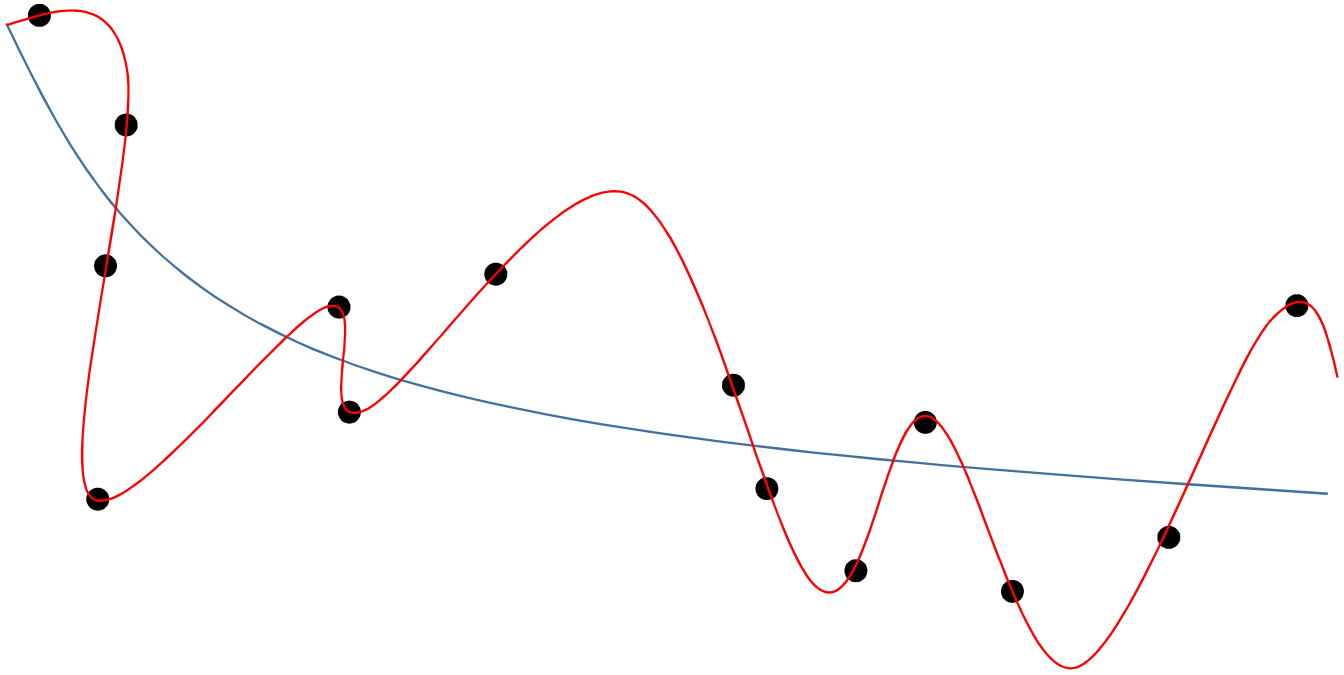
교차검증

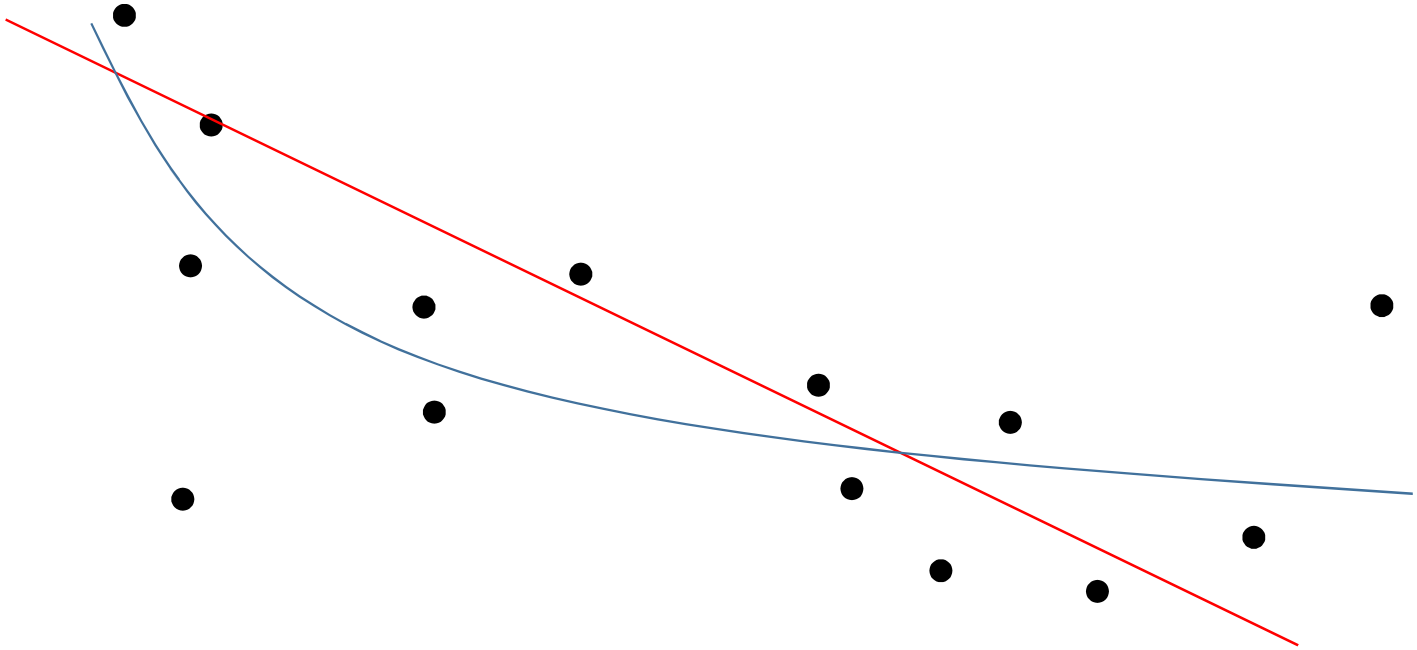
과대적합(Overfitting)

- 모델 파라미터들을 학습 데이터에 