


- scikit-learn은 2007년 구글 썸머 코드에서 처음 구현됐으며 현재 파이썬으로 구현된 가장 유명한 기계 학습 오픈 소스 라이브러리다. scikit-learn의 장점은 라이브러리 외적으로는 scikit 스택을 사용하고 있기 때문에 다른 라이브러리와의 호환성이 좋다. 내적으로는 통일된 인터페이스를 가지고 있기 때문에 매우 간단하게 여러 기법을 적용할 수 있어 쉽고 빠르게 최상의 결과를 얻을 수 있다.
- 라이브러리의 구성은 크게 지도 학습, 비지도 학습, 모델 선택 및 평가, 데이터 변환으로 나눌 수 있다(scikit-learn 사용자 가이드 참조, http://scikit-learn.org/stable/user_guide.html). 지도 학습에는 서포트 벡터 머신, 나이브 베이즈(Naïve Bayes), 결정 트리(Decision Tree) 등이 있으며 비지도 학습에는 군집화, 이상치 검출 등이 있다. 모델 선택 및 평가에는 교차 검증(cross-validation), 파이프라인(pipeline)등 있으며 마지막으로 데이터 변환에는 속성 추출(Feature Extraction), 전처리(Preprocessing)등이 있다.

<https://scikit-learn.org/stable/>

[Install](#) [User Guide](#) [API](#) [Examples](#) [More](#)

scikit-learn

Machine Learning in Python

[Getting Started](#) [Release Highlights for 0.23](#) [GitHub](#)

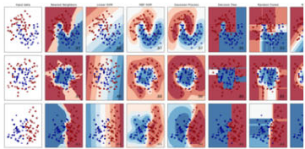
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: SVM, nearest neighbors, random forest, and more...



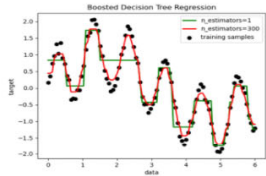
Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, nearest neighbors, random forest, and more...




Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, and more...



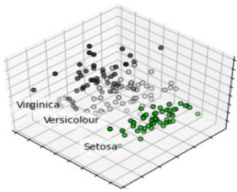
Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...



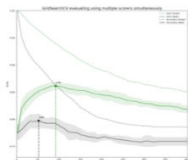
Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning

Algorithms: grid search, cross validation, metrics, and more...



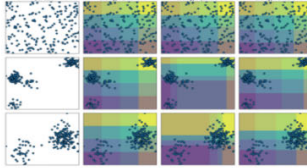
Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.

Algorithms: preprocessing, feature extraction, and more...



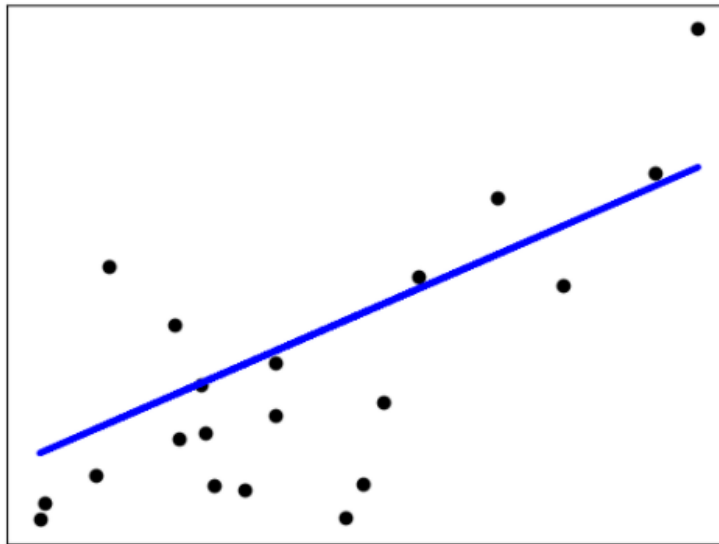
Examples

머신러닝의 주요 지도학습 알고리즘

- k-최근접 이웃 k-nearest neighbors
- 선형 회귀 linear regression
- 로지스틱 회귀 logistic regression
- 서포트 벡터 머신 support vector machine (SVM)
- 결정 트리 decision tree 와 랜덤 포레스트 random forest
- 신경망 neural networks⁶

Linear Regression

Linear Regression은 단일 변수나 다변수 x 에 대해 w_0, w_1, \dots, w_p 값을 찾아서 \hat{y} 를 정확하게 예측하는 문제이다.



$$\hat{y}(w, x) = w_0 + w_1x_1 + \dots + w_px_p$$

$w = (w_1, \dots, w_p)$ as `coef_` and w_0

Linear Regression

- X와 Y데이터를 생성 후,
- 모델 선언 / 학습 / 예측의 순서로 진행한다.

```
# 모델 선언
```

```
model = LinearRegression()
```

```
# 학습
```

```
model.fit(x, y)
```

```
# 예측
```

```
prediction = model.predict([[10.0]])
```

