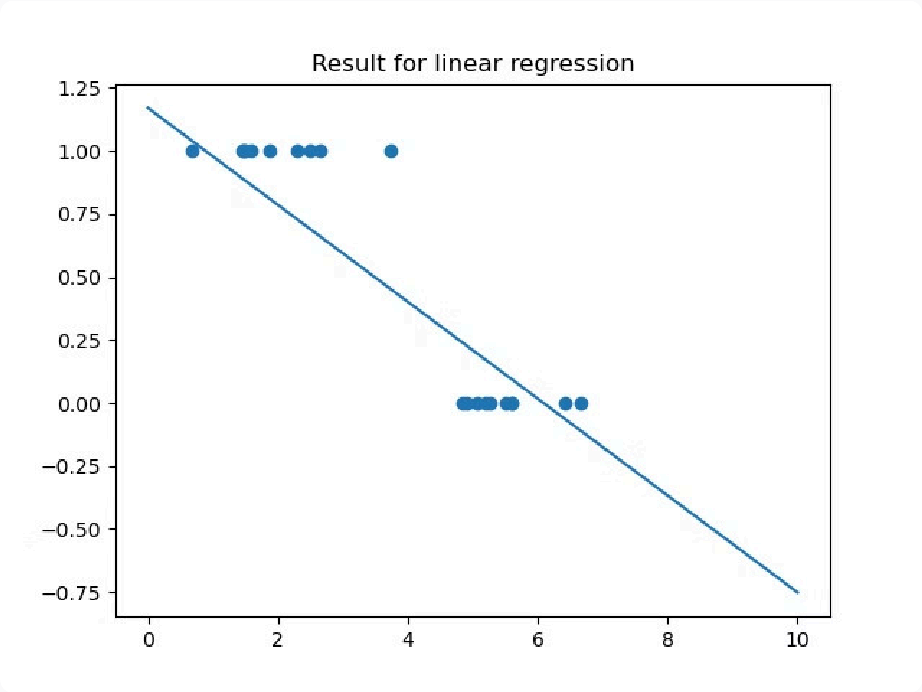
- f(x) = wx+b 에서 w와 b(바이어스)의 최적값 찾기.
- 비용함수: mean_squared_error 사용.
- SGD 경사하강법 사용.

```
learning_rate = 0.001
training_epochs = 5000

model = models.Sequential([
    layers.Input(shape=(1,)),
    layers.Dense(units=1,use_bias=True)#바이어스 포함
])
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=learning_rate),
              loss='mean_squared_error')
model.fit(X, Y, epochs=training_epochs, verbose=2)
```

선형 회귀 파이썬 코드-결과

- f(x) = wx+b 에서 w와 b(바이어스)의 최적값 찾음. w=w_val[0][0][0], b=w_val[1][0]

```
w_val = model.layers[0].get_weights()
print('learned parameters: ', w_val)

plt.scatter(x_train, y_train)
plt.title('Original dataset for linear regression')
all_xs = np.linspace(0, 10, 100)
plt.plot(all_xs, all_xs * w_val[0][0][0] + w_val[1][0])
plt.title('Result for linear regression')
```



선형 회귀의 한계

1

이상치 민감성

극단적인 데이터 포인트가 전체 모델에 큰 영향을 미침.

2

선형성 가정

실제 데이터가 선형 관계를 따르지 않을 수 있음.

3

임계값 설정의 어려움

적절한 분류 임계값을 결정하는 것이 어려움.

4

확률 해석의 부재

출력을 확률로 해석하기 어려움.

```
#임계값(0.5)를 설정하여 0과1로 분류
y_pred = all_xs * w_val[0][0][0] + w_val[1][0]
y_pred_binary = (y_pred > 0.5).astype(int).flatten()
```

로지스틱 회귀를 이용한 분류

로지스틱 회귀는 선형 회귀의 한계를 극복하기 위한 방법으로써, 이 방법은 선형 회귀와 유사하지만, 다른 활성함수와 비용 함수 사용.

선형 모델

기본적인 선형 모델 y(x) = wx를 시작점으로 사용.

시그모이드 함수 적용

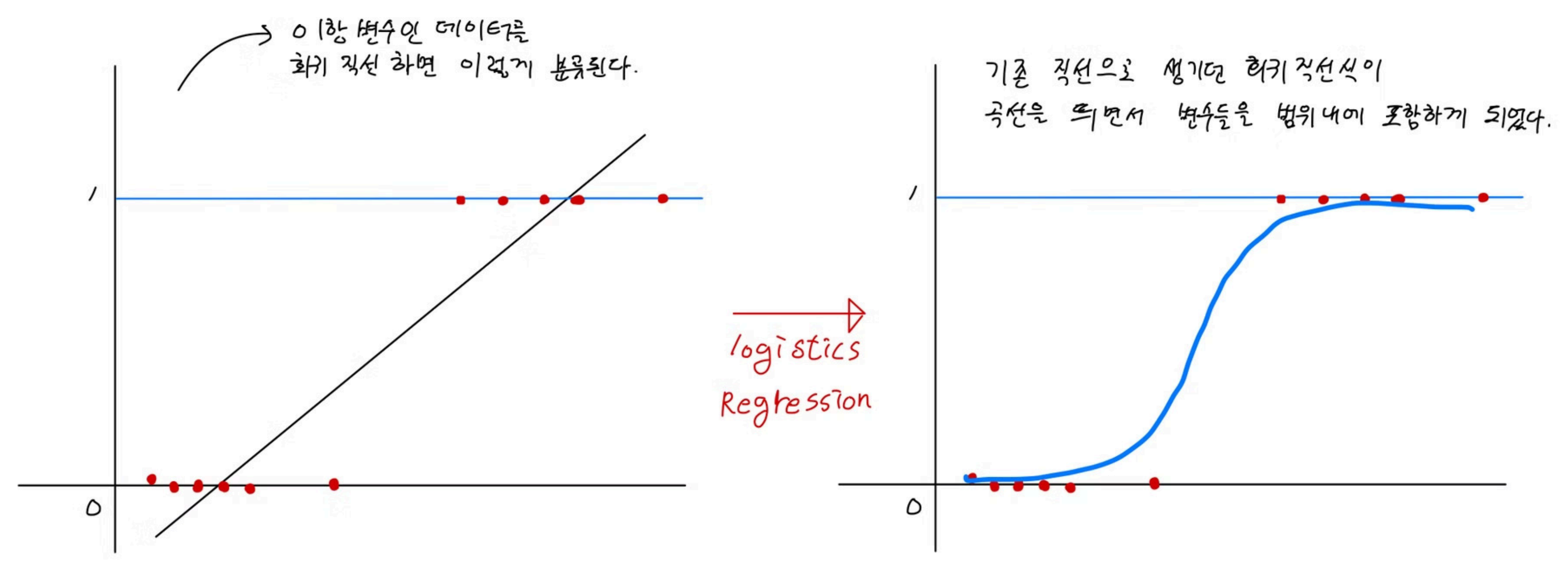
활성함수는 시그모이드 함수를 적용하여 0과 1 사이의 값으로 변환.

새로운 비용 함수

로그 손실(log loss) 또는 binary_crossentropy 비용 함수.

최적화

경사 하강법 등의 최적화 알고리즘을 사용하여 모델 파라미터를 학습.