너무 가깝게 맞췄을 경우 발생하는 문제
- 학습 데이터에 과하게 학습되어 실제 데이터에서는 오차가 증가하는 현상



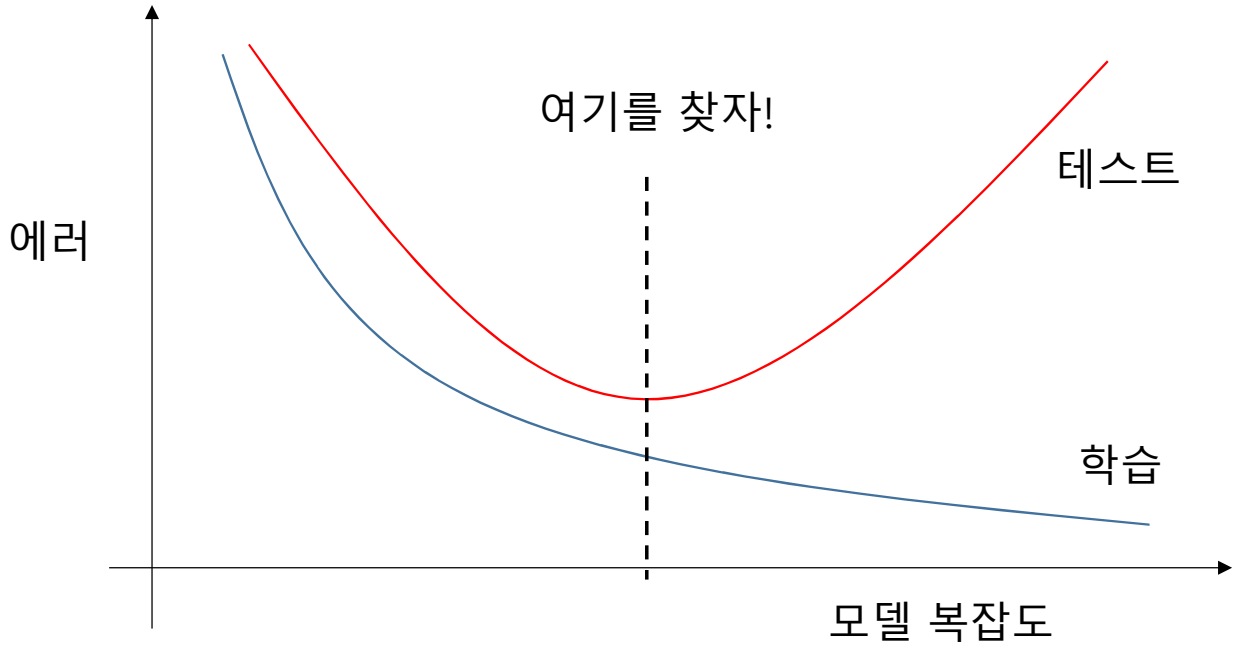
과소적합(Underfitting)