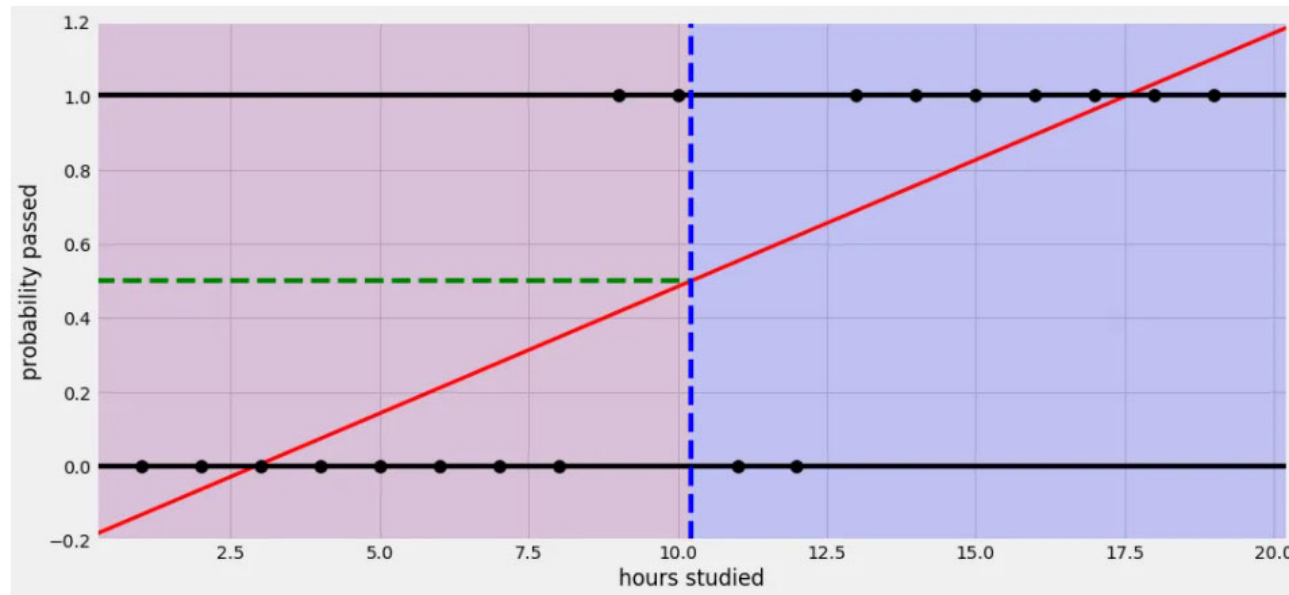
ex1_sklearn_start.ipynb 파일 참조

Logistic Regression

- 로지스틱 회귀는 회귀를 사용하여 데이터가 어떤 범주에 속할 확률을 0에서 1 사이 (Logistic Regression)의 값으로 예측하고 그 확률에 따라 가능성이 더 높은 범주에 속하는 것으로 분류해주는 **지도 학습** 알고리즘이다.
- 스팸 메일 분류기 같은 예시를 생각하면 쉽다. 어떤 메일을 받았을 때 그것이 스팸일 확률이 0.5 이상이면 spam으로 분류하고, 확률이 0.5보다 작은 경우 ham으로 분류하는 거다. 이렇게 데이터가 2개의 범주 중 하나에 속하도록 결정하는 것을 **2진 분류(binary classification)**라고 한다.

Logistic Regression

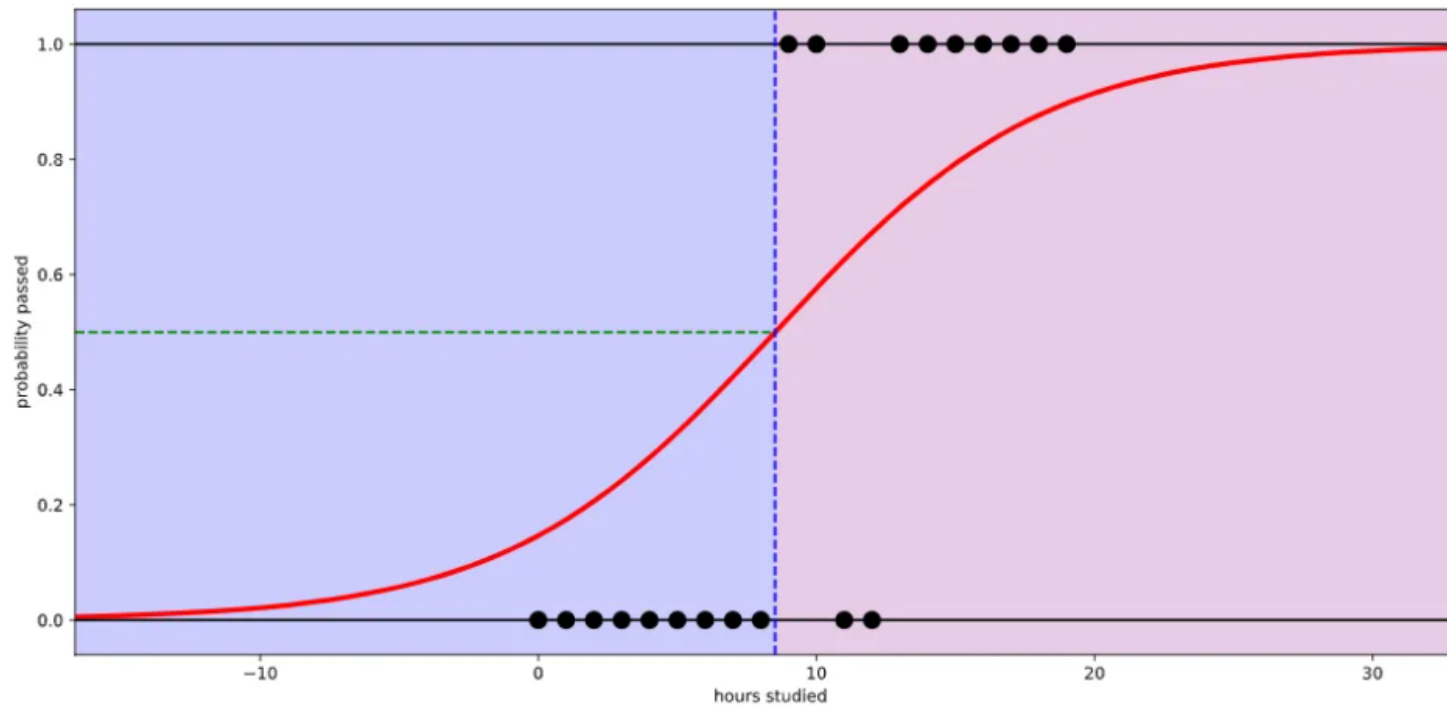
- 예를 들어 어떤 학생이 공부하는 시간에 따라 시험에 합격할 확률이 달라진다고 해보자. 선형 회귀를 사용하면 아래와 같은 그림으로 나타낼 수 있다.



- 공부한 시간이 적으면 시험에 통과 못하고, 공부한 시간이 많으면 시험에 통과한다는 식으로 설명할 수 있다. 그런데 이 회귀선을 자세히 살펴보면 **확률이 음과 양의 방향으로 무한대까지 뻗어 간다**. 말 그대로 '선'이라서. 그래서 공부를 2시간도 안 하면 시험에 통과할 확률이 0이 안 된다. 이건 말이 안 된다.

Logistic Regression

- 로지스틱 회귀를 사용하면 아래와 같이 나타난다.
- 시험에 합격할 확률이 0과 1사이의 값으로 그려진다.



Logistic Regression

로지스틱 모형 식은 독립 변수가 $[-\infty, \infty]$ 의 어느 숫자이든 상관 없이 종속 변수 또는 결과 값이 항상 범위 $[0, 1]$ 사이에 있도록 한다. 이는 오즈(odds)를 로짓(logit) 변환을 수행함으로써 얻어진다.

1. odds

성공 확률이 실패 확률에 비해 몇 배 더 높은가를 나타내며 그 식은 아래와 같다.

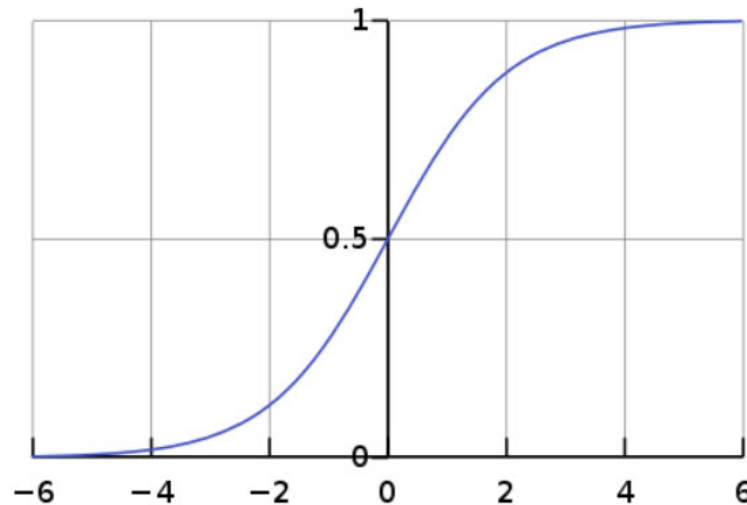
$$\text{odds} = \frac{p(y = 1|x)}{1 - p(y = 1|x)}$$

2. 로짓변환

오즈에 로그를 취한 함수로서 입력 값의 범위가 $[0, 1]$ 일때 출력 값의 범위를 $[-\infty, \infty]$ 로 조정한다.