1D로지스틱 회기-모델 구축

- 활성함수 'sigmoid'사용
- 비용함수 'binary_crossentropy' 사용

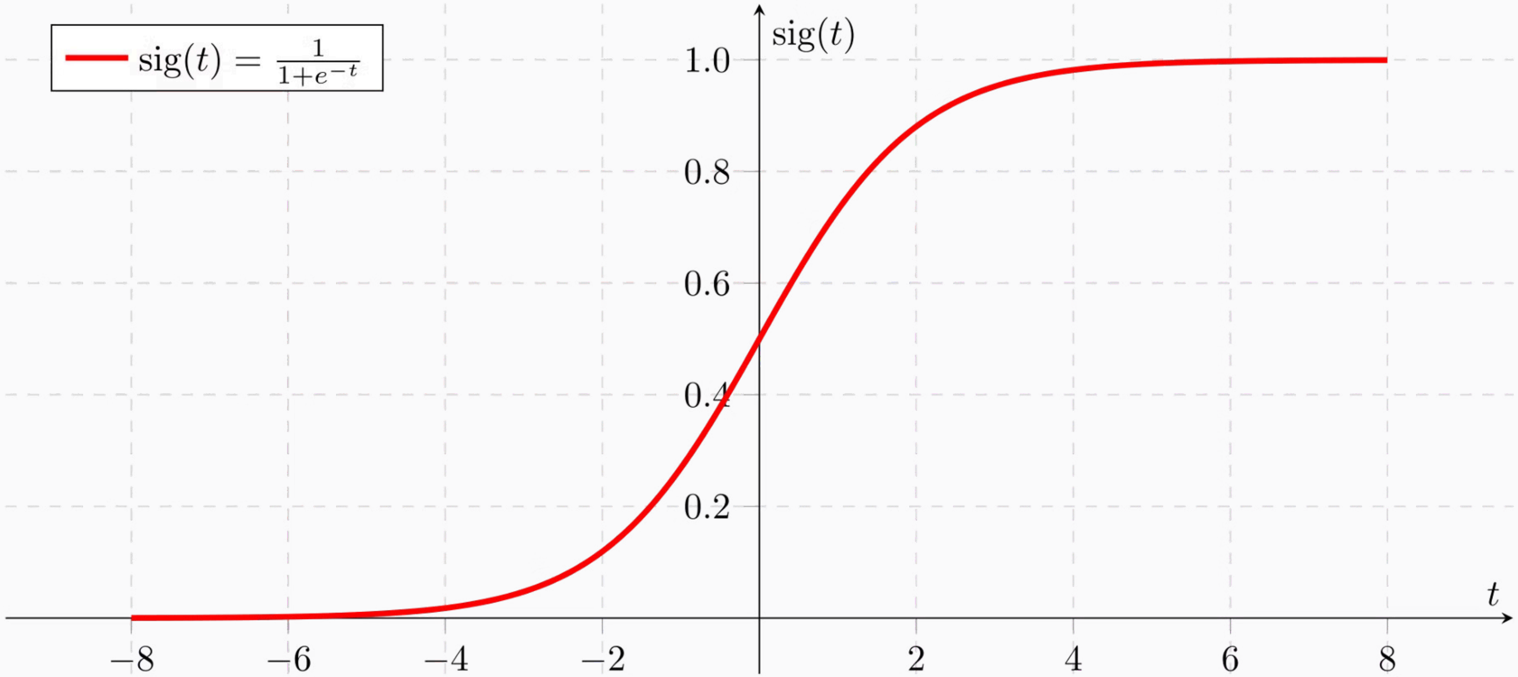
```
model = models.Sequential([
    layers.Input(shape=(1,)),
    layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=learning_rate),
              loss='binary_crossentropy')
model.fit(X, Y, epochs=training_epochs, verbose=2)
```

시그모이드 함수

로지스틱 회귀에서 핵심적인 역할은 시그모이드 활성함수. 이 함수는 연속적이고 미분 가능함.

1. x가 0일 때, 함수 값은 0.5(임계값).
2. x가 증가할수록 함수 값은 1에 수렴.
3. x가 음의 무한대로 감소할수록 함수 값은 0에 수렴.

이러한 특성 때문에 시그모이드 함수는 이진 분류에 매우 적합.

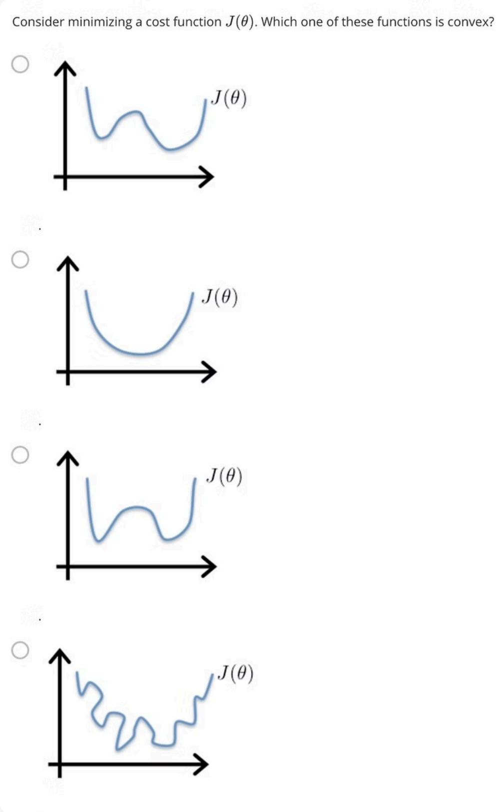


로지스틱 회귀 비용 함수의 수학적 표현

로지스틱 회귀에서 사용되는 비용 함수는 실제 값(y)과 모델 예측값(h) 사이의 차이를 측정. 수학적으로 표현하면 다음과 같음. J(θ) = BCE(binary_crossentropy)

$$BCE = -\frac{1}{N}\sum_{i=0}^{N} y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i)$$

여기서 N은 학습 데이터의 개수이고, y는 실제 정답, y^는 모델이 예측한 확률값. 이 함수는 아래로 볼록 함수이므로 최적화 과정에서 전역 최소값을 찾기 쉬움.



1D 로지스틱 회귀-경사하강법

로지스틱 회귀의 비용함수 :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

경사 하강법 :

