로지스틱회귀-다중분류

로지스틱회귀-이진분류

$$z = w_1 x_1 + \cdots + w_n x_n + b$$

학습

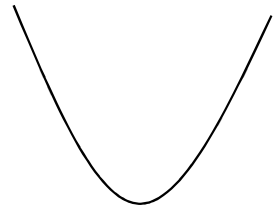
$$y' = \frac{1}{1 + e^{-z}}$$

테스트

$y' \geq 0.5, y' = 1$

$y' < 0.5, y' = 0$

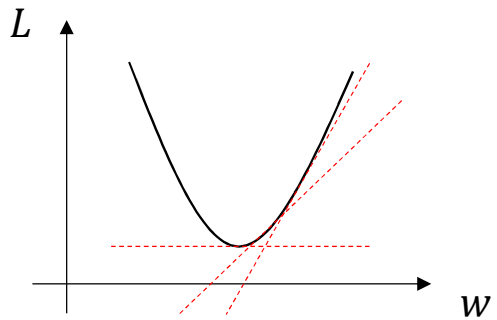
$$\mathrm{L} = -y log(y') - (1 - y) log(1 - y')$$



지수, 로그의 반대특징 이용

$$y' = h(x) = wx + b$$

$$L(y, h(x)) = \frac{1}{m} \sum (h(x) - y)^2$$

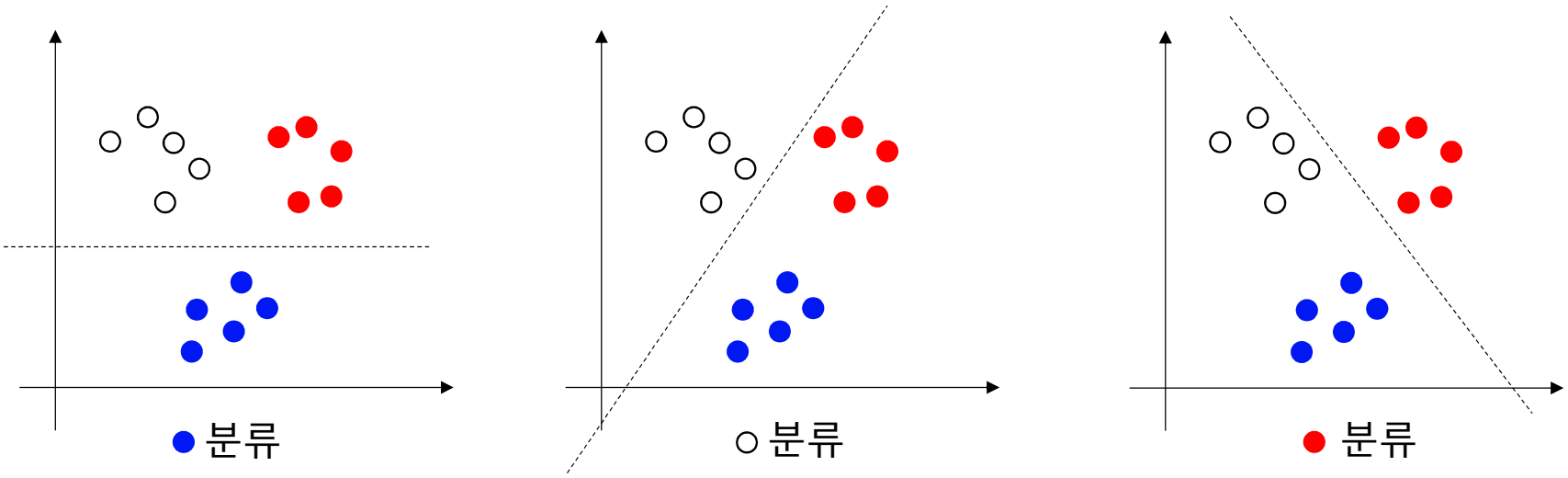


선형회귀

로지스틱회귀-다중분류

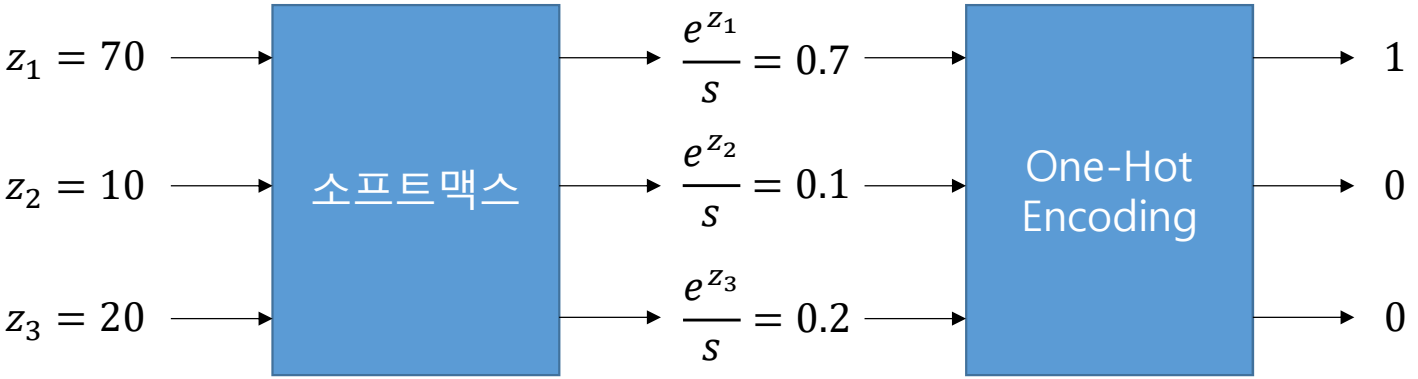