- 모델이 너무 단순하여, 학습 데이터의 특징 및 구조를 충분히 반영하지 못한 경우 발생
- 학습 데이터가 충분하지 않을 경우, 테스트 데이터가 학습 데이터 특징에 대한 유추가 힘듦



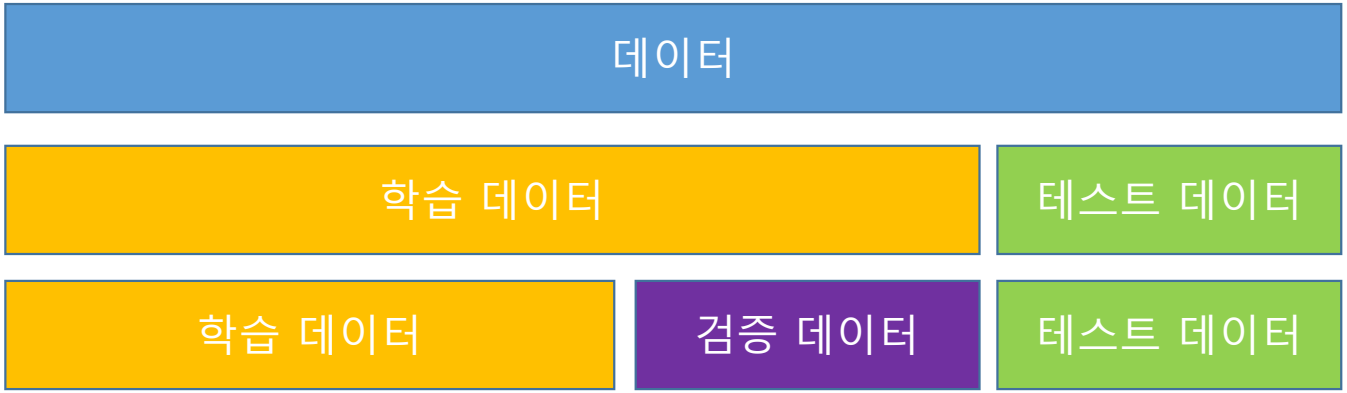
모델 최적화

- 모델이 너무 복잡하거나, 학습 데이터에 너무 최적화하여 과대적합일 경우 현실에서 사용하기 힘듦
- 모델이 너무 단순하거나, 학습이 충분하지 않아 과소적합일 경우에도 에러가 높아 사용할 수 없음



데이터 분할

- 학습(Train) 데이터 : 모델 학습하면서 파라미터를 찾기 위한 데이터
- 검증(Validation) 데이터 : 학습이 완료된 모델을 검증하기 위한 데이터
- 테스트(Test) 데이터 : 학습 과정과 무관한 모델 성능을 평가하기 위한 데이터



테스트 데이터에 과대적합 가능

학습 데이터 부족으로 과소적합 가능

검증 데이터

```python
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")
```

```python
from sklearn.model_selection import train_test_split
data2, test_data, t2, test_target = train_test_split(
    data, t, test_size = 0.3, random_state = 42, stratify = t)
```

구현(학습:검증=8:2)

