

Classification (Basics)

Dr. Saerom Park
Statistical Learning and Computational Finance Lab.
Department of Industrial Engineering
[*psr6275@snu.ac.kr*](mailto:psr6275@snu.ac.kr)
<http://slcf.snu.ac.kr>

This document is confidential and is intended solely for the use

목차

1. 분류 알고리즘 선택
2. Linear Discriminant Analysis
3. 로지스틱 회귀(Logistic regression) – 분류 확률

Reference

- **Reading:** [Raschka. (2017), chapter 3], [Müller. (2016), chapter 2], [GÉRON. (2017), chapter 3 & 5 & 6]

분류 문제

■ 지도 학습

- 지도 학습은 입력과 출력 데이터가 있고 주어진 입력으로부터 출력을 예측하기 위한 학습 방법임
 - 분류 문제: 미리 정의된 여러 클래스 레이블 중 하나를 예측하는 것
 - 회귀 문제: 연속적인 숫자 혹은 실수를 예측하는 것
- 지도학습에서는 훈련데이터로 학습한 모델이 훈련 데이터와 특성이 같다면 처음 보는 새로운 데이터가 주어져도 정확히 예측할 것이라 기대
 - 일반화 (generalization) / 과대적합 (overfitting) vs. 과소적합 (underfitting)

■ No Free Lunch

- 모든 시나리오에 대해 최고 성능인 단일 분류기(classifier)는 없음
- 여러 알고리즘들의 성능을 비교: confusion matrix, precision, recall

■ 분류 문제와 scikit-learn

- 지도 학습 중 **분류 문제**의 경우에 예측 값을 다음의 두 함수 중 하나로 제공
 - decision_function: 예측 값으로 분류된 클래스 라벨을 제공
 - predict_proba: 예측 값으로 각 클래스 라벨일 확률을 제공

분류 알고리즘 선택

분류 알고리즘 선택

■ 알고리즘 learning process

- Feature 선택
- Metric 선택
- **분류 알고리즘 선택**
- 최적화 알고리즘 선택
- 모델 성능 평가
- 알고리즘 튜닝

■ 분류 알고리즘

- 퍼셉트론(Perceptron)
- Adaline
- 로지스틱 회귀(Logistic Regression)
- 서포트 벡터 머신(Support Vector Machine)
- 의사결정 나무(Decision Tree)
- KNN(k nearest neighbors)

LINEAR DISCRIMINANT ANALYSIS

Linear Discriminant Analysis

■ LDA?

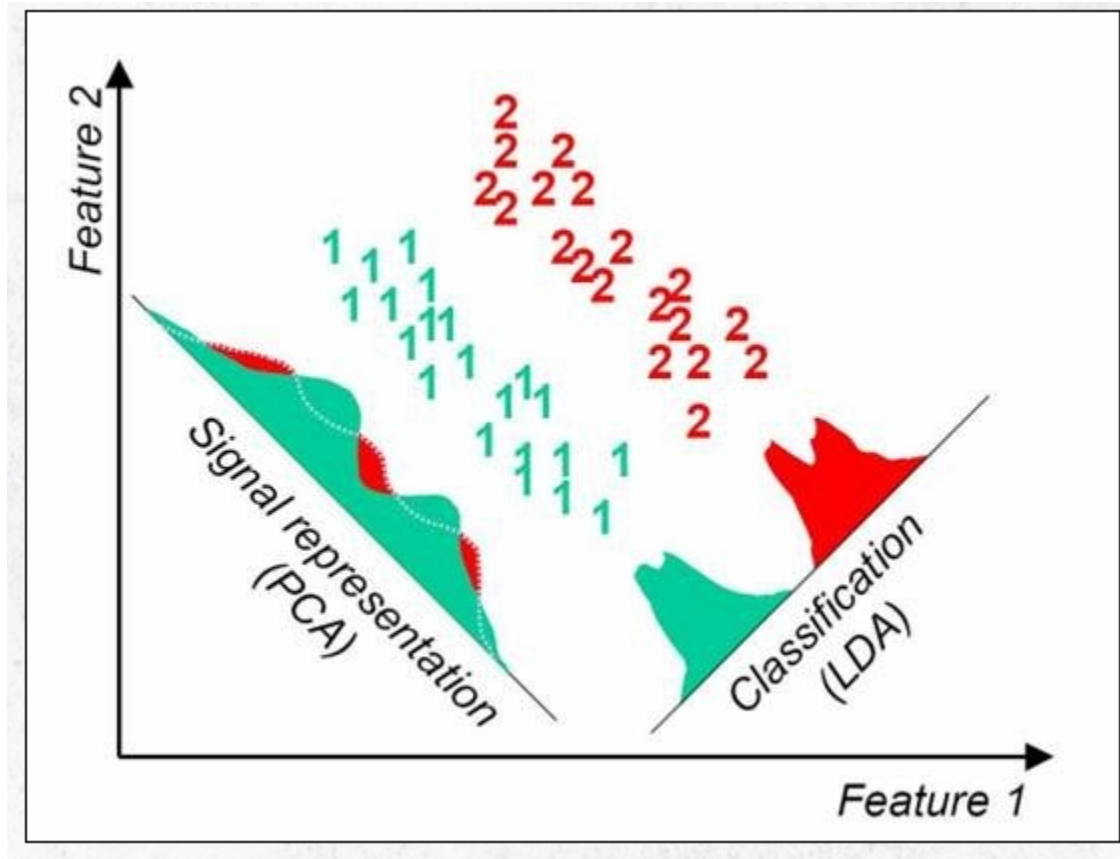
- Linear Discriminant Analysis(LDA)는 non-regularized model들에서 차원의 저주로 인 행 발생할 수 있는 overfitting을 줄이고 계산 효율을 높일 수 있는 feature extraction technique 중 하나이다.