$$z_1 = w_{11}x_1 + w_{12}x_2 + b_1$$
$$z_2 = w_{21}x_1 + w_{22}x_2 + b_2$$
$$z_3 = w_{31}x_1 + w_{32}x_2 + b_3$$

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + w_{12}x_2 \\ w_{21}x_1 + w_{22}x_2 \\ w_{31}x_1 + w_{32}x_2 \end{bmatrix}$$



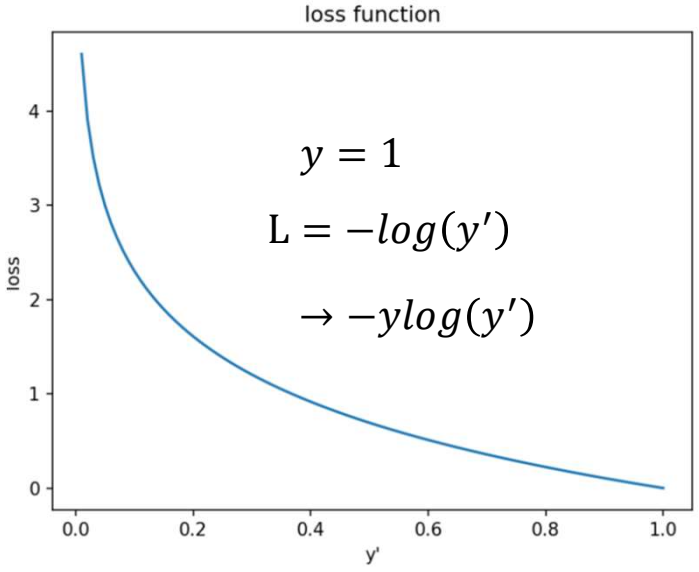
소프트맥스(Softmax) 함수

- 시그모이드 함수는 이진분류를 위해 한 개의 선형 방정식 출력값을 0과 1사이의 값으로 표현
- 소프트맥스 함수는 다중분류를 위해 복수 개의 선형 방정식 출력값들을 0과 1사이의 값으로 표현
- 소프트맥스 출력값에 One-Hot Encoding 방식을 사용하면 최대값은 1, 나머지는 0으로 변환