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

Logistic Regression



- 로지스틱 함수 (logistic function)

로지스틱 함수의 그래프는 Figure 1과 같고 이는 독립 변수 x 가 주어졌을 때 종속 변수가 1의 범주에 속할 확률을 의미한다. 즉, $p(y = 1|x)$ 를 의미한다.

로지스틱 함수는 로짓 변환을 통해 만들어지고, 그 형태는 다음과 같다.

$$\text{logistic function} = \frac{e^{\beta \cdot X_i}}{1 + e^{\beta \cdot X_i}}$$

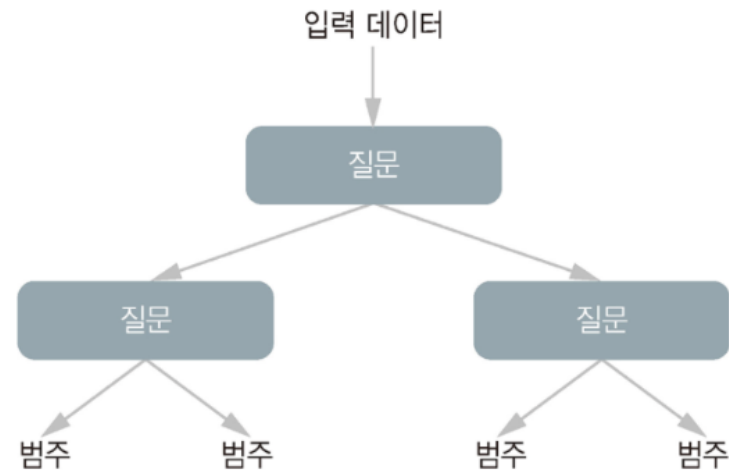
Logistic Regression

참조 사이트

- https://ko.wikipedia.org/wiki/%EB%A1%9C%EC%A7%80%EC%8A%A4%ED%8B%B1_%ED%9A%8C%EA%B7%80
- <http://hleecaster.com/ml-logistic-regression-concept/>

결정 트리 (Decision tree)

- 플로차트와 같은 구조를 가지며 입력 데이터 포인트를 분류하거나 주어진 입력에 대해 출력 값을 예측합니다. 결정 트리는 시각화하고 이해하기 쉽습니다.

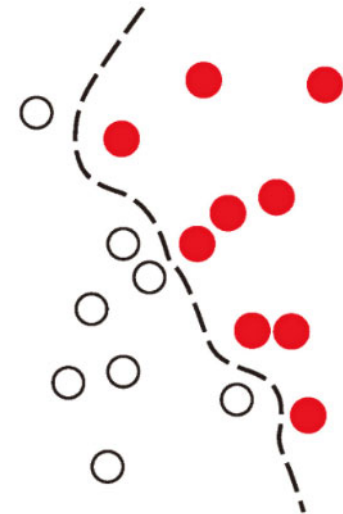


학습된 파라미터는 데이터에 관한 질문으로,
"데이터에 있는 두번째 특성이 3.5보다 큰가?"
같은 질문이 될 수 있다.

SVM

현대적인 SVM의 공식은 1995년에 벨 연구소에서 공개되었습니다. SVM은 분류 문제를 해결하기 위해 2개의 다른 범주에 속한 데이터 포인트 그룹 사이에 좋은 결정 경계를 찾습니다. 결정 경계는 훈련 데이터를 2개의 범주에 대응하는 영역으로 나누는 직선이나 표면으로 생각할 수 있습니다. 새로운 데이터 포인트를 분류하려면 결정 경계 어느 쪽에 속하는지를 확인하면 됩니다.

결정 경계

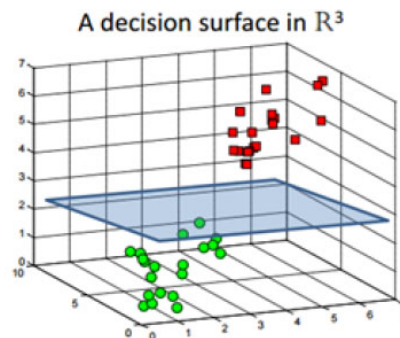
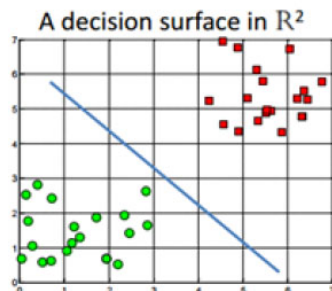


SVM이 결정 경계를 찾는 과정은 두 단계입니다.

1. 결정 경계가 하나의 초평면(hyperplane)으로 표현될 수 있는 새로운 고차원 표현으로 데이터를 매핑합니다(그림 1-10과 같은 2차원 데이터라면 초평면은 직선이 됩니다).
2. 초평면과 각 클래스의 가장 가까운 데이터 포인트 사이의 거리가 최대가 되는 최선의 결정 경계(하나의 분할 초평면)를 찾습니다. 이 단계를 **마진 최대화**(maximizing the margin)라고 부릅니다. 이렇게 함으로써 결정 경계가 훈련 데이터셋 이외의 새로운 샘플에 잘 일반화되도록 도와줍니다.

초평면(Hyperplane)이란?

SVM은 보통 범주 값을 유사한 데이터들로 그룹짓기 위해 초평면(hyperplane)이라는 경계를 사용하는데 초평면은 $n-1$ 차원의 subspace를 의미하는 것이며, 3차원의 경우 초평면은 2차원의 면이 되고, 2차원의 경우 초평면은 1차원의 선이 된다.



랜덤 포레스트

의사결정트리 학습법은 좋은 방법이지만, 주어진 데이터에 따라 생성되는 의사결정트리가 매우 달라져서 일반화하여 사용하기 어렵고, 의사결정트리를 이용한 학습 결과 역시 성능과 변동의 폭이 크다는 단점이 있습니다.

이에 단점을 보완한 방법이 랜덤 포레스트 입니다.

1. 주어진 트레이닝 데이터에서 무작위로 중복을 허용해서 n 개를 선택합니다.
2. 선택한 n 개의 데이터 샘플에서 데이터 특성값을 중복 허용없이 d 개 선택합니다.
3. 1~2단계를 k 번 반복합니다.
4. 1~3단계를 통해 생성된 k 개의 의사결정트리를 이용해 예측하고, 예측된 결과의 평균이나 가장 많이 등장한 예측 결과를 선택하여 최종 예측값으로 결정합니다.

