



로지스틱 회귀 (logistic regression)

# 로지스틱 회귀 (logistic regression)

## ➤ 로지스틱 회귀 (logistic regression)

- 선형 또는 바이너리 분류 문제를 위한 단순하면서도 보다 강력한 분류 알고리즘  
- 로지스틱 회귀는 선형 분리 모델에서 훌륭하게 동작하며, 실제로 가장 많이 사용되는 분류 알고리즘 중 하나입니다.

$$\text{odds ratio} = \frac{p}{1-p}$$

- $p$ 는 어떤 특정 사건이 발생할 확률입니다.
- odds ratio는 어떤 특정 사건이 일어날 확률과 그 사건이 일어나지 않을 확률의 비로 정의됩니다
- odds ratio의 로그값을 함수값으로 가지는 함수를 정의하면....

$$f(p) = \log \frac{p}{1-p}$$

- $p$ 는 0과 1사이의 수이고,  $f(p)$ 의 범위는 실수 전체가 됩니다.
- 순입력 함수의 리턴값을  $z$ 는  $w$ 의 값에 따라 실수 전체에 대해 매핑되는 값이며,  $z$ 의 값에 따라 입력된 트레이닝 데이터가  $X$ 가 어떤 집단에 속하는지 아닌지 결정하게 되는 값입니다.

$$z = \sum_j w_j x_j$$

- $f(p)$ 의 값을  $z$ 로 매핑할 수 있습니다.