교차-엔트로피(Cross-Entropy) 손실함수

- 로지스틱회귀의 이진분류에서는 이진 교차 엔트로피 함수를 사용하였음
- 정답값이 1인 클래스를 제외한 모든 항은 0



$$z = w_1 x_1 + \cdots + w_n x_n + b$$

학습

$$y' = \frac{e^z}{s}$$

테스트

One-Hot Encoding

$$\mathrm{L} = -ylog(y')$$

데이터 실수화, 분할

```
import seaborn as sns
iris = sns.load_dataset("iris")

data = iris.drop("species", axis = 1)
t = iris[["species"]].copy()
t[t["species"] == "setosa"] = 0
t[t["species"] == "versicolor"] = 1
t[t["species"] == "virginica"] = 2
t = t["species"].astype("int")

from sklearn.model_selection import train_test_split
train_data, test_data, train_target, test_target = train_test_split(
    data, t, test_size = 0.3, random_state = 42, stratify = t)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

붓꽃 품종

Setosa

Versicolor

Verginica

https://en.wikipedia.org/wiki/Iris_flower_data_set

데이터 변환

- 데이터가 가진 특성 간 스케일 차이가 심하면 패턴을 찾는데 문제 발생
- 표준화 (Standardization) – 데이터가 표준정규분포의 속성을 가지도록 조정

$$x_{std} = \frac{x - mean(x)}{sd(x)}$$

- 정규화 (Normalization) – 데이터의 값을 [0, 1]로 조정

$$x_{nor} = \frac{x - min\,(x)}{\max(x) - min\,(x)}$$

sklearn.preprocessing.StandardScaler

class sklearn.preprocessing.StandardScaler(*, *copy=True*, *with_mean=True*, *with_std=True*) [source]

Standardize features by removing the mean and scaling to unit variance.

The standard score of a sample `x` is calculated as:

z = (x - u) / s

where `u` is the mean of the training samples or zero if `with_mean=False`, and `s` is the standard deviation of the training samples or one if `with_std=False`.

```
>>> from sklearn.preprocessing import StandardScaler
>>> data = [[0, 0], [0, 0], [1, 1], [1, 1]]
>>> scaler = StandardScaler()
>>> print(scaler.fit(data))
StandardScaler()
>>> print(scaler.mean_)
[0.5 0.5]
>>> print(scaler.transform(data))
[[-1. -1.]
 [-1. -1.]
 [ 1.  1.]
 [ 1.  1.]]
>>> print(scaler.transform([[2, 2]]))
[[3. 3.]]
```

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler

sklearn.preprocessing.MinMaxScaler

class sklearn.preprocessing.MinMaxScaler(*feature_range=(0, 1)*, *, *copy=True*, *clip=False*) [source]

Transform features by scaling each feature to a given range.

This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

The transformation is given by:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

where min, max = feature_range.

```
>>> from sklearn.preprocessing import MinMaxScaler
>>> data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
>>> scaler = MinMaxScaler()
>>> print(scaler.fit(data))
MinMaxScaler()
>>> print(scaler.data_max_)
[ 1. 18.]
>>> print(scaler.transform(data))
[[0.   0.  ]
 [0.25 0.25]
 [0.5  0.5 ]
 [1.   1.  ]]
>>> print(scaler.transform([[2, 2]]))
[[1.5 0. ]]
```

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler

로지스틱회귀 정확도

```
from sklearn.preprocessing import StandardScaler
```

구현

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
# from sklearn.neighbors import KNeighborsClassifier
# model = KNeighborsClassifier()
model.fit(train_data_scaled, train_target)
print("Train-Eval:", model.score(train_data_scaled, train_target))
print("Test-Eval :", model.score(test_data_scaled, test_target))
```

	로지스틱회귀	이웃회귀
표준화	Train-Eval: 0.9809523809523809 Test-Eval : 0.9111111111111111	Train-Eval: 0.9809523809523809 Test-Eval : 0.9111111111111111
정규화	Train-Eval: 0.9523809523809523 Test-Eval : 0.8666666666666667	Train-Eval: 0.9809523809523809 Test-Eval : 0.9333333333333333

참고자료

- 지능기전공학부 최유경 교수님 자료, https://github.com/sejongresearch/2021.MachineLearning
- 코랩(Colab), https://colab.research.google.com/
- 파이썬(Python), https://www.python.org/doc/
- 사이킷런(sckit-learn), https://scikit-learn.org/stable/index.html
- 판다스(pandas), https://pandas.pydata.org/
- 맷플롯립(matplotlib), https://matplotlib.org/
- 씨본(seaborn), https://seaborn.pydata.org/
- 캐글(Kaggle), https://www.kaggle.com/
- 넘파이(numpy), https://numpy.org/doc/stable/
- 스택오퍼플러우(stackoverflow), https://stackoverflow.com/