■ PCA vs. LDA

- 공통점: 차원 축소에 쓰이는 linear transformation technique 이다.
- 차이점:
 - ① PCA는 데이터의 클래스에 상관없이 모든 원소들이 가장 넓게 퍼질 수 있는 축을 찾는 알고리즘이며, LDA는 클래스 내의 원소들은 뭉치고 다른 클래스의 원소들은 멀어지는 축을 찾는 알고리즘이다.
 - ② LDA는 데이터가 normally distributed이고, feature들이 서로 독립적이라고 가정한다.

Linear Discriminant Analysis

■ PCA vs. LDA



Linear Discriminant Analysis

■ Linear Discriminant Analysis 과정


- ① d -차원 데이터 셋을 정규화한다 .
- ② 각 class에 대하여, d -차원의 mean vector를 계산한다.
- ③ Between-class scatter matrix S_B 와 within-class scatter matrix S_w 를 계산한다.
- ④ $S_w^{-1}S_B$ 의 eigenvector와 eigenvalue들을 계산한다.
- ⑤ eigenvalue를 내림차순으로 정렬하고, 이에 맞는 eigenvector도 정렬한다.
- ⑥ k 개의 eigenvalue와 이에 맞는 eigenvector를 고르고, $d \times k$ 차원의 transformation matrix W 를 만든다(이 때, eigenvector들이 matrix의 column)
- ⑦ 데이터 샘플들을 projection matrix W 를 사용하여 new feature subspace에 project한다.

Linear Discriminant Analysis

■ Computing the scatter matrices

- PCA 부분에서 이미 정규화 해놓은 Wine dataset을 사용하자.
- 각 mean vector m_i 에는 class i 에 속한 샘플들의 mean feature value μ_m 들이 저장된다.

$$m_i = \frac{1}{n_i} \sum_{x \in D_i}^c x_m$$


feature

- 이에 따라, wine data set의 경우 세 개의 mean vector를 구할 수 있다.(class가 3가지이기 때문):

$$m_i = \begin{bmatrix} \mu_{i,alcohol} \\ \mu_{i,malic\ acid} \\ \vdots \\ \mu_{i,proline} \end{bmatrix} \quad i \in \{1,2,3\}$$

Linear Discriminant Analysis

■ Computing the scatter matrices

- Mean vector를 사용하여, within-class scatter matrix S_w 를 구할 수 있다:

$$S_w = \sum_{i=1}^c S_i$$

- 그러기 위해서는 먼저, 각 클래스 i 의 individual scatter matrix S_i 를 구해야 한다.