$$z = \log \frac{p}{1-p}$$

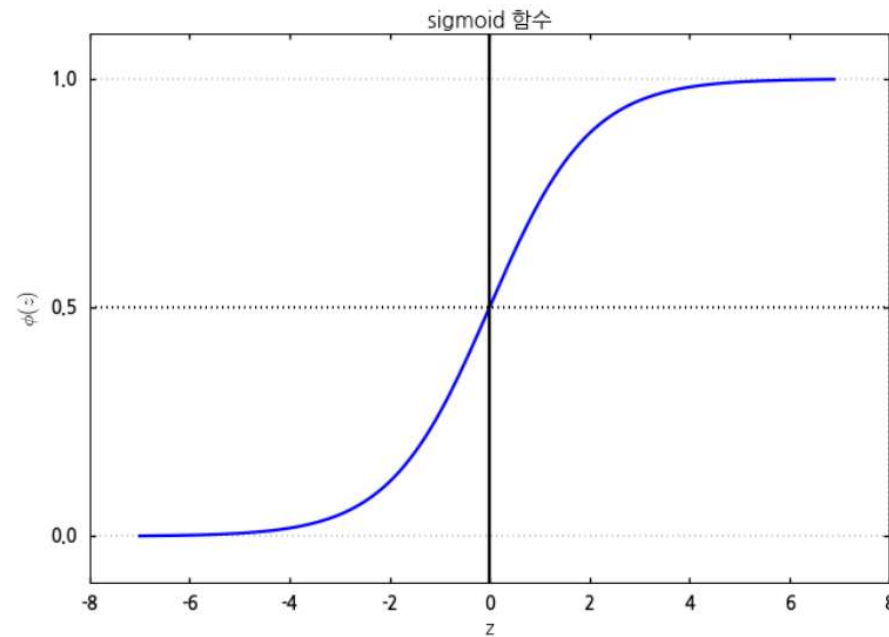
- $p$ 를  $z$ 에 관한 함수로 표현하면

$$p = \frac{1}{1 + e^{-z}} \quad \phi(z) = \frac{1}{1 + e^{-z}}$$

# 로지스틱 회귀 (logistic regression)

## ➤ 로지스틱 회귀 (logistic regression)

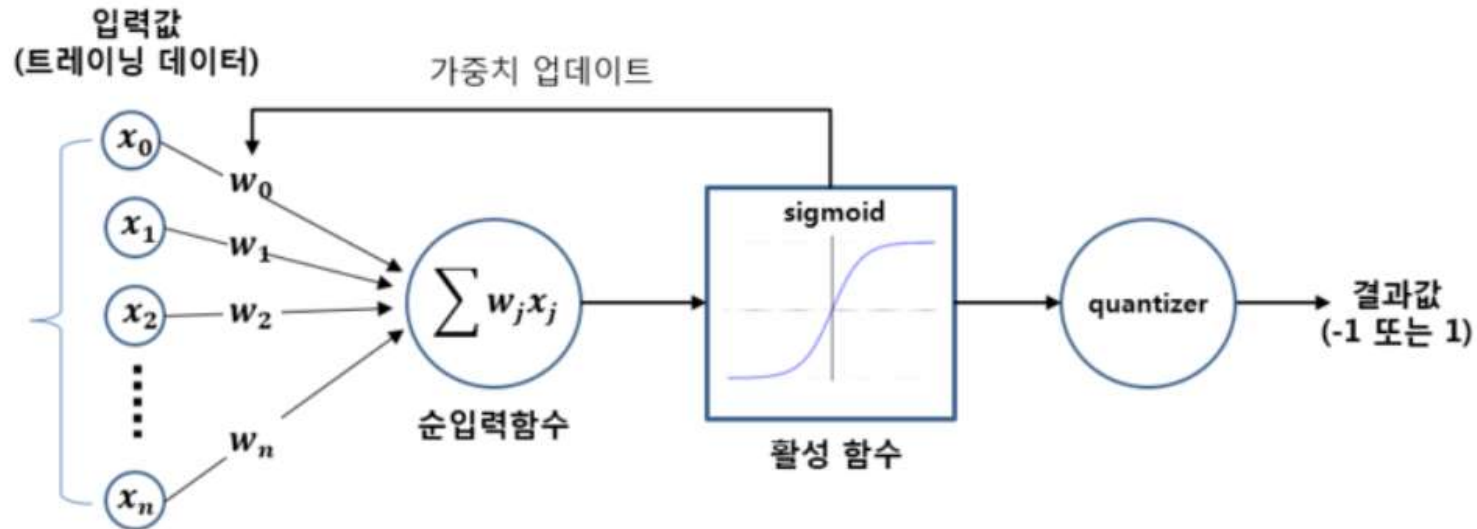
- 함수를 그래프로 그려보면 s자 형태의 곡선으로 나타납니다. (sigmoid 함수)
- 로지스틱 회귀에서는 순입력 함수의 리턴값에 대해 가중치 업데이트 여부를 결정하는 활성 함수로 sigmoid 함수를 이용합니다.



# 로지스틱 회귀 (logistic regression)

## ➤ 로지스틱 회귀 (logistic regression)

- 로지스틱 회귀는 순입력 함수의 리턴값을 sigmoid 함수에 대입하여 그 결과값을 가지고 가중치 업데이트를 할지 말지를 결정합니다.
- sigmoid 함수의 리턴값은 곧 입력한 트레이닝 데이터가 특정 클래스에 속할 확률이 되므로, 로지스틱 회귀는 입력되는 트레이닝 데이터의 특정 클래스에 포함될 예측 확률에 따라 머신러닝을 수행하는 알고리즘입니다.
- 로지스틱 회귀에서 활용되는 비용함수  $J(w)$ 와 가중치 업데이트 식은 퍼셉트론과 동일한 원리입니다.