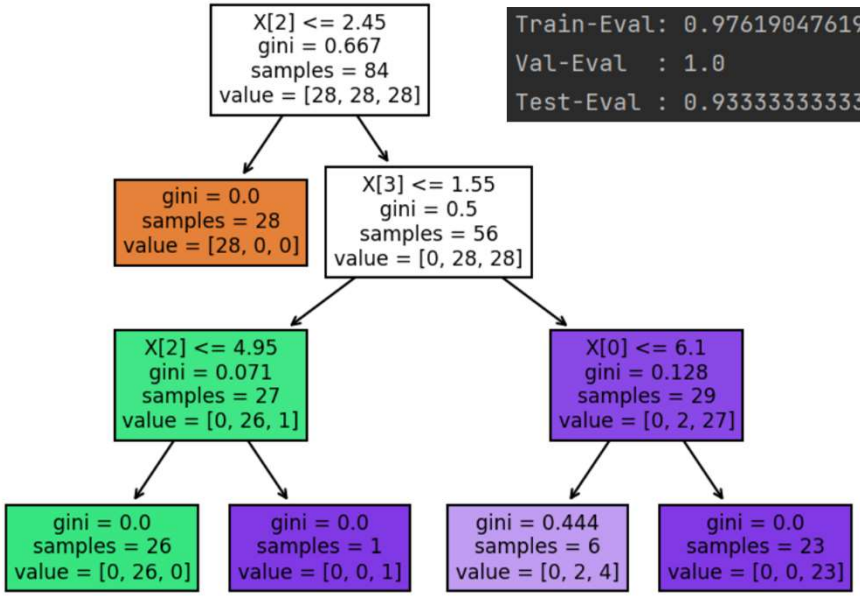
```python
# print(data.shape, t.shape)
# print(data2.shape, t2.shape)
# print(test_data.shape, test_target.shape)
# print(train_data.shape, train_target.shape)
# print(val_data.shape, val_target.shape)
```

```
(150, 4) (150,)
(105, 4) (105,)
(45, 4) (45,)
(84, 4) (84,)
(21, 4) (21,)
```