$$S_i = \sum_{x \in D_i} (x - m_i)(x - m_i)^T$$

- 체크: Scatter matrix들을 계산하는 데에 있어서 가정은, training set에 존재하는 class의 비율이 동일하다는 것이다(uniformly distributed). 하지만, 현재 training set의 클래스 별 숫자를 출력해보면, 그렇지 않다는 것을 알 수 있다.(Class label distribution: [41 50 33])

Linear Discriminant Analysis

■ Computing the scatter matrices

- 이에 따라, 각각의 scatter matrix S_i 에 대한 스케일링을 한 뒤에, within-class scatter matrix S_w 를 만들어야 한다.

=> 이를 위해서는 각 클래스 별 숫자 n_i 로 scatter matrices를 나누어 주어야 하는데, 이는 공분산 Σ_i 의 정의와 일치한다. 즉,

$$\Sigma_i = \frac{1}{n_i} S_i = \frac{1}{n_i} \sum_{x \in D_i}^c (x - m_i)(x - m_i)^T$$

- 이를 통해 scaled within-class matrix S_w 를 구한다.

Linear Discriminant Analysis

■ Computing the scatter matrices

- 다음으로는 between-class scatter matrix인 S_B 를 계산하여야 한다.

$$S_B = \sum_{i=1}^c n_i (m_i - m)(m_i - m)^T$$

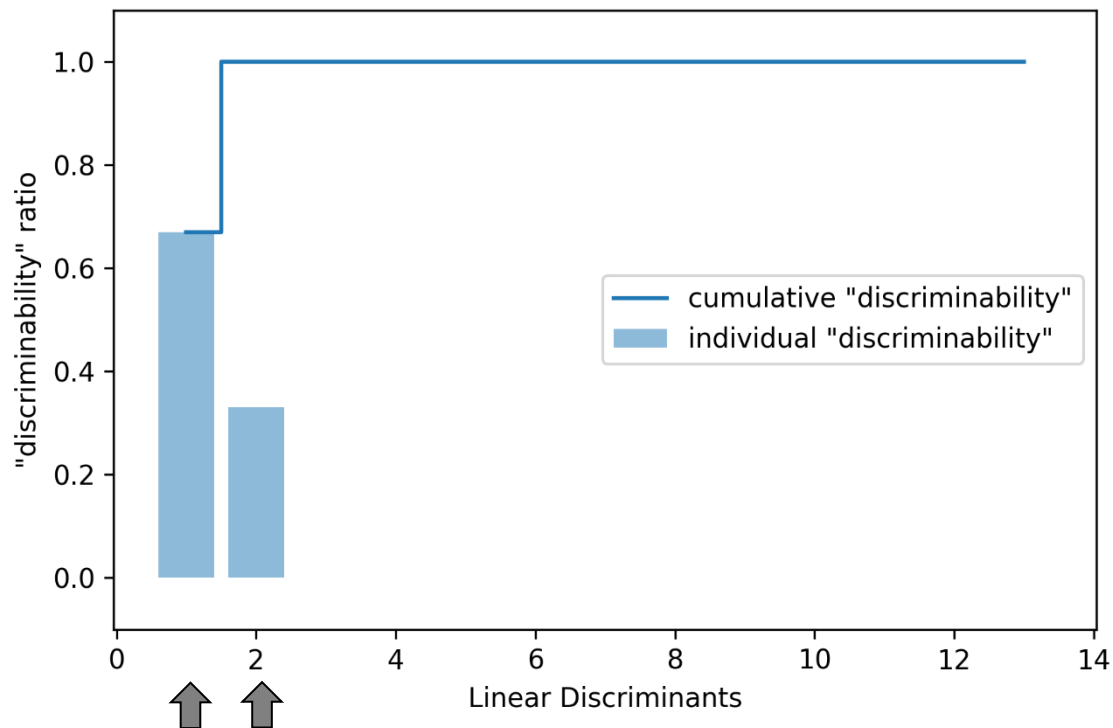
(m 은 전체 샘플의 평균)

■ Selecting linear discriminants for the new feature subspace

- PCA 과정과 비슷하게 행렬 $S_w^{-1}S_B$ 의 eigenvalue 문제를 푼다.
(Why? $S_B w = \lambda S_w w \Rightarrow S_w^{-1}S_B w = \lambda w$)
- Eigenpair들을 모두 계산한 뒤에는, eigenvalue들을 내림차순으로 정렬한다.

