
Logistic Regression
python (scikitlearn)
MACHINE LEARNING

사이킷런을 사용해 로지스틱 회귀 모델 훈련하기

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(solver='liblinear', multi_class='auto', C=100.0, random_state=1)
lr.fit(X_train_stand, y_train)

X_combined_stand = np.vstack((X_train_stand, X_test_stand))
y_combined = np.hstack((y_train, y_test))

plot_decision_regions(X_combined_stand, y_combined,
                      classifier=lr, test_idx=range(105, 150))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.tight_layout()
plt.show()
```

`sklearn.linear_model`: Linear Models

The `sklearn.linear_model` module implements a variety of linear models.

User guide: See the Linear Models section for further details.

The following subsections are only rough guidelines: the same estimator can fall into multiple categories, depending on its parameters.

Linear classifiers

`linear_model.LogisticRegression`([penalty, ...])	Logistic Regression (aka logit, MaxEnt) classifier.
`linear_model.LogisticRegressionCV`(*[, Cs, ...])	Logistic Regression CV (aka logit, MaxEnt) classifier.
`linear_model.PassiveAggressiveClassifier`(*)	Passive Aggressive Classifier
`linear_model.Perceptron`(*[, penalty, alpha, ...])	Read more in the User Guide.
`linear_model.RidgeClassifier`([alpha, ...])	Classifier using Ridge regression.
`linear_model.RidgeClassifierCV`([alphas, ...])	Ridge classifier with built-in cross-validation.
`linear_model.SGDClassifier`([loss, penalty, ...])	Linear classifiers (SVM, logistic regression, etc.) with SGD training.

Classical linear regressors

`linear_model.LinearRegression`(*[, ...])	Ordinary least squares Linear Regression.
`linear_model.Ridge`([alpha, fit_intercept, ...])	Linear least squares with l2 regularization.
`linear_model.RidgeCV`([alphas, ...])	Ridge regression with built-in cross-validation.
`linear_model.SGDRegressor`([loss, penalty, ...])	Linear model fitted by minimizing a regularized empirical loss with SGD

Solver

최적화의 문제를 풀어내는 이론들

	Solvers				
Penalties	**'liblinear'**	**'lbfgs'**	**'newton-cg'**	**'sag'**	**'saga'**
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes
Elastic-Net	no	no	no	no	yes
No penalty ('none')	no	yes	yes	yes	yes
Behaviors					
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

Solver

Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm

Multinomial 다항의

OVR = One vs Rest

OVO = One vs One

topmost

모두다 약자 (두문자어, acronym)

	Solvers				
Penalties	**'liblinear'**	**'lbfgs'**	**'newton-cg'**	**'sag'**	**'saga'**
Multinomial + L2 penalty	no	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	no	no	no	no	yes
OVR + L1 penalty	yes	no	no	no	yes
Elastic-Net	no	no	no	no	yes
No penalty ('none')	no	yes	yes	yes	yes
Behaviors					
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

Solver

The “saga” solver is often the best choice.

The “liblinear” solver is used by default for historical reasons.

LIBLINEAR – the winner of ICML 2008 large-scale learning challenge. It applies Automatic parameter selection

‘A Library for Large Linear Classification’ https://www.csie.ntu.edu.tw/~cjlin/liblinear/

SAG – Mark Schmidt, Nicolas Le Roux, and Francis Bach (2013).

‘Minimizing Finite Sums with the Stochastic Average Gradient’ https://hal.inria.fr/hal-00860051/document

SAGA – Defazio, A., Bach F. & Lacoste-Julien S. (2014). (a variant of SAG that also supports L1 option (L2 or L1))

‘SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives’ https://arxiv.org/abs/1407.0202

Solver

최적화의 문제를 풀어내는 이론들 → 결국 데이터에 맞는 방법 찾기

OVR = One vs Rest

	Solvers				
Penalties	**'liblinear'**	**'lbfgs'**	**'newton-cg'**	**'sag'**	**'saga'**
Multinomial + L2 penalty	~~no~~	yes	yes	yes	yes
OVR + L2 penalty	yes	yes	yes	yes	yes
Multinomial + L1 penalty	~~no~~	~~no~~	~~no~~	~~no~~	yes
OVR + L1 penalty	yes	no	no	no	yes
Elastic-Net	no	no	no	no	yes
No penalty ('none')	no	yes	yes	yes	yes
Behaviors					
Penalize the intercept (bad)	yes	no	no	no	no
Faster for large datasets	no	no	no	yes	yes
Robust to unscaled datasets	yes	yes	yes	no	no

pipeline module.Pipeline class

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
```

sklearn.pipeline.Pipeline

class sklearn.pipeline.**Pipeline**(*steps*, *, *memory=None*, *verbose=False*)