1단계에서 무작위로 중복을 허용해서 선택한 n 개의 데이터를 선택하는 과정을 부트스트랩이라고 부르며, 부트스트랩으로 추출된 n 개의 데이터를 부트스트랩 샘플이라 부릅니다. scikit-learn이 제공하는 랜덤 포레스트 API는 부트스트랩 샘플의 크기 n 의 값으로 원래 트레이닝 데이터 전체 개수와 동일한 수를 할당합니다.

랜덤 포레스트

4단계에서 여러 개의 의사 결정트리로부터 나온 예측 결과들의 평균이나 다수의 예측 결과를 이용하여 방법을 앙상블기법이라고 합니다. 다수의 예측 결과를 선택하는 것은 다수 결의 원칙과 비슷합니다.

부트스트랩을 이용해 무작위 의사 결정트리의 집합인 랜덤 포레스트를 구성하는 것처럼 부트스트랩으로 다양한 분류기에 대해 앙상블 기법을 활용하여 특징적인 하나의 분류기로 구성하는 방법을 배깅(Bagging)이라고 합니다.

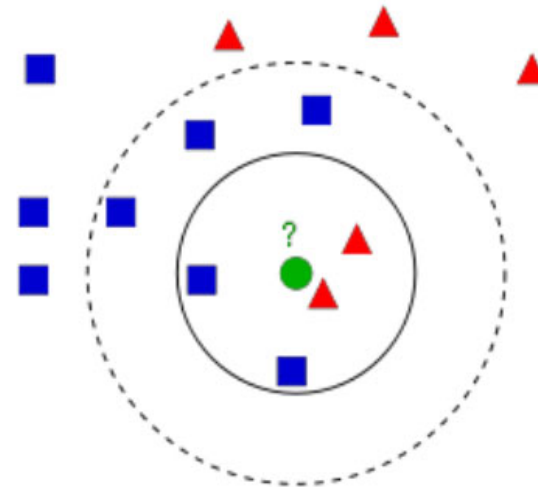
KNN(K-Nearest Neighbors)

지도학습에 활용되는 가장 단순한 종류의 알고리즘입니다.

KNN은 트레이닝 데이터로부터 예측을 위한 함수를 학습하는 것이 아니라 트레이닝 데이터 자체를 기억하는 것이 핵심입니다.

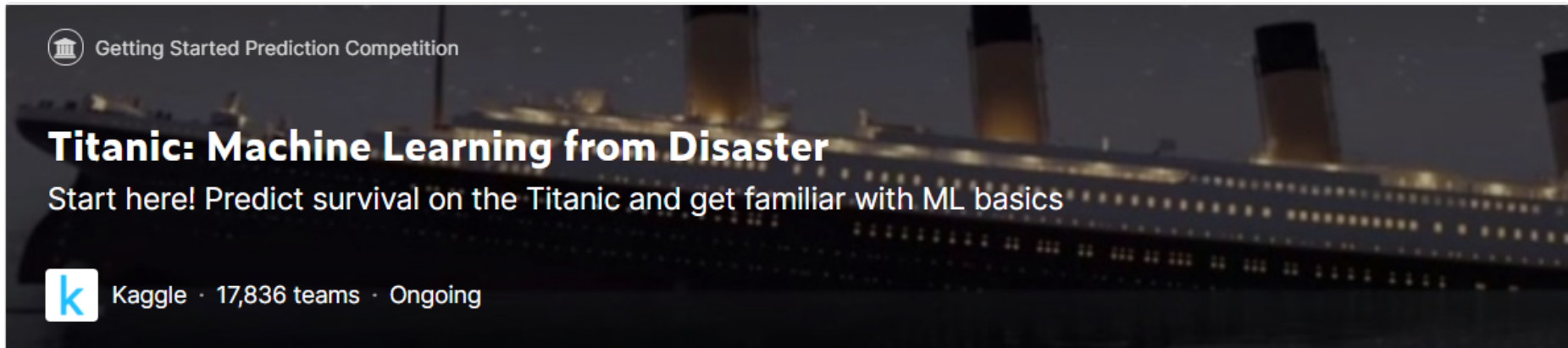
새로운 데이터와 가장 가깝게 위치하는 데이터가 속해 있는 그룹으로 분류합니다.

그러나 이는 오류를 발생할 수 있는 여지가 있습니다. 따라서 통계적인 분포를 보고 판단하게 되면 더 정확한 판단을 할 수 있습니다. 주변에 있는 데이터를 몇개를 보고 판단하느냐가 k 입니다. 작은 원의 경우 $k=4$ 인 경우이고, 점선의 경우 $k=7$ 입니다.



Titanic: Machine Learning from Disaster


- <https://www.kaggle.com/c/titanic/overview>

The banner features a night-time photograph of the Titanic ship, illuminated by its own lights and the lights of the city in the background. The ship is viewed from a low angle, emphasizing its scale.

Getting Started Prediction Competition

Titanic: Machine Learning from Disaster

Start here! Predict survival on the Titanic and get familiar with ML basics

 Kaggle · 17,836 teams · Ongoing

Overview Data Notebooks Discussion Leaderboard Rules

[Join Competition](#)

Data Description

Overview

The data has been split into two groups:

- training set (train.csv)
- test set (test.csv)

Titanic: 데이터 확인하기

```
import numpy as np
import pandas as pd
train = pd.read_csv('https://bit.ly/fc-ml-titanic')
train.head()
# 전체 데이터중 학습에 사용할 데이터를 선택하자.
feature = ['Pclass', 'Sex', 'Age', 'Fare']

# 예측 해야 할 값
label = ['Survived']

# 데이터 확인하기
train[feature].head()
train[label].head()
```

Titanic : train_test_split / 결측치 채우기

```
from sklearn.model_selection import train_test_split
```

```
# 전체 데이터를 train data와 validation data의 비율을 8:2로 나눈다.
```

```
x_train, x_valid, y_train, y_valid = train_test_split(train[feature], train[label], test_size=0.2, shuffle=True, random_state=30)
```

```
# 전체 데이터수 891개와 비교해서 갯수가 부족한 항목을 확인하자
```

```
train.info()
```

```
train.isnull().sum()
```

```
train['Age'].isnull().sum()
```

```
# 0으로 채웠을때의 통계값을 출력해보자
```

```
train['Age'].fillna(0).describe()
```

```
# 평균 값으로 채웠을 때의 통계값을 출력해보자
```

```
train['Age'].fillna(train['Age'].mean()).describe()
```

데이터셋 구성하기

Training		Test
Training	Validation	Test

2가지로 나누었을 때
3가지로 나누었을 때

Training set	학습에 사용하는 데이터 셋
Validation set	학습에 사용하지 않고 파라미터 튜닝에 사용하는 데이터 셋
Test	학습이 완전히 끝난 후에 모델을 평가하기 위한 데이터 셋