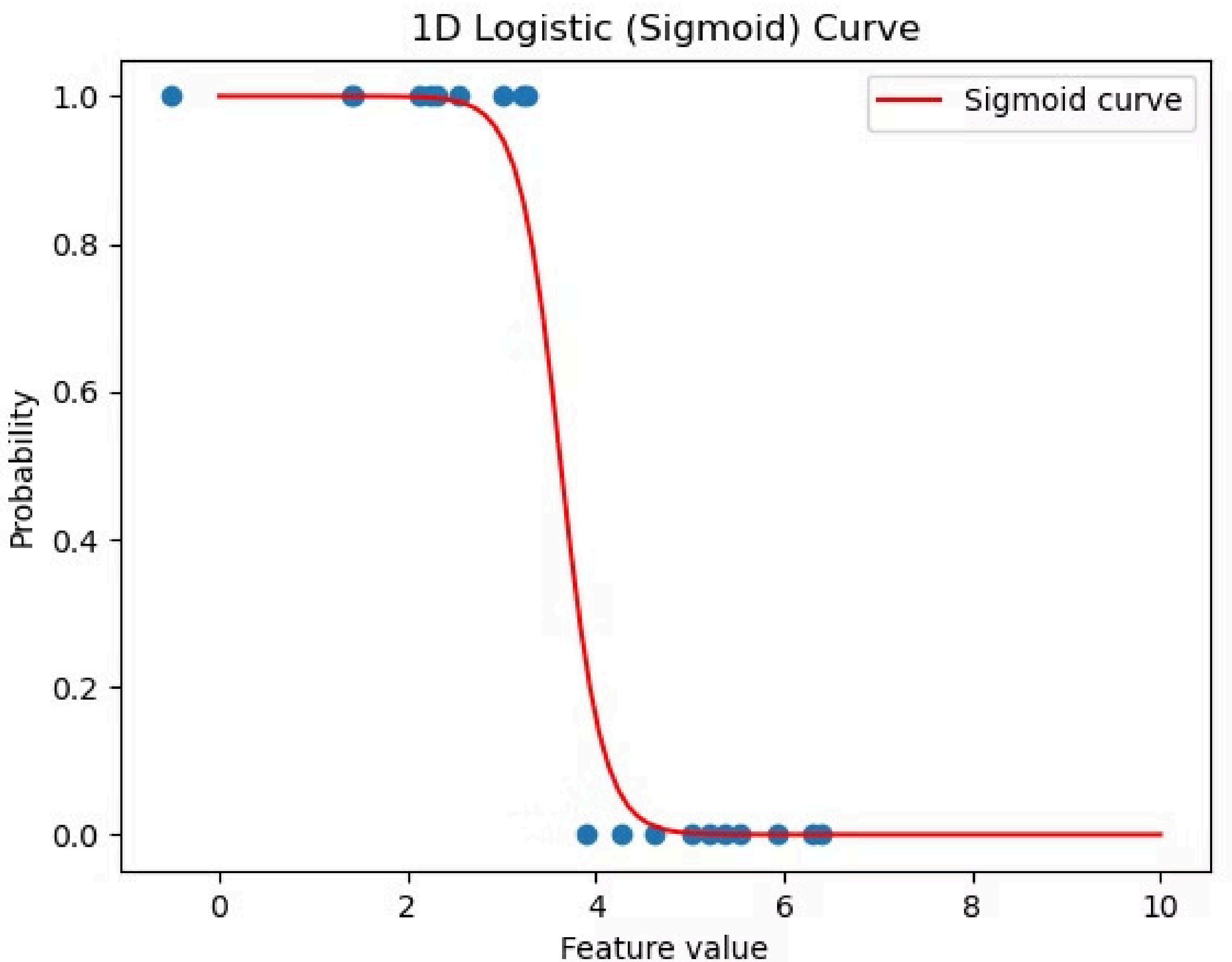
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad (\text{동시업데이트 } j = 0, 1, \dots, n)$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

1D로지스틱 회기-결과 코드

```
w_val = model.layers[0].get_weights()
print('learned parameters: ', w_val)

x_test = np.linspace(0, 10, 100)
x_test = tf.convert_to_tensor(x_test,dtype=tf.float32)
y_test = model.predict(x_test)

plt.scatter(x_train, y_train)
plt.plot(x_test, y_test, '-r', label='Sigmoid curve')
plt.xlabel('Feature value')
plt.ylabel('Probability')
plt.title('1D Logistic (Sigmoid) Curve')
plt.legend()
```

1D로지스틱 회기-결과 그래프



로지스틱 회귀의 장점

1. **해석 용이성**

   모델의 결과를 확률로 쉽게 해석 가능.

2. **효율성**

   계산이 빠르고 대규모 데이터에도 적용 가능.

3. **과적합 방지**

   선형 모델 기반이므로 과적합 위험이 낮음.

4. **정규화 용이**

   L1, L2 정규화를 쉽게 적용 가능.

로지스틱 회귀의 한계

1. **선형성 제약**

   복잡한 비선형 관계 모델링 어려움.

2. **특성 독립성 가정**

   특성 간 상호작용을 고려하지 않음.

3. **이상치 민감성**

   극단적인 이상치에 영향을 받음.

4. **클래스 불균형**

   불균형한 데이터셋에서 성능이 저하됨.

분류 성능 평가-혼동 행렬

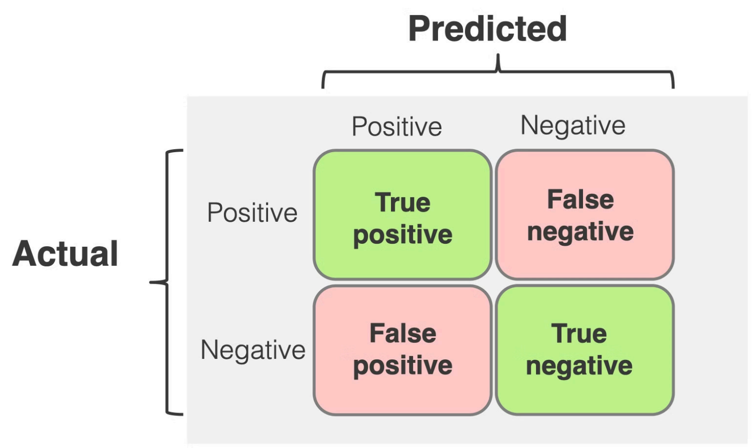
- 혼동 행렬은 분류기의 성능을 더 자세히 분석할 수 있는 도구.
- 예측된 클래스와 실제 클래스를 비교하는 표. 이진 분류의 경우, 혼동 행렬은 다음 네 가지 범주로 구성됨.

진양성(TP): 올바르게 양성으로 예측

위양성(FP): 잘못 양성으로 예측

위음성(FN): 잘못 음성으로 예측

진음성(TN): 올바르게 음성으로 예측



분류 성능 평가-정밀도/정확도/재현율/F1

1 정밀도 (Precision)

양성 예측의 정확성을 측정. 높은 정밀도는 거짓 양성이 적음을 의미.

TP / (TP + FP)

2 재현율 (Recall)

실제 양성 중 양성으로 발견된 비율을 측정. 높은 재현율은 거짓 음성이 적음을 의미.

TP / (TP + FN)

3 정확도

(정답 수) / (전체 문제 수)

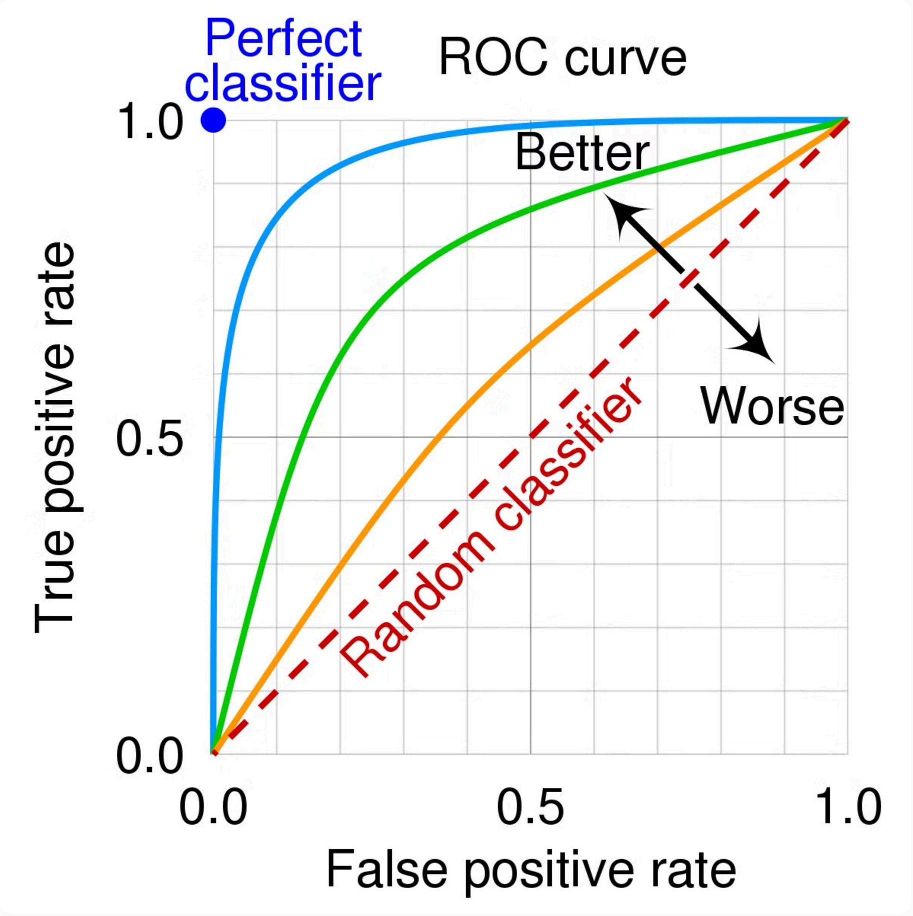
TP / (TP+TN+FP+FN)

4 F1 점수

정밀도와 재현율의 조화 평균으로, 두 지표의 균형을 나타냄.