Linear Discriminant Analysis

- Selecting linear discriminants for the new feature subspace
 - Linear discriminant를 eigenavlue의 크기 순으로 플로팅한다.
(PCA의 explained variance와 매우 흡사)



The first two linear discriminants (almost 100% of the information)

Linear Discriminant Analysis

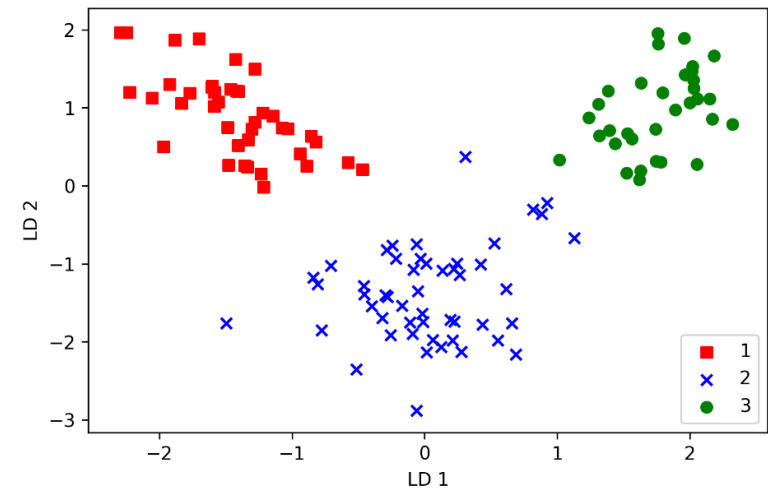
■ Feature transformation

- Selecting linear discriminants for the new feature subspace
 - 두 개의 가장 큰 eigenvalue들에 대한 eigenvector 값을 구한다.
 - 두 eigenvector를 합쳐 놓으면, 13 x 2 dimensional projection matrix W 가 완성된다.
- Transformation matrix W 를 사용하면, 기존의 training dataset X 를 변환시킬 수 있다:

$$X' = XW$$

■ Classification

- 각 class의 데이터들이 정규 분포라 가정
- 가장 스코어가 높은 class로 할당



Linear Discriminant Analysis

- LDA via scikit-learn
 - `sklearn.lda.LDA(solver, .., n_components, ...)`
 - <http://scikit-learn.org/0.16/modules/generated/sklearn.lda.LDA.html>

Methods

<code>decision_function(X)</code>	Predict confidence scores for samples.
<code>fit(X, y[, store_covariance, tol])</code>	Fit LDA model according to the given training data and parameters.
<code>fit_transform(X[, y])</code>	Fit to data, then transform it.
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>predict(X)</code>	Predict class labels for samples in X.
<code>predict_log_proba(X)</code>	Estimate log probability.
<code>predict_proba(X)</code>	Estimate probability.
<code>score(X, y[, sample_weight])</code>	Returns the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>transform(X)</code>	Project data to maximize class separation.

LOGISTIC REGRESSION

로지스틱 회귀(Logistic Regression)

■ Odds

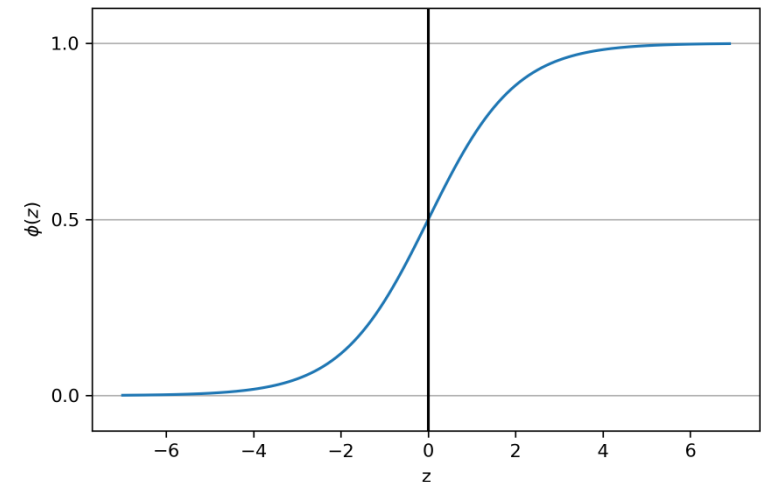
- $Odds = \frac{p}{1-p}$
- p 는 positive event의 확률