Titanic : SimpleImputer

```
from sklearn.impute import SimpleImputer
```

```
imputer = SimpleImputer(strategy='mean')
```

fit() 을 통해 결측치에 대한 학습을 진행합니다.

```
imputer.fit(train[['Age', 'Pclass']])
```

transform() 은 실제 결측치에 대한 처리를 해주는 함수 입니다.

```
result = imputer.transform(train[['Age', 'Pclass']])
```

Titanic : LabelEncoder

```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
```

```
train['Sex_num'] = le.fit_transform(train['Sex'])
```

```
train['Sex_num'].value_counts()
```

```
le.classes_
```

```
le.inverse_transform([0, 1, 1, 0, 0, 1, 1])
```

```
le.fit_transform(train['Embarked'])
```

Titanic : OneHot Encoding -> get_dummies

```
[ ] train['Embarked'][:6]
```

```
0    S
1    C
2    S
3    S
4    S
5    Q
Name: Embarked, dtype: object
```

```
[ ] train['Embarked_num'][:6]
```

```
0    2
1    0
2    2
3    2
4    2
5    1
Name: Embarked_num, dtype: int64
```

```
pd.get_dummies(train['Embarked_num'][:6])
```

	0	1	2
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
5	0	1	0

Titanic : Normalize

네이버 영화평점 (0점 ~ 10점): [2, 4, 6, 8, 10]

넷플릭스 영화평점 (0점 ~ 5점): [1, 2, 3, 4, 5]

```
movie = {'naver': [2, 4, 6, 8, 10],  
         'netflix': [1, 2, 3, 4, 5]  
        }
```

```
movie = pd.DataFrame(data=movie)  
movie
```

	naver	netflix
0	2	1
1	4	2
2	6	3
3	8	4
4	10	5

Titanic : Normalize

```
from sklearn.preprocessing import MinMaxScaler
```

```
min_max_scaler = MinMaxScaler()
```

```
min_max_movie = min_max_scaler.fit_transform(movie)
```

```
pd.DataFrame(min_max_movie, columns=['naver', 'netflix'])
```

	naver	netflix
0	0.00	0.00
1	0.25	0.25
2	0.50	0.50
3	0.75	0.75
4	1.00	1.00

Titanic : StandardScaler()

전체 데이터의 평균이 0과 표준편차가 1이 되도록 변환

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

```
standard_scaler = StandardScaler()
```

```
x = np.arange(10)
```

```
# outlier 추가
```

```
x[9] = 1000
```

```
x.mean(), x.std()
```

```
scaled.mean(), scaled.std()
```

```
round(scaled.mean(), 2), scaled.std()
```

sklearn.datasets: Datasets

Scikit-learn에서 사용할 수 있는 데이터셋

- <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets>

sklearn.datasets: Datasets

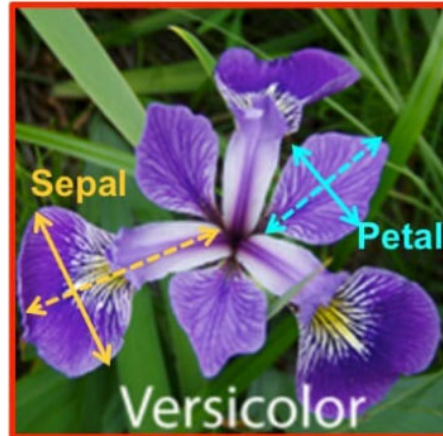
The `sklearn.datasets` module includes utilities to load datasets, including methods to load and fetch popular reference datasets. It also features some artificial data generators.

User guide: See the [Dataset loading utilities](#) section for further details.

Loaders

<code>datasets.clear_data_home([data_home])</code>	Delete all the content of the data home cache.
<code>datasets.dump_svmlight_file(X, y, *[, ...])</code>	Dump the dataset in svmlight / libsvm file format.
<code>datasets.fetch_20newsgroups(*[, data_home, ...])</code>	Load the filenames and data from the 20 newsgroups dataset (classification).
<code>datasets.fetch_20newsgroups_vectorized(*[, ...])</code>	Load the 20 newsgroups dataset and vectorize it into token counts (classification).
<code>datasets.fetch_california_housing(*[, ...])</code>	Load the California housing dataset (regression).
<code>datasets.fetch_covtype(*[, data_home, ...])</code>	Load the covtype dataset (classification).
<code>datasets.fetch_kddcup99(*[, subset, ...])</code>	Load the kddcup99 dataset (classification).
<code>datasets.fetch_lfw_pairs(*[, subset, ...])</code>	Load the Labeled Faces in the Wild (LFW) pairs dataset (classification).
<code>datasets.fetch_lfw_people(*[, data_home, ...])</code>	Load the Labeled Faces in the Wild (LFW) people dataset (classification).
<code>datasets.fetch_olivetti_faces(*[, ...])</code>	Load the Olivetti faces data-set from AT&T (classification).
<code>datasets.fetch_openml([name, version, ...])</code>	Fetch dataset from openml by name or dataset id.
<code>datasets.fetch_rcv1(*[, data_home, subset, ...])</code>	Load the RCV1 multilabel dataset (classification).
<code>datasets.fetch_species_distributions(*[, ...])</code>	Loader for species distribution dataset from Phillips et.
<code>datasets.get_data_home([data_home])</code>	Return the path of the scikit-learn data dir.
<code>datasets.load_boston(*[, return_X_y])</code>	Load and return the boston house-prices dataset (regression).
<code>datasets.load_breast_cancer(*[, return_X_y, ...])</code>	Load and return the breast cancer wisconsin dataset (classification).
<code>datasets.load_diabetes(*[, return_X_y, as_frame])</code>	Load and return the diabetes dataset (regression).
<code>datasets.load_digits(*[, n_class, ...])</code>	Load and return the digits dataset (classification).

sklearn.datasets: iris 데이터셋



<https://www.snaplogic.com/machine-learning-showcase/iris-flower-classification>

sklearn.datasets.load_iris

```
sklearn.datasets.load_iris(*, return_X_y=False, as_frame=False)
```

[\[source\]](#)

Load and return the iris dataset (classification).

The iris dataset is a classic and very easy multi-class classification dataset.