2*(정밀도*재현율) / (정밀도+재현율)

이진 분류기 성능 측정법-수신기 작동 특성(ROC) 곡선



- ROC 곡선은 x축은 거짓 양성 비율을, y축은 참 양성 비율을 비교할 수 있게 해주는 그래프.
- 두 곡선이 교차하지 않을 때 한 방법이 다른 방법보다 확실히 우수. 좋은 알고리즘은 기준선보다 훨씬 위에 있음.
- 곡선 아래 면적(AUC)을 계산하여 분류기의 성능을 평가.

ROC 곡선의 해석

ROC 곡선을 해석할 때는 곡선 아래 면적(AUC)을 함.

우수한 분류기

AUC > 0.9

보통의 분류기

0.7 < AUC < 0.9

약한 분류기

0.5 < AUC < 0.7

무작위 추측

AUC ≈ 0.5

2D 로지스틱 회기 구현

2D 로지스틱 회귀에는 두 개의 독립 변수를 기반으로 데이터 포인트를 분류하는 작업.

1

Data Representation

2D 평면(위도 대 경도)에 범죄 활동을 플롯.

2

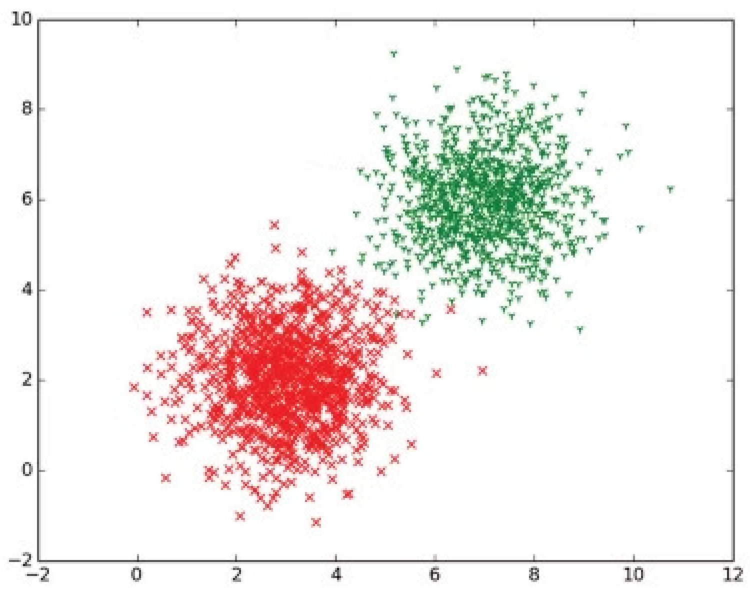
Cluster Identification

각 갱단의 활동 클러스터를 식별 (예: (3,2) 및 (7,6) 주변)

3

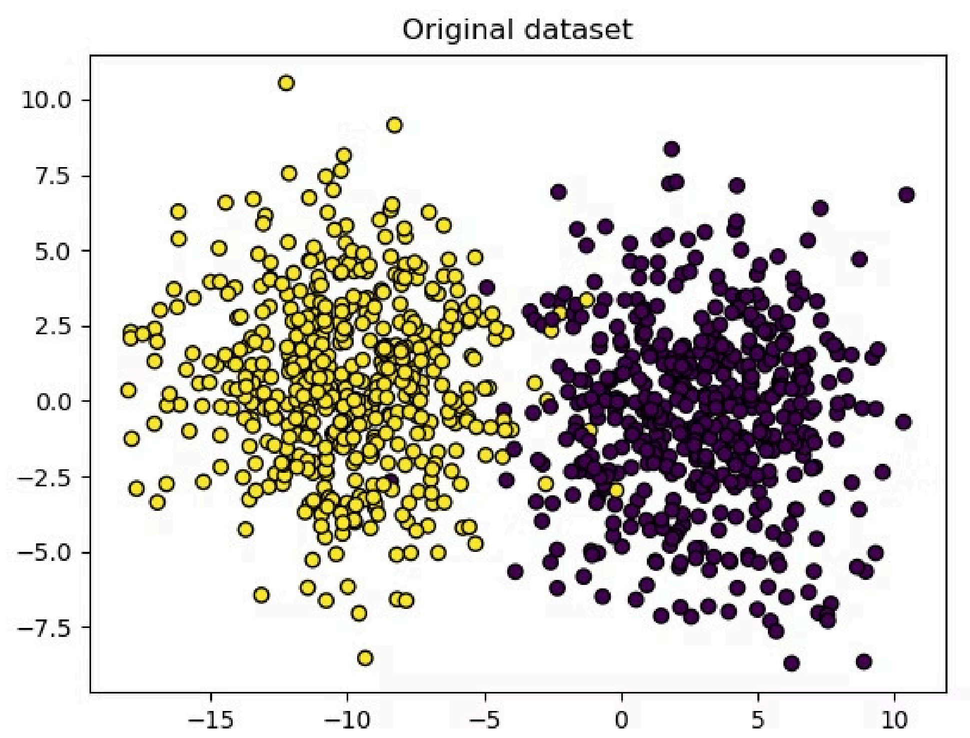
Classification

특정 위치에서 새로운 범죄를 저지른 갱단이 누구인지 확인함.



2D 로지스틱 회기 파이썬 코드-입력데이터

```
from sklearn.datasets import make_blobs
x_train, y_train = make_blobs(n_samples=1000, centers=2, n_features=2,cluster_std=3.0)

plt.scatter(x_train[:, 0], x_train[:, 1],c=y_train, edgecolors='k', marker='o')
plt.title('Original dataset')
```