■ Logit 함수

- Log-odds
- $logit(p) = \log(Odds) = \log\left(\frac{p}{1-p}\right)$

■ 시그모이드 함수(Sigmoid function)

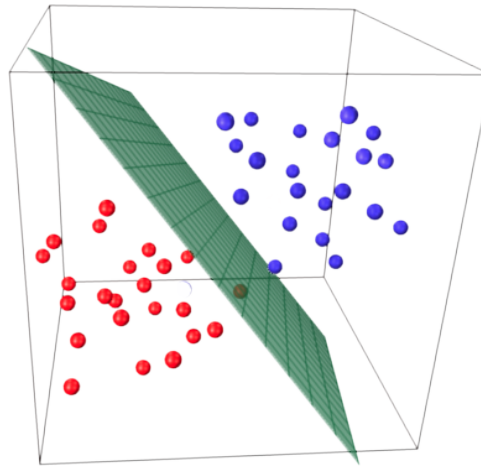
- $\phi(z) = sigmoid(z) = logit^{-1}(z) = \frac{1}{1+e^{-z}}$



로지스틱 회귀(Logistic Regression)

■ 로지스틱 회귀(Logistic Regression)

- 0,1으로 분류하는 이진분류에서 1로 분류되는 event를 positive event라 하면 'logit function을 feature의 선형결합으로 모델링'하고 학습하는 분류 모형
- $\text{logit}(P(y = 1|\mathbf{x})) = \mathbf{w}^T \mathbf{x}$
- $P(y = 1|\mathbf{x}) = \phi(\mathbf{w}^T \mathbf{x})$
- feature의 선형결합을 이용하여 선형결합값의 부호에 따라 0,1으로 이진분류



로지스틱 회귀(Logistic Regression)

로지스틱 회귀(Logistic Regression)

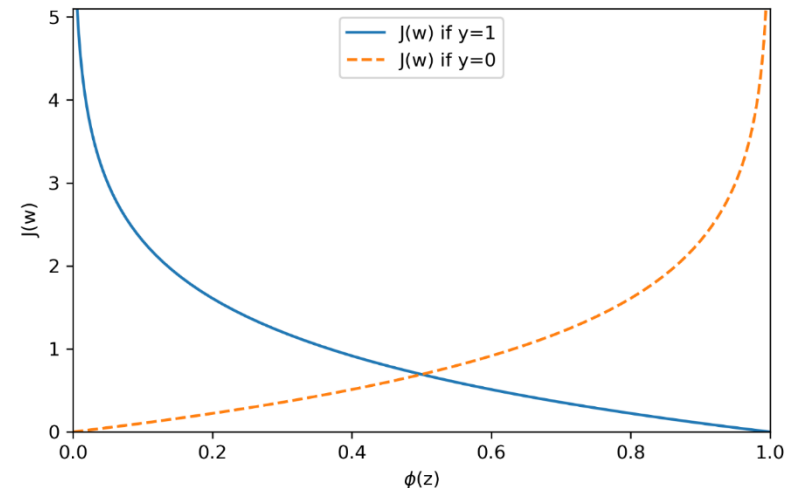
- $z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)}$
- $\text{logit} \left(P(y = 1 | \mathbf{x}^{(i)}) \right) = z^{(i)}$
- $P(y = 1 | \mathbf{x}^{(i)}) = \phi(z^{(i)})$
- $P(y = 0 | \mathbf{x}^{(i)}) = 1 - \phi(z^{(i)})$
- $P(y | \mathbf{x}^{(i)}) = \phi(z^{(i)})^y (1 - \phi(z^{(i)}))^{1-y}$

우도(Likelihood)