Classes	3
Samples per class	50
Samples total	150
Dimensionality	4
Features	real, positive

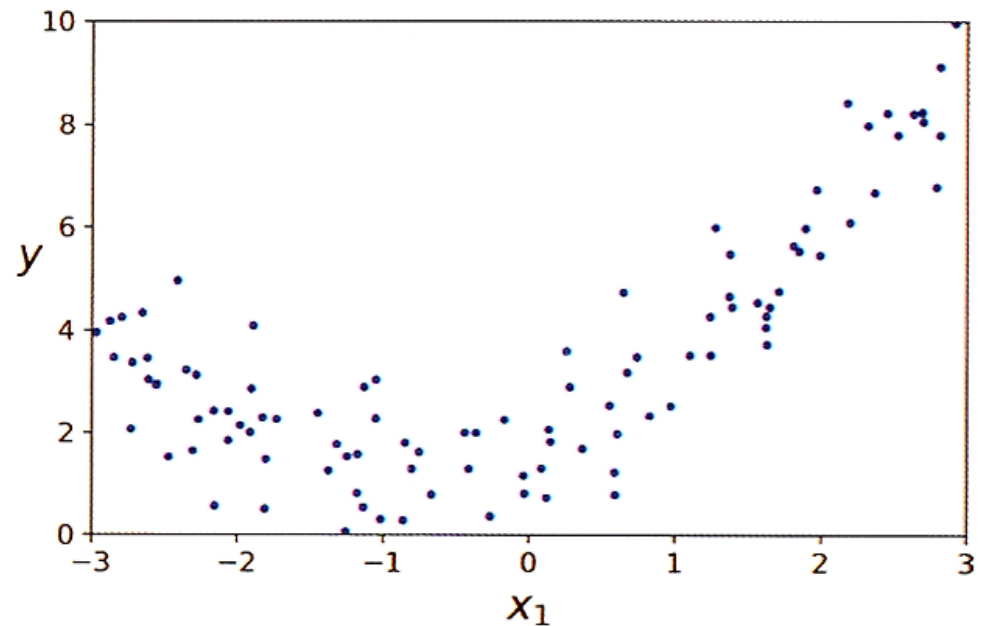
ex3_sklearn_classification.ipynb

보스턴 주택 가격 - 다항 회귀

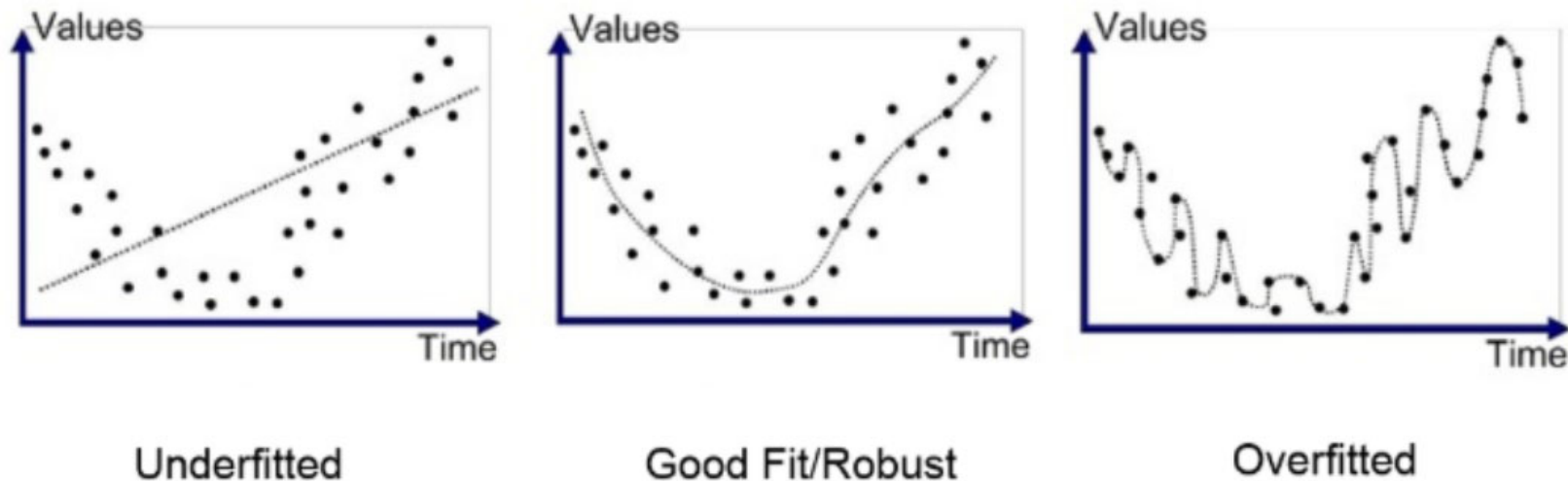
가지고 있는 데이터가 단순한 직선보다 복잡한 형태라면?