Verb 동사 말 많은,
= 지금 상황 다 설명해줘

데이터 처리, 분류 등의 steps단계를 묶어서 라인으로 만든 class의 인스턴스를 만들 수 있다.

```
lr_tfidf = Pipeline([('vect', tfidf),
                     ('clf', LogisticRegression(solver='liblinear', random_state=0))])
```

feature_extraction module

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import TfidfVectorizer
```

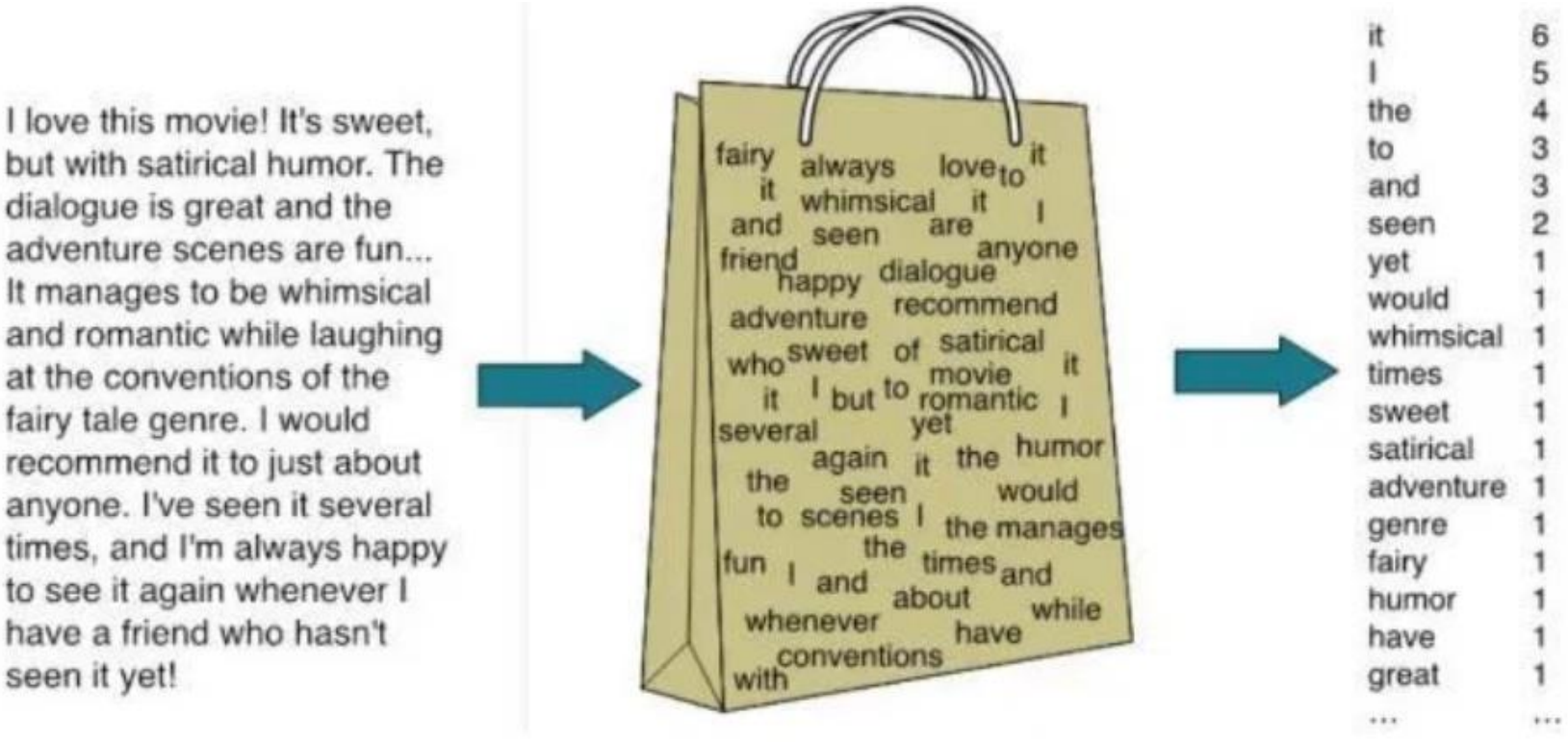
TfidfVectorizer

: 문서를 tf-idf의 feature matrix로 벡터변환하는 클래스

BOW (Bag of Words)

고정된 bag (multiset) 가방 자리 를 만들고

Di 라는 개별 문서의 가방 자리에 해당하는 단어들이 포함되어 있는지 표시



Vectorizer 벡터화 class

CountVectorizer:

문서 집합에서 단어 토큰을 생성하고

각 단어의 수를 세어 BOW 인코딩한 벡터를 만든다.

TfidfVectorizer:

TF-IDF 방식으로 단어의 중요도를 조정한 BOW 벡터를 만든다.

sklearn.feature_extraction.text.TfidfVectorizer

class sklearn.feature_extraction.text.**TfidfVectorizer**(*, *input='content'*, *encoding='utf-8'*, *decode_error='strict'*,

`sklearn.feature_extraction.text` submodule gathers utilities to build feature vectors from text documents.

sklearn.feature_extraction.text.TfidfVectorizer

```
class sklearn.feature_extraction.text.TfidfVectorizer(*, input='content', encoding='utf-8', decode_error='strict',
```

TfidfVectorizer:

TF-IDF 방식으로 단어의 가중치를 조정한 BOW 벡터를 만든다.

단순 단어 빈도로 접근하는 게 아니라,

어떤 단어가 한 문서에서 많이 나타난 동시에

다른 문서에서는 잘 나타나지 않는 것까지 고려하기 위한 개념

TF-IDF (Term Frequency-Inverse Document Frequency)