의사결정나무 분류

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth = 3, random_state = 42)
dt.fit(train_data, train_target)
print("Train-Eval:", dt.score(train_data, train_target))
print("Val-Eval  :", dt.score(val_data, val_target))
print("Test-Eval :", dt.score(test_data, test_target))
```

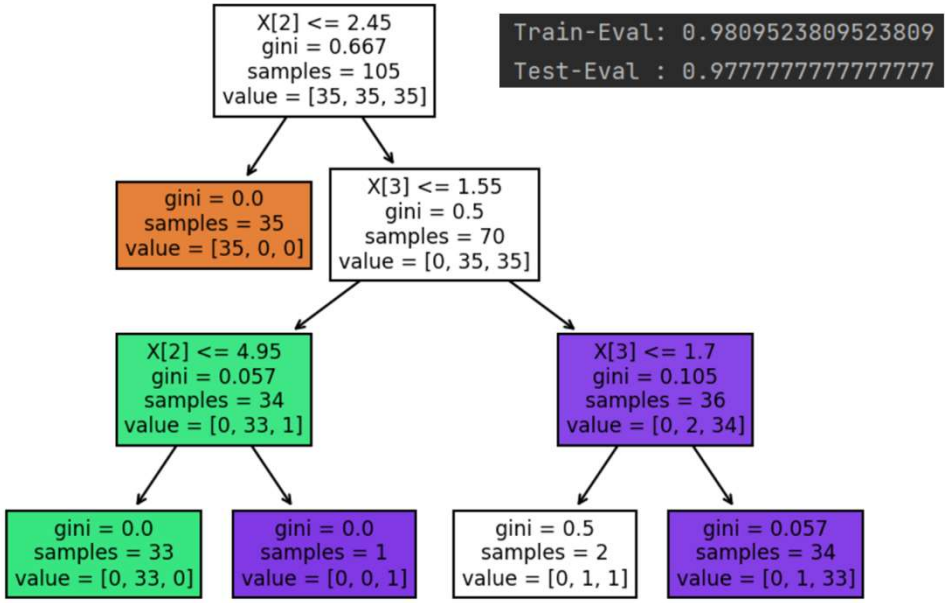
```
Train-Eval: 0.9761904761904762
Val-Eval  : 1.0
Test-Eval : 0.9333333333333333
```



학습:검증:테스트=7:0:3

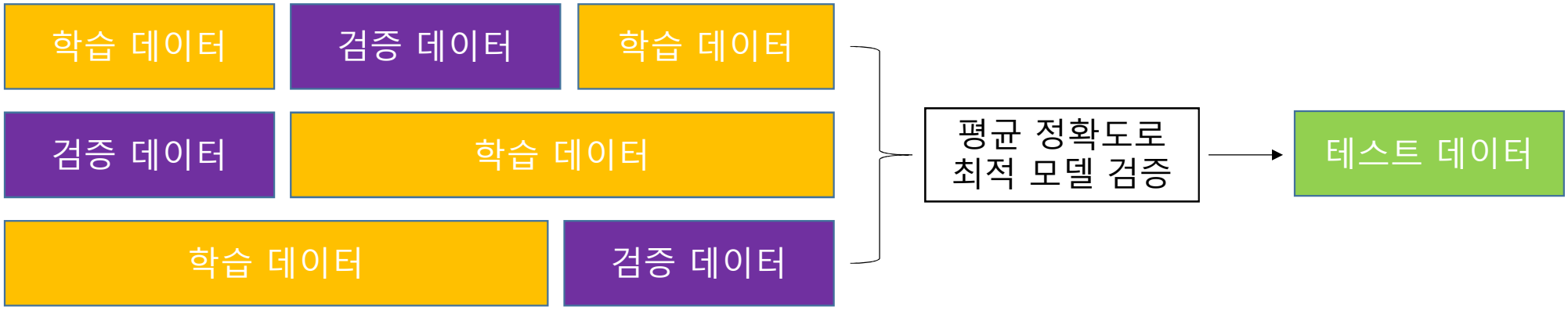
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth = 3, random_state = 42)
dt.fit(train_data, train_target)
print("Train-Eval:", dt.score(train_data, train_target))
print("Test-Eval :", dt.score(test_data, test_target))
```

```
Train-Eval: 0.9809523809523809
Test-Eval : 0.9777777777777777
```



교차검증(Cross Validation)

- 검증 데이터를 일부 떼어 내어 모델에 대한 평가를 여러 번 반복하는 과정
- 모든 데이터들을 활용함으로써, 안정적으로 학습과 검증을 수행할 수 있음
- 학습 과정에서 데스트 데이터 미사용함으로써, 테스트 데이터에 과대적합을 방지
- 검증 데이터 할당으로 인한 학습 데이터 부족으로 발생할 수 있는 과소적합 해결 가능



K-폴드(K-Fold) 교차검증

- 가장 일반적인 교차검증 방법으로, 학습 데이터 구분 수에 따라 K 결정
- 1번 폴드의 데이터로 테스트한 후, 2번 폴드, 3번 폴드, 차례대로 사용
- 각 폴드에서 계산한 결과를 평균하여 모델 평가

검증/폴드	1번 폴드	2번 플드	3번 플드	4번 플드	5번 플드
1번 검증	테스트	학습	학습	학습	학습
2번 검증	학습	테스트	학습	학습	학습
3번 검증	학습	학습	테스트	학습	학습
3번 검증	학습	학습	학습	테스트	학습
3번 검증	학습	학습	학습	학습	테스트

평균 정확도로
최적 모델 검증

sklearn.model_selection.cross_validate

sklearn.model_selection.cross_validate(*estimator, X, y=None, *, groups=None, scoring=None, cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', return_train_score=False, return_estimator=False, error_score=nan*) [source]

Evaluate metric(s) by cross-validation and also record fit/score times.

scores : *dict of float arrays of shape (n_splits,)*
Array of scores of the estimator for each run of the cross validation.

A dict of arrays containing the score/time arrays for each scorer is returned.