이를 위해서 비선형 데이터를 학습하는데 선형 모델을 사용할 수 있습니다. 이렇게 하는 간단한 방법은 각 특성의 거듭제곱을 새로운 특성으로 추가하고, 이 확장된 특성을 포함한 데이터셋에 선형 모델을 훈련시키는 것입니다. 이런 기법을 다항 회귀라고 합니다.

```
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```



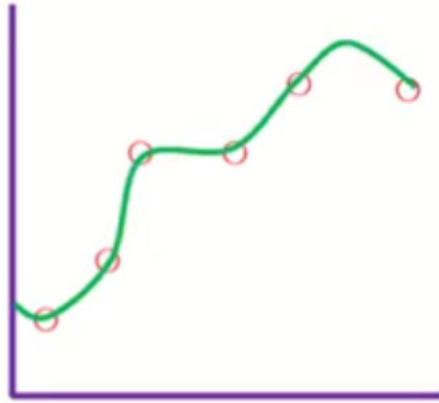
보스턴 주택 가격 – overfitting 해결하기



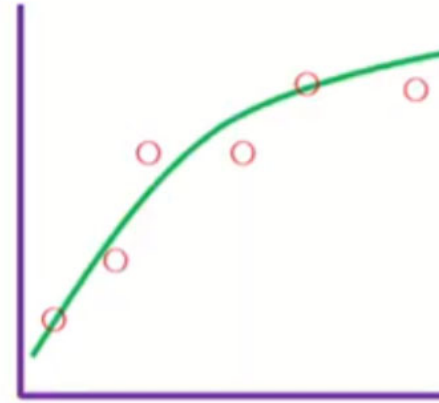
overfitting을 해결하는 방법은 크게 두 가지로 볼 수 있습니다.

1. **특성(Feature)의 갯수를 줄이기**
주요 특징을 직접 선택하고 나머지는 버린다.
Model selection algorithm을 사용한다.
2. **정규화(Regularization)를 수행한다.**
모든 특성을 사용하되, 파라미터(세타)의 갯수를 줄인다.

보스턴 주택 가격 – overfitting 규제하기



$$\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 x^4$$



$$\beta_0 + \beta_1 x + \beta_2 x^2$$

전체식이 최소가 되려면 β_3 와 β_4 가 0일 될 때입니다.

규제가 있는 선형모델

현재 데이터에 대한 예측력도 중요하지만 미래에 예측할 데이터도 중요하기 때문에 일반화가 중요합니다

$$\beta_1, \beta_2, \dots, \beta_p$$

$$L(\beta) = \min_{\beta} \underbrace{\sum_{i=1} (y_i - \hat{y}_i)^2}_{(1) \text{ Training accuracy}} + \underbrace{\lambda \sum_{j=1}^p \beta_j^2}_{(2) \text{ Generalization accuracy}}$$

Training accuracy만 있으면 최소 제곱법과 다른 것이 없는데 Generalization accuracy가 추가되면서 베타에 제약을 줄 수 있어 정규화가 가능해지게 됩니다.

규제가 있는 선형 모델 - 릿지 회귀

릿지 회귀는 규제가 추가된 선형 회귀 버전입니다. 이는 학습 알고리즘을 데이터에 맞추는 것뿐만 아니라 모델의 가중치가 가능한 작게 유지되도록 합니다. 규제항은 훈련하는 동안에만 비용 함수에 추가됩니다. 모델의 훈련이 끝나면 모델의 성능을 규제가 없는 성능 지표로 평가합니다.

아래 식은 릿지 회귀의 비용함수입니다.

$$J(\theta) = \text{MSE}(\theta) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2$$