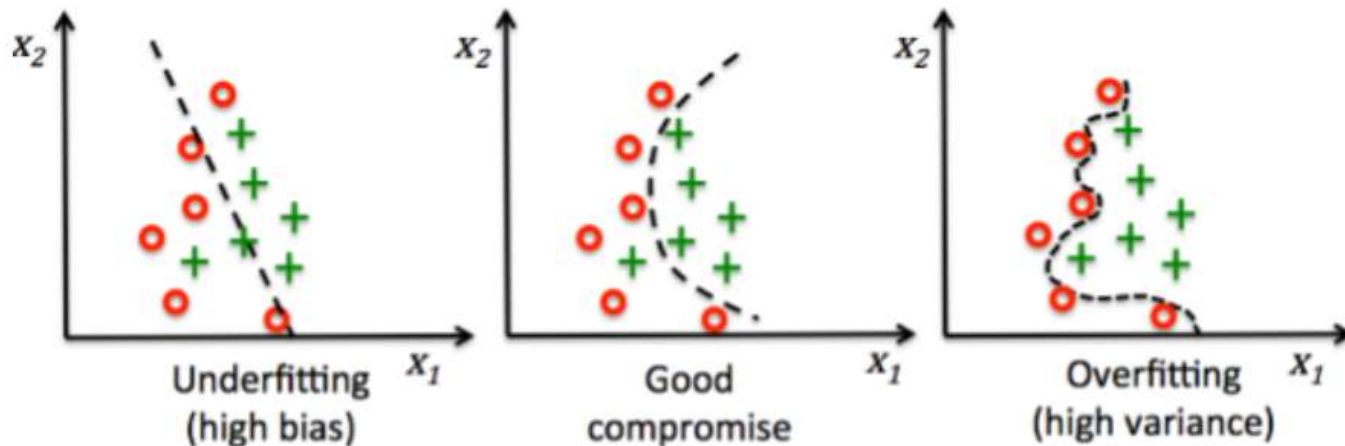


# 로지스틱 회귀 (logistic regression)

## ➤ 로지스틱 회귀 (logistic regression)



- 오버피팅(Overfitting)은 머신러닝을 위해 입력된 값에만 너무 잘 맞아 떨어지도록 계산을 해서 입력에 사용된 트레이닝 데이터 이외의 데이터들에 대해서는 잘 맞아 떨어지지 않는 경우가 발생하게 되어 미지의 데이터에 대해 그 결과를 예측하기 위한 머신러닝의 결과로는 바람직하지 않습니다.



- 최적의 조건을 찾기 위해 사용되는 것이 정규화(regularization)라는 기법인데, 정규화를 통해 데이터 모델의 복잡성을 튜닝하여 언더피팅과 오버피팅의 트레이드 오프(trade-off)를 찾아내는 것입니다.
- 로지스틱 회귀에서 사용되는 정규화 기법은 가장 일반적으로 사용되는 L2 정규화이며 사이킷런의 LogisticRegression 클래스의 매개변수  $C=1000.0$  부분은 L2 정규화와 관련된 인수입니다.
- L2 정규화는 로지스틱 회귀의 비용함수  $J(w)$ 의 변형 식입니다.

$$J(w)_{reg} = J(w) + \frac{\lambda}{2} \sum_j w_j^2$$

# 로지스틱 회귀 (logistic regression)

## ➤ 로지스틱 회귀 (logistic regression)

- $\lambda$  를 정규화 파라미터라고 부르며, C값은  $\lambda$  의 역수로 정의합니다.
- C값을 감소시키면  $\lambda$ 가 커지게 되며, 정규화를 강하게 한다는 의미입니다

$$C = \frac{1}{\lambda}$$

# 로지스틱 회귀 (logistic regression)

## ➤ 이진 분류기 훈련

- 타깃 벡터의 값이 두개
- 로지스틱 회귀에서 선형 모델은 로지스틱 함수(시그모이드 함수)에 포함됩니다.
- 로지스틱 함수는 함수의 출력을 0과 1 사이로 제한하는 효과가 있습니다.
- P가 0.5보다 크면 클래스 1로 예측하고 그렇지 않으면 클래스 0으로 예측합니다

$$P(y_i = 1 \mid X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

```
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

iris = datasets.load_iris()           # 데이터 로드
features = iris.data[:100,:]          # 두 개의 클래스만 선택
target = iris.target[:100]

scaler = StandardScaler()             # 특성을 표준화
features_standardized = scaler.fit_transform(features)

logistic_regression = LogisticRegression(random_state=0)
model = logistic_regression.fit(features_standardized, target)
new_observation = [[.5, .5, .5, .5]]
model.predict(new_observation)
model.predict_proba(new_observation)
```

# 로지스틱 회귀 모델 생성  
# 모델 훈련  
# 새로운 샘플 데이터 생성  
# 클래스 예측  
# 예측 확률 확인

# 로지스틱 회귀 (logistic regression)

## ➤ 다중 클래스 분류기 훈련

- **OVR(one-vs-rest)** 로지스틱 회귀는 클래스마다 모델을 만듭니다. (다중 분류 기법)
- OVR(one-vs-rest)는 개별 모델은 샘플이 해당 클래스에 속하는지 여부를 예측합니다(이진 분류 문제)
- 개별 분류 문제(클래스 0이거나 아니거나)는 독립적이라고 가정합니다
- 다항 로지스틱 회귀 (multinomial logistic regression) **MLR**는 로지스틱 함수를 소프트맥스 함수로 바꿉니다.
- MLR은 `predict_proba()`를 사용해 예측한 확률을 더 신뢰할 수 있습니다(보정이 잘 되어 있습니다)

$$P(y_i = k | X) = \frac{e^{\beta_k x_i}}{\sum_{j=1}^K e^{\beta_j x_i}}$$

```
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

iris = datasets.load_iris()           # 데이터 로드
features = iris.data
target = iris.target

scaler = StandardScaler()             # 특성을 표준화
features_standardized = scaler.fit_transform(features)
# OVR 로지스틱 회귀 모델을 만듭니다.
logistic_regression = LogisticRegression(random_state=0, multi_class="ovr")
model = logistic_regression.fit(features_standardized, target)  # 모델 훈련
```