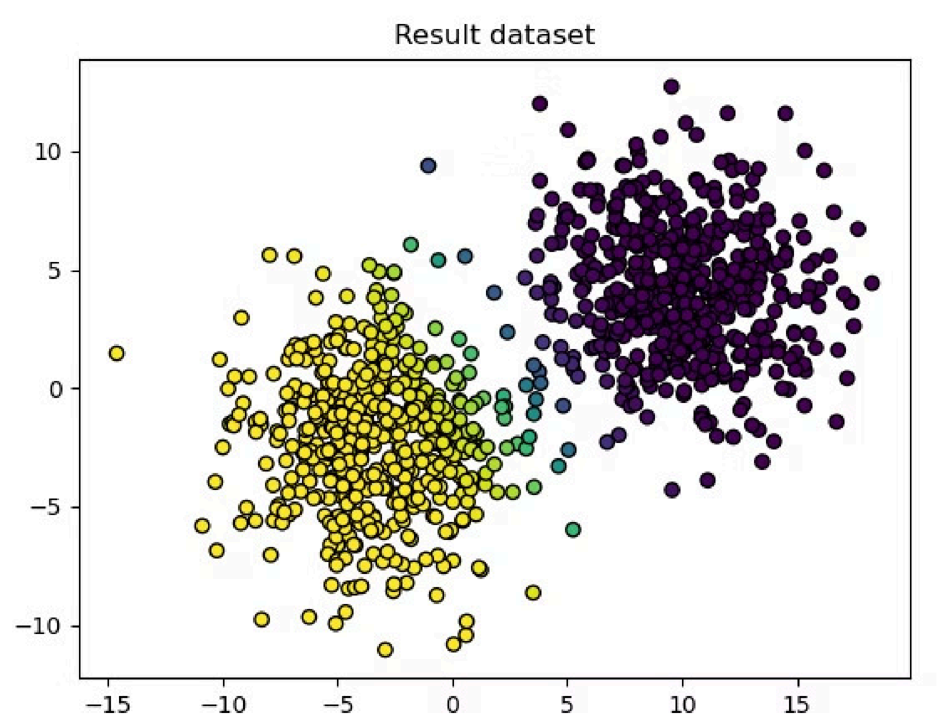
2D로지스틱 회기-모델 구축

- 활성함수 'sigmoid'사용
- 비용함수 'binary_crossentropy' 사용
- layers.Input(shape=(2,)), 2D이므로 "2"사용

```
learning_rate=0.01
learning_epo5chs=0
batch_size = 8

model = models.Sequential([
    layers.Input(shape=(2,)),
    layers.Dense(1, activation='sigmoid')
])
optimizer = SGD(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
x_train = tf.convert_to_tensor(x_train,dtype=tf.float32)
y_train = tf.convert_to_tensor(y_train,dtype=tf.float32)

history = model.fit(x_train, y_train, epochs=learning_epochs, batch_size=batch_size)
```

2D로지스틱 회기-결과 코드

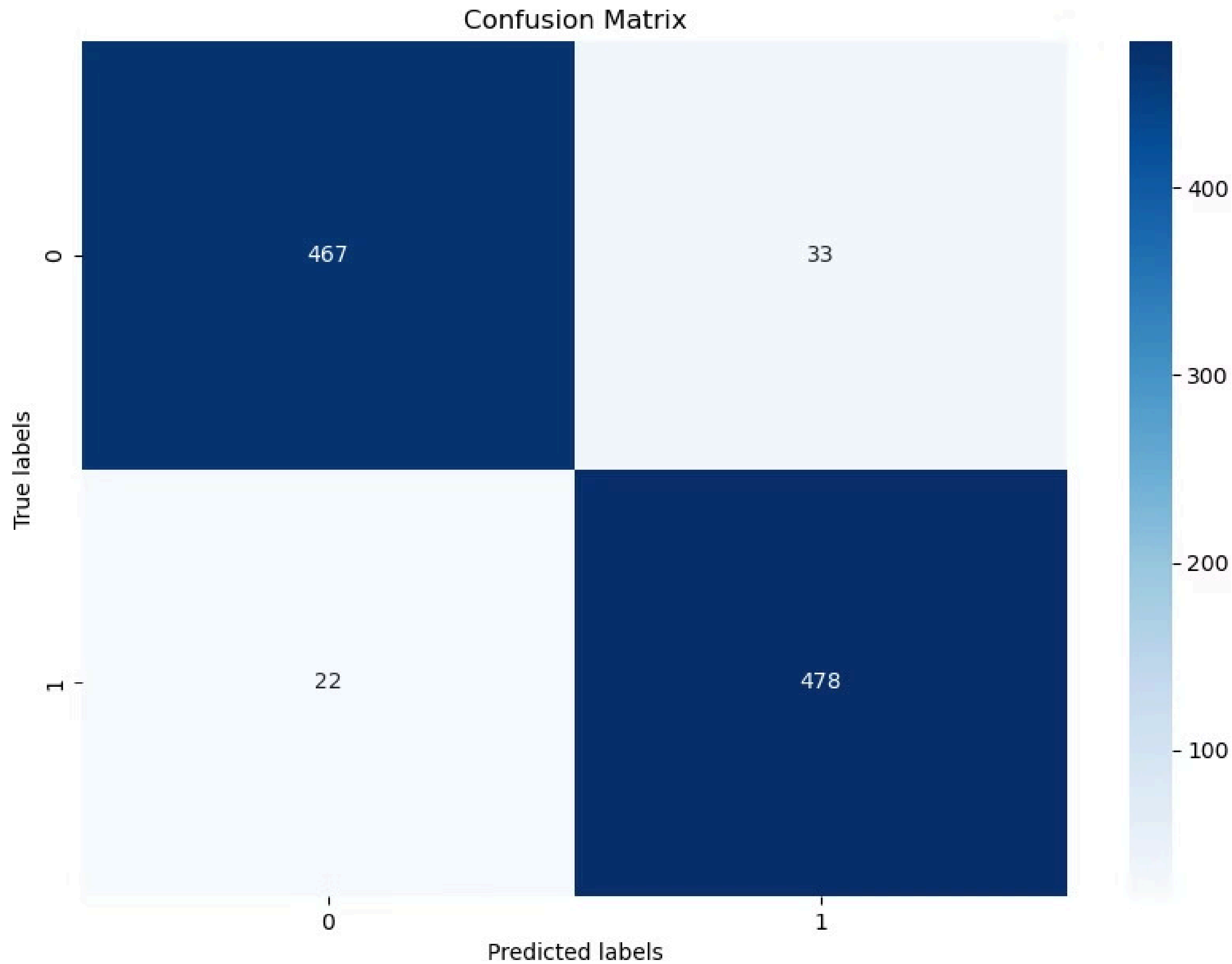
```
y_pred = model.predict(x_train)
plt.figure()
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_pred, cmap='viridis', edgecolors='k', marker='o')
plt.title('Result dataset')

accuracy_history = history.history['accuracy']
# 정확도 평균 계산
average_accuracy = np.mean(accuracy_history)
print(f"Training Accuracy 평균: {average_accuracy:.2f}")
```



2D로지스틱 회기-혼동 행렬 코드

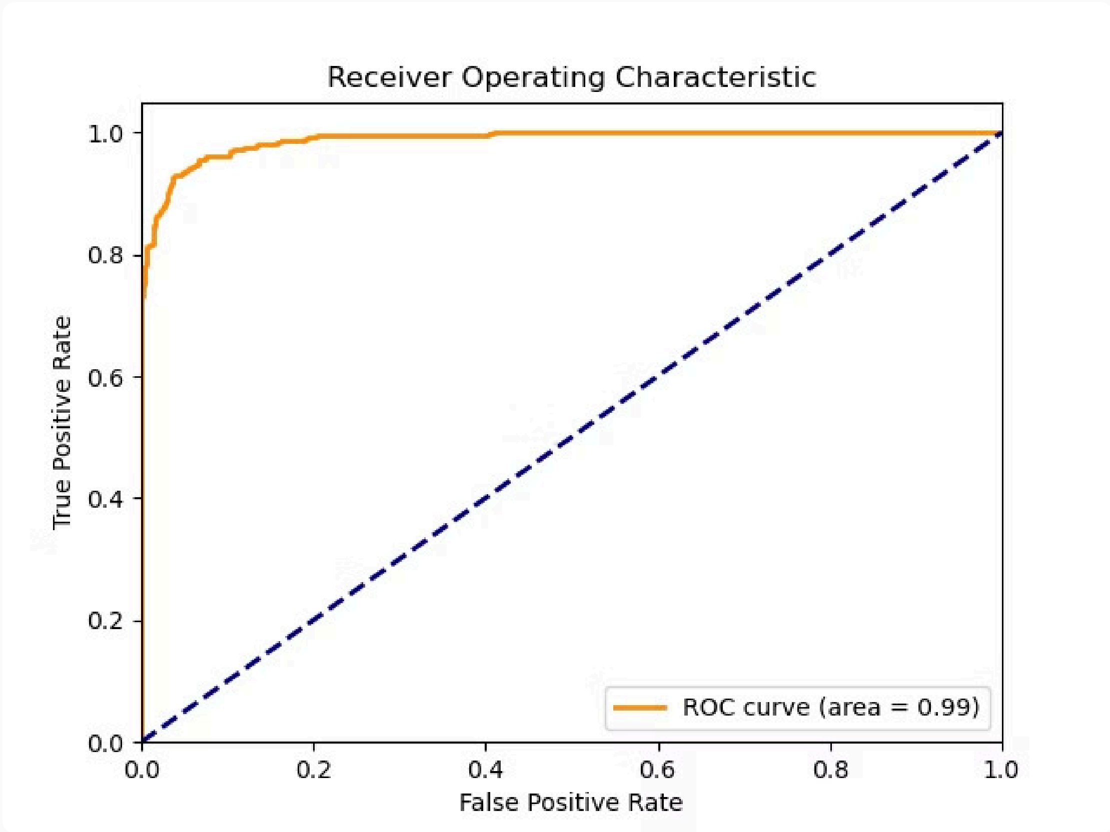
```
y_predictions = model.predict(x_train)
y_pred = (y_predictions > 0.5).astype(int).flatten()
cm = confusion_matrix(y_train, y_pred)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')

# 성능지수 계산
accuracy = accuracy_score(y_train, y_pred)
precision = precision_score(y_train, y_pred)
recall = recall_score(y_train, y_pred)
f1 = f1_score(y_train, y_pred)
```