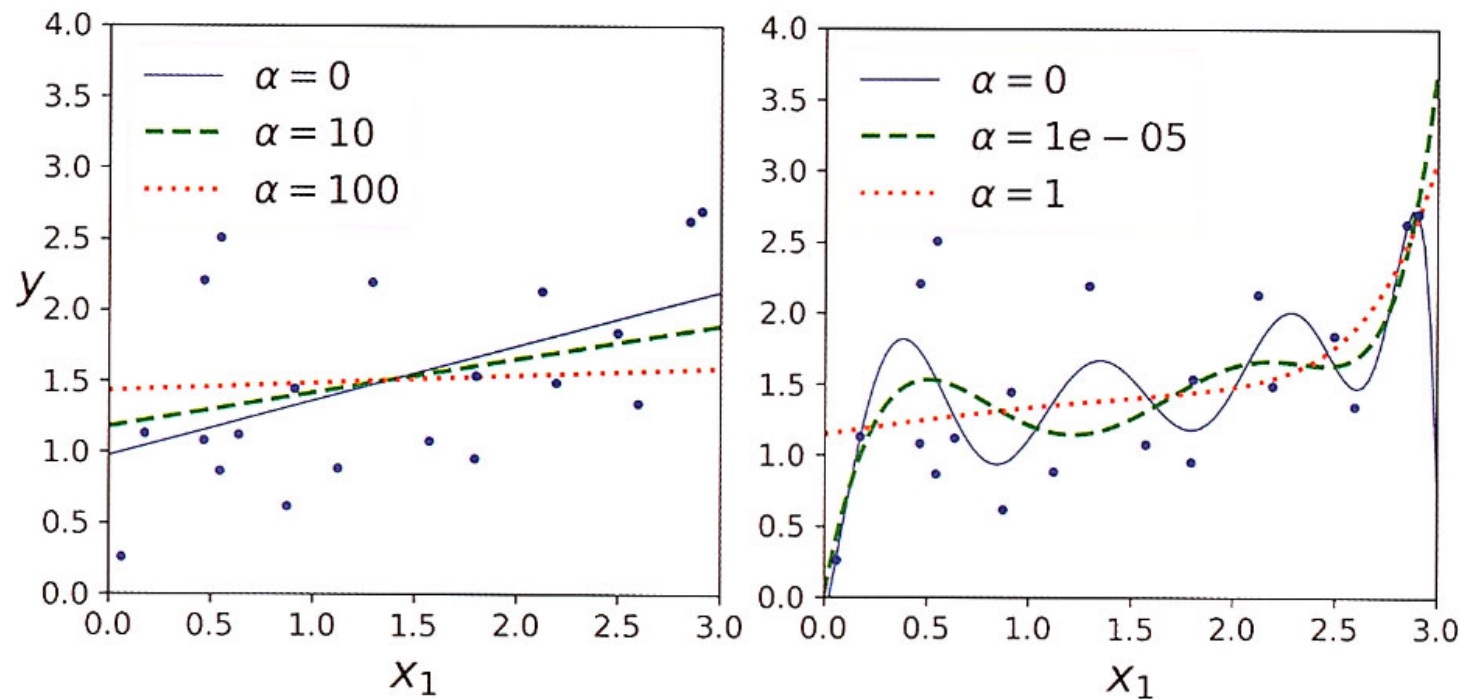
α 값으로 조절

규제 요소

릿지 회귀는 입력 특성의 스케일에 민감하기 때문에 수행하기 전에 데이터의 스케일을 맞추는 것이 중요합니다. (예를 들면 StandardScaler를 사용하고 릿지 회귀를 사용한다.)

릿지 회귀

하이퍼 파라미터 α 는 모델을 얼마나 많이 규제할지 조절합니다. $\alpha=0$ 이면 릿지 회귀는 선형 회귀와 같아집니다. α 가 아주 크면 모든 가중치가 거의 0에 가까워지고, 결국 데이터의 평균을 지나는 수평선이 됩니다.



라쏘 회귀

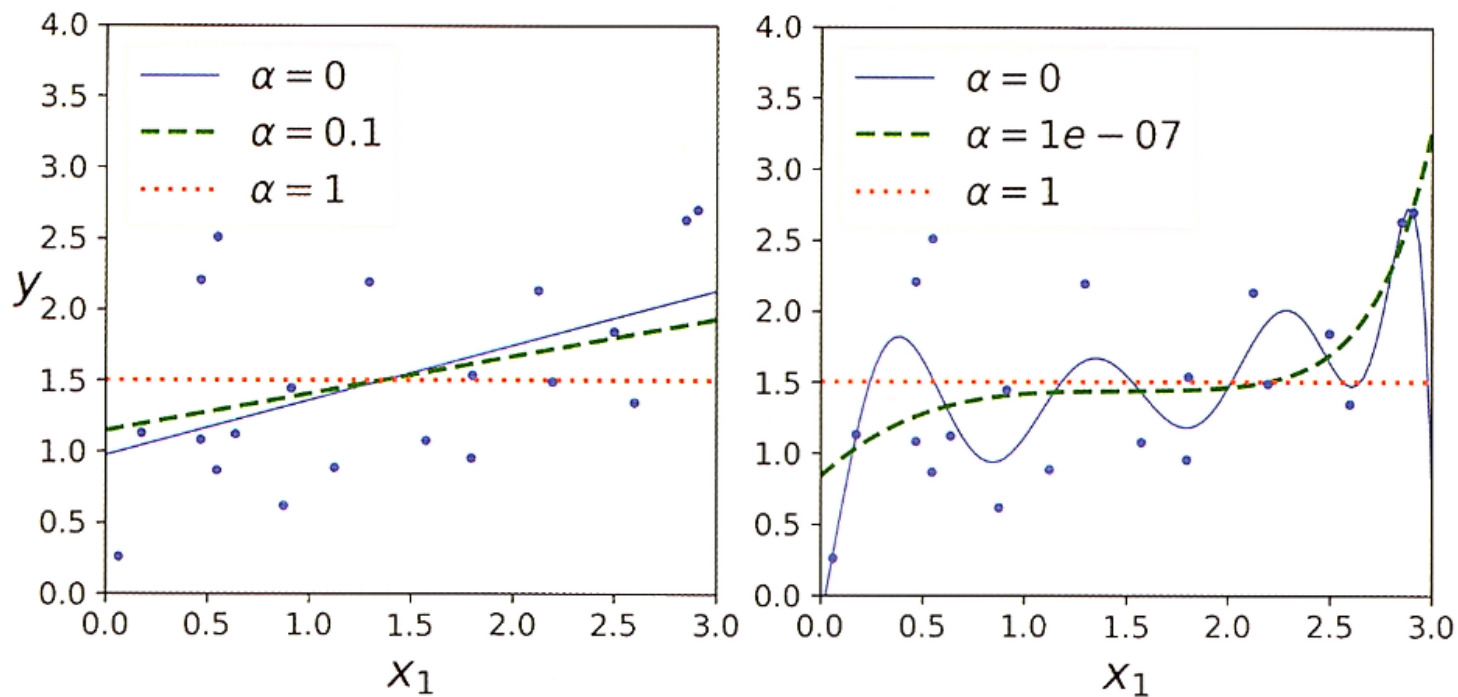
라쏘 회귀는 선형 회귀의 또 다른 규제 버전입니다. 릿지 회귀처럼 비용함수에 규제항을 더하지만, L2 NORM의 제곱을 2로 나눈 것 대신 가중치 벡터의 L1 NORM을 사용합니다.

$$J(\theta) = \text{MSE}(\theta) + \alpha \sum_{i=1}^n |\theta_i|$$

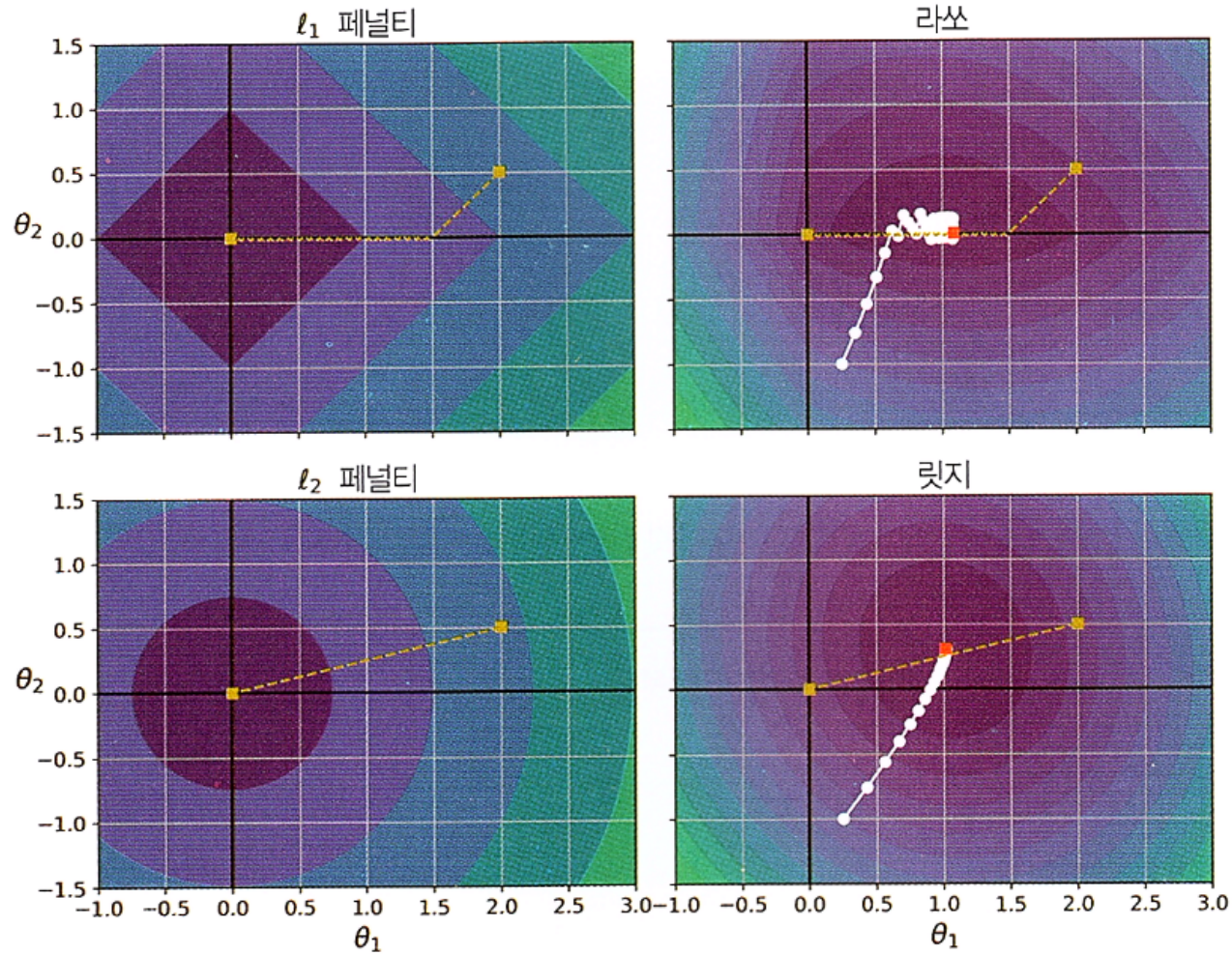
α 값으로 조절

릿지 회귀

라쏘 모델은 릿지 모델보다 조금 더 작은 α 값을 사용했다.



릿지 회귀 VS 라쏘 회귀



성능 측정 지표

성능 지표를 선택할 때, 회귀 문제는 전형적으로 평균제곱근 오차를 많이 사용합니다. 오차가 커질수록 이 값은 더욱 커지므로 예측에 얼마나 많은 오류가 있는지 가늠하게 됩니다. 아래 식은 RMSE를 계산하는 공식입니다.

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m} \sum_{i=1}^m (h(\mathbf{x}^{(i)}) - y^{(i)})^2}$$

RMSE가 일반적으로 회귀 문제에 선호되는 성능 측정 방법이지만 경우에 따라 다른 함수를 사용할 수 있습니다. 예를 들어 이상치(Outlier)로 보이는 값이 많다면, 이런 경우에는 평균 절대 오차(MAE)를 고려해 볼 수 있습니다.

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^m |h(\mathbf{x}^{(i)}) - y^{(i)}|$$

성능 측정 지표

RMSE와 MAE 모두 예측값의 벡터와 타깃값의 벡터 사이의 거리를 재는 방법입니다. 거리 측정에는 여러가지 방법이 있습니다. 노름(NORM)도 그 중에 하나입니다.

제곱항을 합한 것의 제곱근(RMSE)계산은 유클리디안 노름(Euclidean norm)에 해당합니다. L2 NORM이라고 부르며 $\|\cdot\|_2$ 로 표시합니다.

절대값의 합을 계산하는 것은 L1 NORM에 해당하며 $\|\cdot\|_1$ 로 표기합니다. 이는 도시의 구획이 직각으로 나뉘어 있을 때 이 도시의 두 지점 사이의 거리를 측정하는 것과 같아 맨하튼 노름이라고도 합니다.