- $L(\mathbf{w}) = P(y | \mathbf{x}; \mathbf{w}) = \prod_i P(y^{(i)} | x^{(i)}; \mathbf{w})$
- $l(\mathbf{w}) = \log L(\mathbf{w}) = \sum_i \log P(y^{(i)} | x^{(i)}; \mathbf{w})$

비용함수(Cost function)

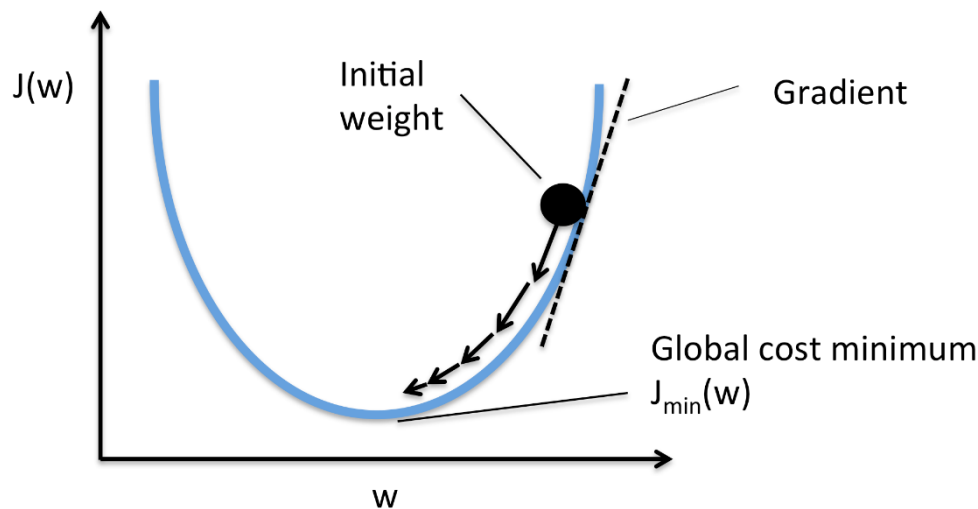
- $J(\mathbf{w}) = -l(\mathbf{w}) = -\sum_i \left(y^{(i)} \log \phi(z^{(i)}) + (1 - y^{(i)}) \log (1 - \phi(z^{(i)})) \right)$



로지스틱 회귀(Logistic Regression)

■ 학습

- 경사하강법(gradient descent)
- $J(\mathbf{w}) = -l(\mathbf{w}) = -\sum_i \left(y^{(i)} \log \phi(z^{(i)}) + (1 - y^{(i)}) \log (1 - \phi(z^{(i)})) \right)$
- $\mathbf{w}' = \mathbf{w} - \eta \text{grad } J(\mathbf{w})$



로지스틱 회귀(Logistic Regression)

■ scikit-learn 이용 로지스틱 회귀(Logistic Regression)

- `from sklearn.linear_model import LogisticRegression`
- `LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1)`

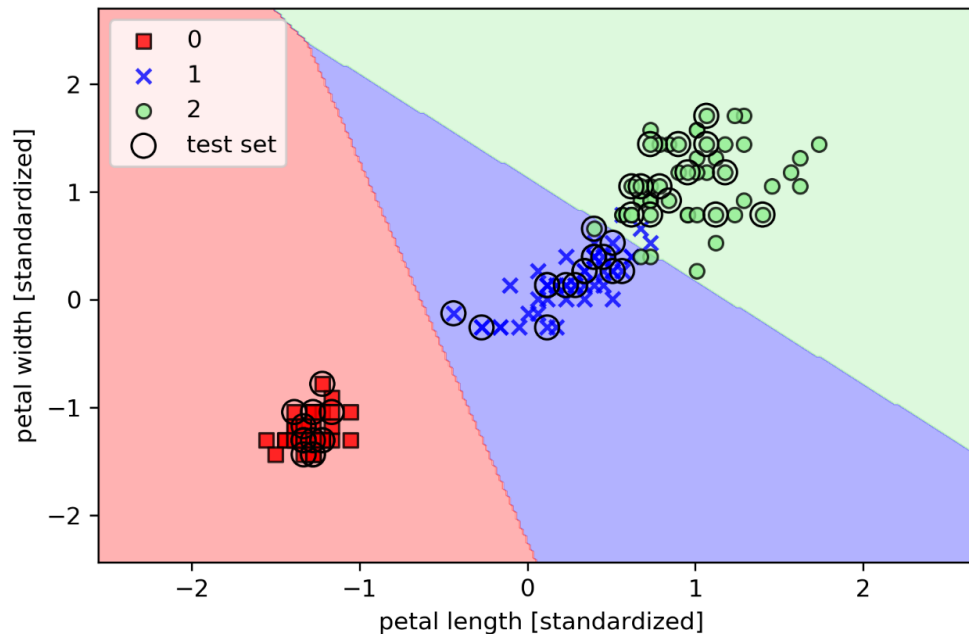
http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Parameters	
penalty	Regularization norm
tol	Tolerance
C	Regularization 계수의 역수
max_iter	최대의 iteration 횟수
multi_class	Multi-class classification 방법
n_jobs	병렬처리 방법
Attributes	
coef_	weight