2D로지스틱 회기-혼동 행렬 테이블



2D로지스틱 회기-ROC그래프

```
# ROC 곡선 그리기
y_pred_prob = model.predict(x_train)
fpr, tpr, thresholds = roc_curve(y_train, y_pred_prob)
roc_auc = auc(fpr, tpr) #AUC계산
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
```



소프트맥스 회귀를 이용한 분류

3D이상의 다중 클래스 분류

모델 구축

softmax 함수를 사용하여 다중 클래스 분류 모델을 정의.

비용 함수 정의

categorical_crossentropy비용 함수를 사용하여 모델의 성능을 측정.

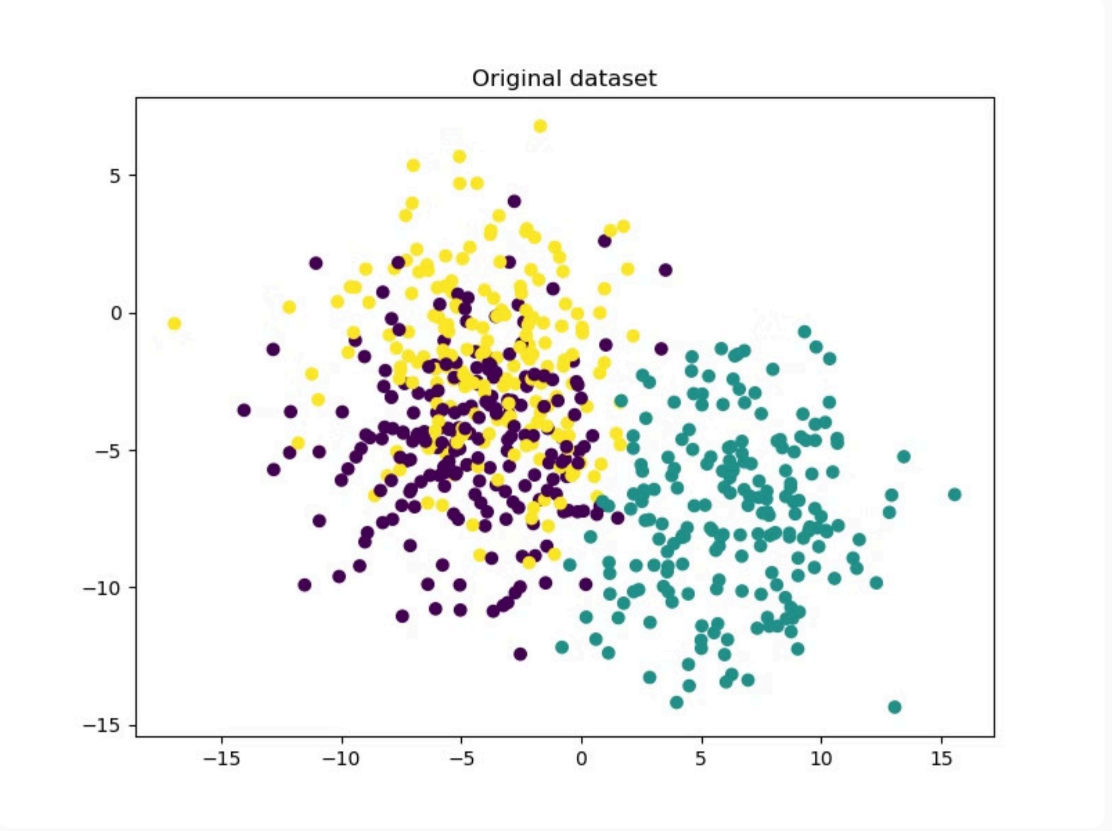
최적화

경사 하강법을 사용하여 모델 파라미터를 최적화.

레이블은 원-핫 인코딩을 사용하여 벡터로 표현.

3D 소프트맥스 회기 파이썬 코드-입력데이터

```
x_train, y_train = make_blobs(n_samples=2000, centers=3, n_features=3,cluster_std=3.0)
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.3, shuffle=True)
```



3D 소프트맥스 회기-모델 구축

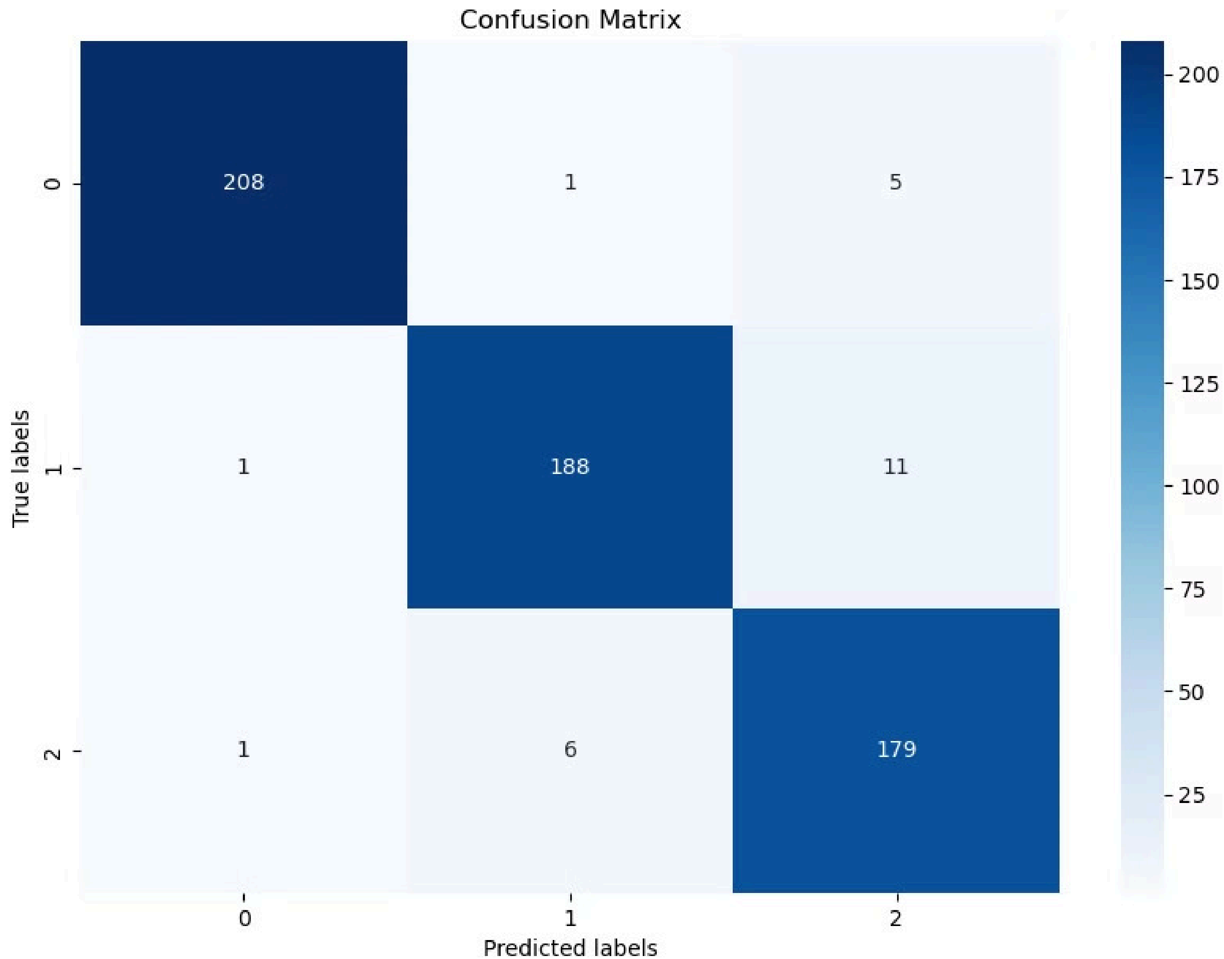
- 활성함수 'softmax'사용
- 비용함수 'categorical_crossentropy' 사용
- layers.Input(shape=(3,)), 3D이므로 "3"사용

```
model = models.Sequential([
    layers.Input(shape=(3,)),
    layers.Dense(20, activation='relu'),
    layers.Dense(3, activation='softmax')
])
learning_rate=0.01
learning_epochs=50
batch_size = 8

optimizer = SGD(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

encoder = OneHotEncoder(sparse_output=False)
y_train = encoder.fit_transform(y_train.reshape(-1,1))
y_test = encoder.fit_transform(y_test.reshape(-1,1))

x_train = tf.convert_to_tensor(x_train,dtype=tf.float32)
y_train = tf.convert_to_tensor(y_train,dtype=tf.float32)
history = model.fit(x_train, y_train, epochs=learning_epochs, batch_size=batch_size)
```