# 로지스틱 회귀 (logistic regression)



## ➤ 규제로 분산 줄이기

- 규제는 복잡한 모델에 패널티를 가해 분산을 줄이는 방법입니다
- 규제는 최소화하려는 손실함수에 패널티 항을 추가합니다
- 규제 강도를 조절하는 하이퍼파라미터 C를 사용합니다.
- LogisticRegressionCV를 사용하여 효율적으로 C 값을 튜닝할 수 있습니다. (매개변수 Cs에서 탐색할 C의 범위를 입력할 수 있습니다.)

$$\alpha \sum_{j=1}^p |\hat{\beta}_j|$$

$$\alpha \sum_{j=1}^p \hat{\beta}_j^2$$

```
from sklearn.linear_model import LogisticRegressionCV
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
```

```
.
iris = datasets.load_iris()           # 데이터를 로드
features = iris.data
target = iris.target
```

```
scaler = StandardScaler()             # 특성을 표준화
features_standardized = scaler.fit_transform(features)
```

```
# 로지스틱 회귀 모델 객체 생성
```

```
logistic_regression = LogisticRegressionCV( penalty='l2', Cs=10, random_state=0, n_jobs=-1)
model = logistic_regression.fit(features_standardized, target) # 모델 훈련
logistic_regression.C_
```

# 로지스틱 회귀 (logistic regression)

## ➤ 대용량 데이터에서 분류기 훈련

- LogisticRegression에서 solver 매개변수를 확률적 평균 경사 하강법으로 지정하여 로지스틱 회귀를 학습시킵니다.
- 데이터셋이 매우 클 때 확률적 평균 경사 하강법이 다른 방법보다 훨씬 빠르게 모델을 훈련할 수 있습니다.
- 확률적 평균 경사 하강법은 특성의 스케일이 매우 민감하기 때문에 특성 표준화가 매우 중요합니다.

```
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

iris = datasets.load_iris()          # 데이터 로드
features = iris.data
target = iris.target
scaler = StandardScaler()           # 특성 표준화
features_standardized = scaler.fit_transform(features)

# 로지스틱 회귀 모델 생성
logistic_regression = LogisticRegression(random_state=0, solver="sag")
model = logistic_regression.fit(features_standardized, target) # 모델 훈련
# 에러 발생
LogisticRegression(random_state=0, solver='liblinear', penalty='none').fit(features_standardized, target)
# 에러 발생
LogisticRegression(random_state=0, solver='sag', penalty='l1').fit(features_standardized, target)
```

# 로지스틱 회귀 (logistic regression)

## ➤ 불균형한 클래스 다루기



- 매우 불균형한 클래스가 있고 전처리 과정에서 처리하지 못했다면 `class_weight` 매개변수로 클래스에 가중치를 부여하여 균형잡힌 클래스를 만들 수 있습니다
- `balanced`로 지정하면 자동으로 클래스 빈도의 역수로 가중치를 부여합니다.

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

iris = datasets.load_iris()           # 데이터 로드
features = iris.data
target = iris.target
features = features[40:,:]             # 처음 40개 샘플을 제거
target = target[40:]                 # 불균형한 클래스를 만듭니다.

# 타깃 벡터에서 0이 아닌 클래스는 모두 1로 만듭니다.
target = np.where((target == 0), 0, 1)
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)    # 특성을 표준화
# 로지스틱 회귀 모델 생성
logistic_regression = LogisticRegression(random_state=0, class_weight="balanced")
model = logistic_regression.fit(features_standardized, target) # 모델 훈련
```

# 로지스틱 회귀 (logistic regression)

## ➤ 불균형한 클래스 다루기

- `class_weight` 매개변수에 {클래스\_레이블:가중치} 형식의 딕셔너리를 전달할 수도 있습니다.
- `class_weight="balanced"`로 설정했을 때 로지스틱 모델이 계산한 클래스 가중치는 `compute_class_weight()`를 사용하여 구할 수 있습니다.

```
from sklearn.utils.class_weight import compute_class_weight

# 클래스 레이블이 0, 1인 데이터의 클래스 가중치를 계산합니다.
compute_class_weight('balanced', [0, 1], target)

# 10:1의 클래스 가중치를 부여한 로지스틱 회귀 모델을 만듭니다.
logistic_regression = LogisticRegression(random_state=0, class_weight={0:10, 1:1})
model = logistic_regression.fit(features_standardized, target) # 모델 훈련
```