로지스틱 회귀(Logistic Regression)

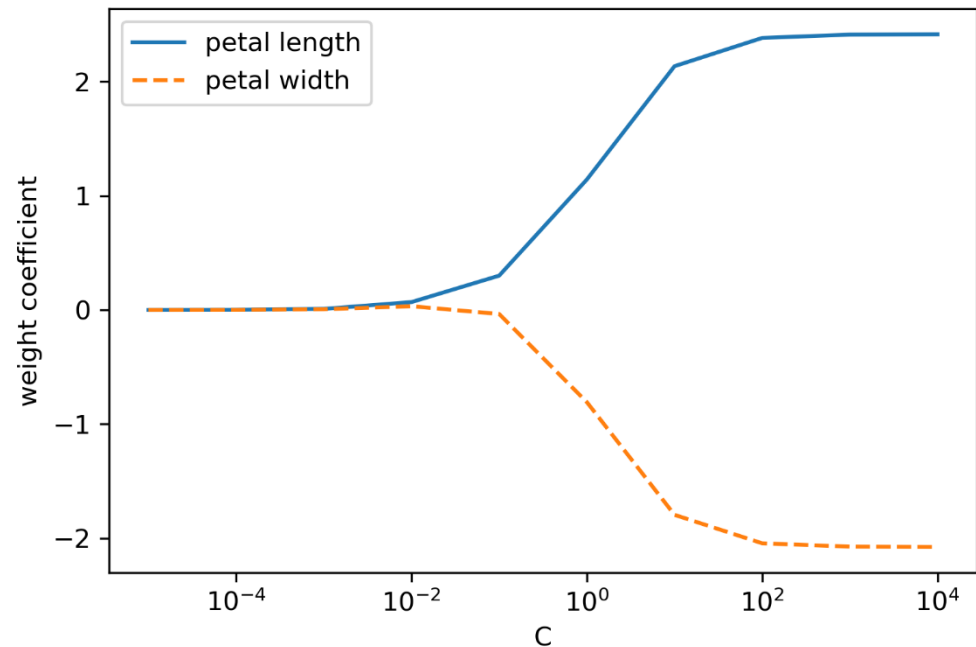
■ scikit-learn 이용 로지스틱 회귀(Logistic Regression)

- `from sklearn.linear_model import LogisticRegression`
- `LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1)`



로지스틱 회귀(Logistic Regression)

- 오버피팅(Overfitting) 방지
 - 정규화(regularization)
 - `LogisticRegression(C, random_state)`
 - C 는 정규화 파라미터의 역수



Classification 실습

■ Iris dataset

- LDA
 - 차원 감소: visualization (plot)
 - 분류 성능: test accuracy
- Logistic Regression
 - 분류 성능: test accuracy
 - Regularization
 - Regularization parameter에 따른 각 변수의 계수의 변화 확인
 - Regularization parameter의 변화에 따른 test accuracy 살펴보기

Reference

- [Raschka. (2017)] Raschka, Sebastian, and Vahid Mirjalili. *Python machine learning*. Packt Publishing Ltd, 2017.
- [Müller. (2016)] Müller, Andreas C., and Sarah Guido. *Introduction to machine learning with Python: a guide for data scientists*. 2016.
- [GÉRON. (2017)] GÉRON, Aurélien. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O, 2017.