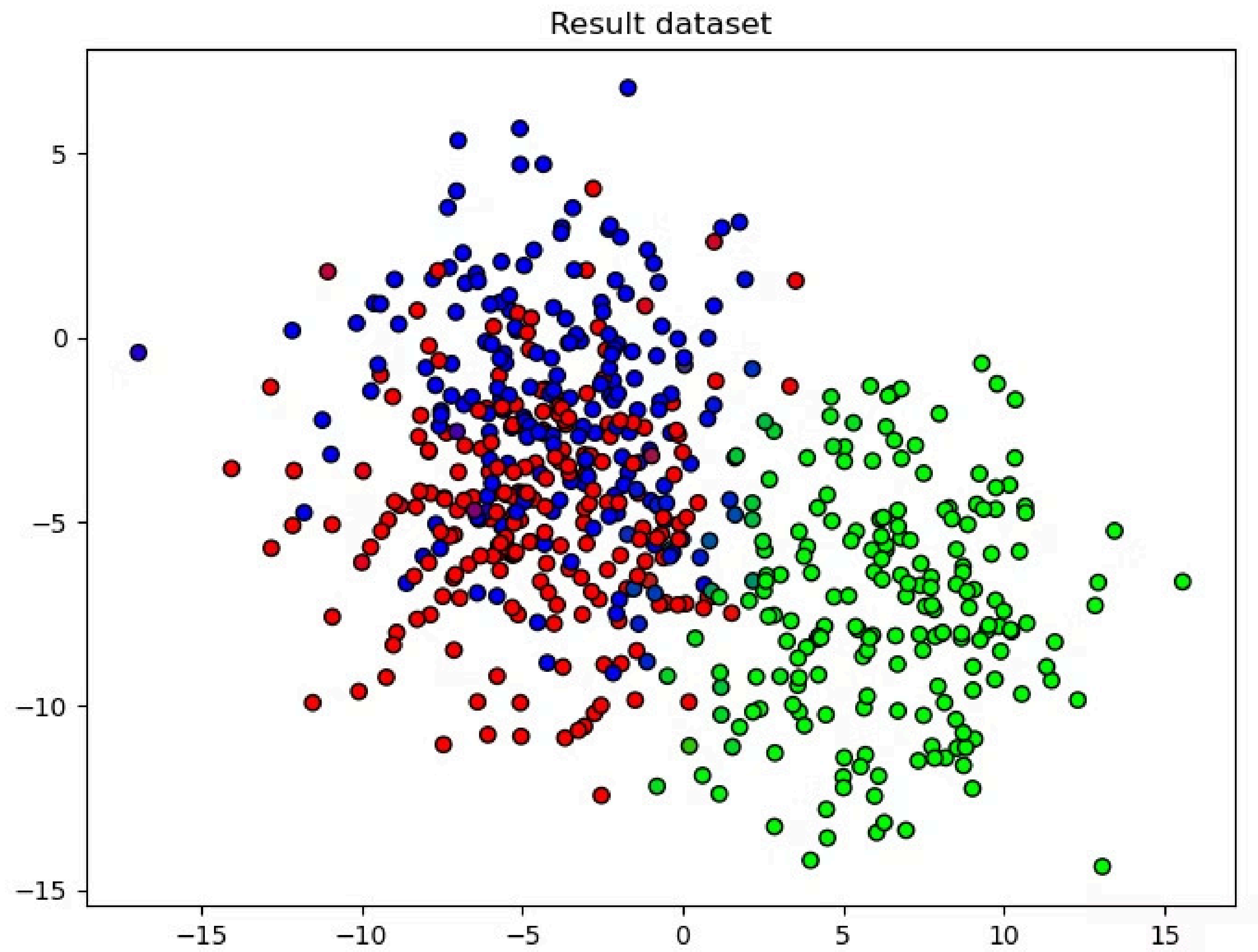
3D 소프트맥스 회기-결과 코드

```
predictions = model.predict(x_test)
y_prediction = np.argmax(predictions, axis=1)
y_label = np.argmax(y_test, axis=1)
correct_predictions = np.sum(y_prediction == y_label)
accuracy = correct_predictions / x_test.shape[0] * 100
print(f"Accuracy: {accuracy:.2f}%")

cm = confusion_matrix(y_label, y_prediction)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')

plt.figure(figsize=(8,6))
plt.scatter(x_test[:, 0], x_test[:, 1], c=predictions, cmap='viridis', edgecolors='k', marker='o')
plt.title('Result dataset')

plt.figure(figsize=(8,6))
plt.scatter(x_test[:,0], x_test[:,1],c=y_prediction)
plt.title('Original dataset')
```

3D 소프트맥스 회기-혼동 행렬 테이블



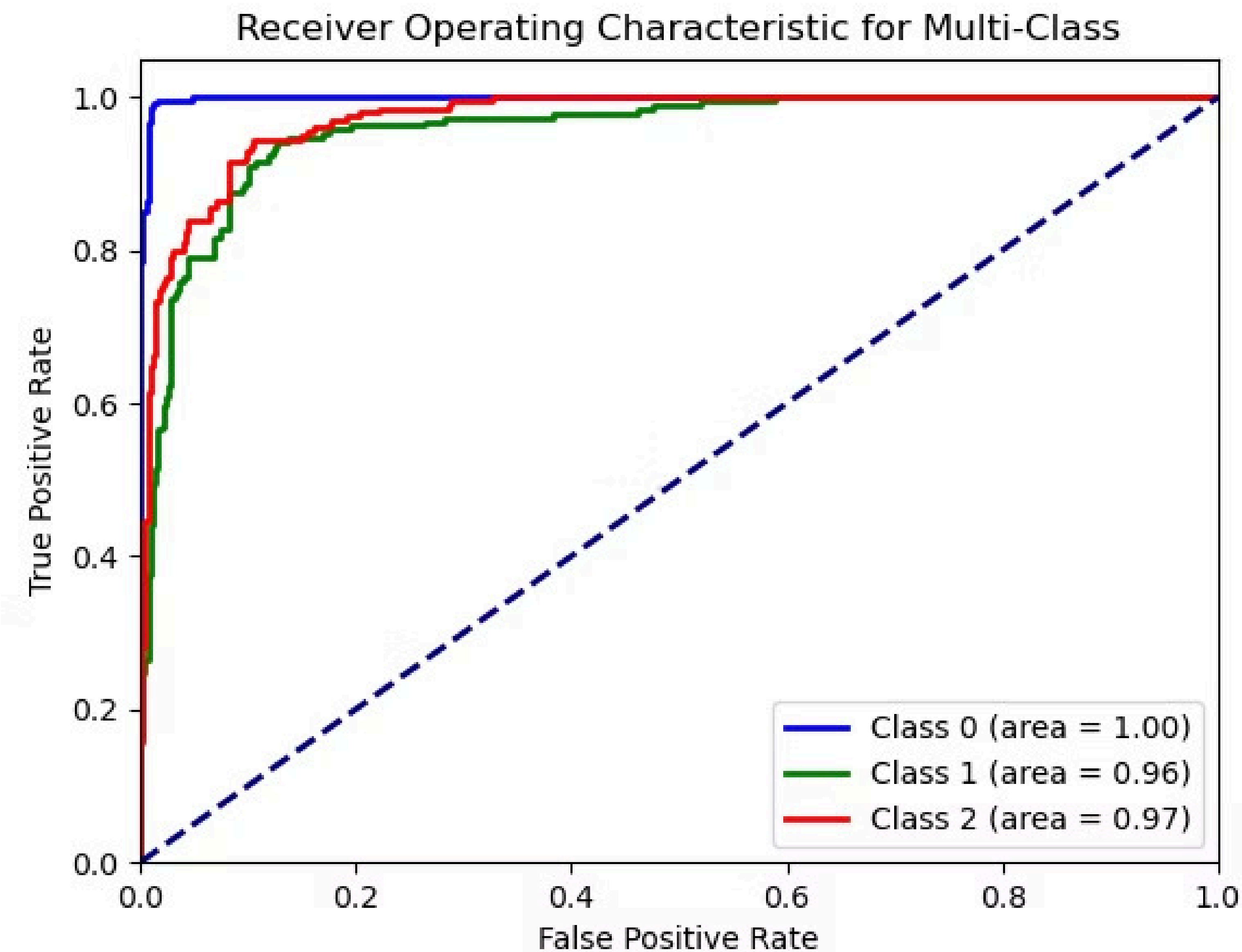
3D 소프트맥스 회기-결과 데이터



3D 소프트맥스 회기-ROC 코드

```
y_pred_prob = model.predict(x_test)
# 각 클래스에 대한 ROC 및 AUC 계산
fpr = {}
tpr = {}
roc_auc = {}

for i in range(3):  # 클래스가 3개이므로 0, 1, 2에 대해 계산
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_pred_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# ROC 곡선 그리기
plt.figure()
colors = ['blue', 'green', 'red']
for i in range(3):
    plt.plot(fpr[i], tpr[i], color=colors[i], lw=2, label=f'Class {i} (area = {roc_auc[i]:.2f})')
# 대각선 기준선 추가
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
# 그래프 설정
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Multi-Class')
plt.legend(loc="lower right")
plt.show()
```

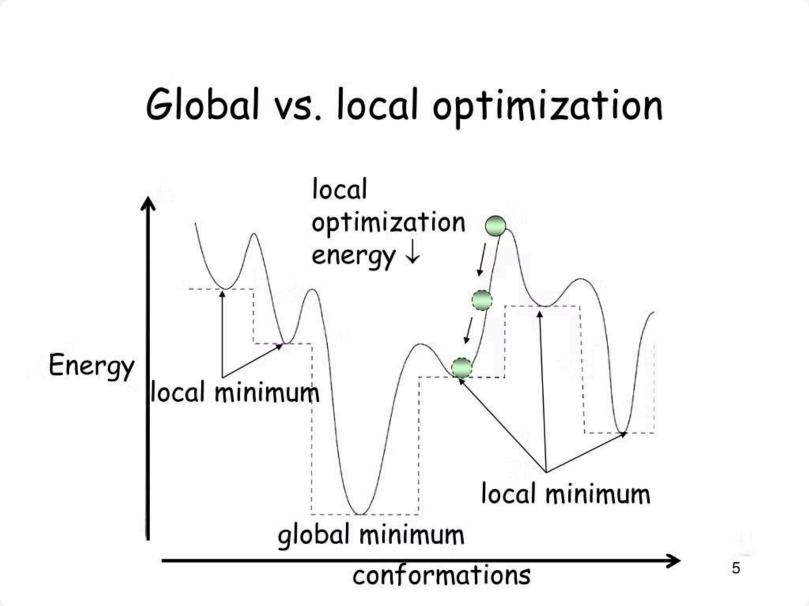
3D 소프트맥스 회기-ROC그래프



배치 학습 개념

- 배치 학습은 데이터를 한 번에 하나씩 전달하는 대신 배치 단위로 모델을 실행하고 파라미터를 업데이트하고 최적화를 실행.
- 배치 학습은 속도 면에서 이점이 있지만, 전역 최적해(global optimum) 대신 국소 최적해(local optimum)에 수렴할 위험이 있음.

1. **평균화된 경사 하강법**: 배치 학습에서는 배치 내의 여러 샘플들의 손실 함수에 대한 평균 경사를 사용해 파라미터를 업데이트하기 때문에 전체 데이터 분포에 대한 일반적인 정보를 얻을 수 있지만, 지역적으로는 최적 경로를 놓칠 수 있음.
2. **업데이트 빈도 감소**: 배치 학습은 전체 배치를 처리한 후에만 파라미터를 업데이트하기 때문에, 각 개별 데이터 포인트의 세세한 변동을 반영하는 데 한계가 있음. 특히 복잡한 비선형 문제에서는 이런 업데이트 방식이 국소 최적해에 갇힐 가능성이 큼.



작은 배치 크기의 장단점

작은 배치 크기의 장점

1. **더 다양한 경로 탐색**: 작은 배치 크기,1개의 데이터 포인트를 사용하는 SGD에서는 매번 데이터 포인트가 다르기 때문에, 경사(gradient)가 더 불안정하게 변화. 이러한 덕분에 최적화 과정에서 여러 경로를 탐색할 수 있으며, 국소 최적해(local optimum)를 피할 가능성이 높아짐.
2. **빠른 업데이트**: 작은 배치를 사용하면 파라미터 업데이트가 더 자주 일어남. 이로 인해 전역 최적해로 수렴할 수 있는 다양한 탐색 기회를 가짐.

작은 배치 크기의 단점

1. **진동 및 수렴 속도 문제**: 배치 크기가 너무 작으면 경사 하강 과정에서 진동을 하게 되며, 이는 전역 최적해에 도달하기보다는 오히려 수렴이 느려지거나 불안정함. 따라서, 작은 배치 크기만으로는 전역 최적해를 보장할 수 없음.
2. **잡음 문제**: 작은 배치 크기에서 발생하는 경사 계산의 잡음은 탐색의 다양성을 제공하지만, 이 잡음이 너무 커지면 최적화가 불안정. 이로 인해 학습률 조정이 더욱 중요해짐.