- `test_score`
  The score array for test scores on each cv split.
- `train_score`
  The score array for train scores on each cv split.
- `fit_time`
  The time for fitting the estimator on the train set for each cv split.
- `score_time`
  The time for scoring the estimator on the test set for each cv split.

cv : *int, cross-validation generator or an iterable, default=None*
Determines the cross-validation splitting strategy. Possible inputs for cv are:

- None, to use the default 5-fold cross validation,
- int, to specify the number of folds in a `(Stratified)KFold`,

return_train_score : *bool, default=False*
Whether to include train scores. Computing training scores is used to get insights on how different parameter settings impact the overfitting/underfitting trade-off. However computing the scores on the training set can be computationally expensive and is not strictly required to select the parameters that yield the best generalization performance.

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html#sklearn.model_selection.cross_validate

교차검증 정확도 점수

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(max_depth = 3, random_state = 42)
```

```
from sklearn.model_selection import cross_validate
scores = cross_validate(model, train_data, train_target, cv = 5)
```

```
print(scores)
```

```
{'fit_time': array([0.00099659, 0.00099659, 0.00099659, 0.00099659, 0.00199318]),
'score_time': array([0.00099659, 0.00099683, 0.        , 0.        , 0.        ]),
'test_score': array([0.95238095, 0.95238095, 1.        , 0.9047619 , 0.95238095])}
```

```
import numpy as np
print(np.mean(scores["test_score"]))
```

```
0.9523809523809523
```

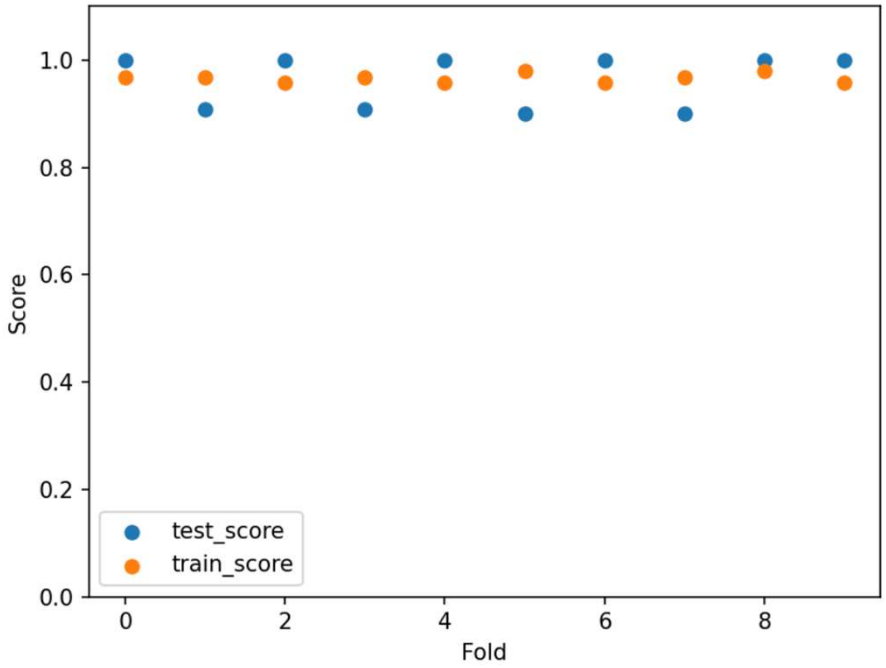
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth = 3, random_state = 42)
dt.fit(train_data, train_target)
print("Train-Eval:", dt.score(train_data, train_target))
print("Val-Eval  :", dt.score(val_data, val_target))
print("Test-Eval :", dt.score(test_data, test_target))
```

```
Train-Eval: 0.9761904761904762
Val-Eval  : 1.0
Test-Eval : 0.9333333333333333
```

서프트벡터머신

```
from sklearn.svm import LinearSVC
model = LinearSVC(random_state = 42, max_iter = 5000)
```

구현




```
import matplotlib.pyplot as plt
```

구현

```
plt.show()
```

```
print(scores)
```

```
import numpy as np
print(np.mean(scores["test_score"]))
```

```
{'fit_time': array([0.01691318, 0.02390504, 0.01694155, 0.01295543, 0.01390982,
       0.01495004, 0.0109632 , 0.01195765, 0.01196051, 0.01191545]), 'score_time': array([0.00099683, 0.0019958 , 0.00099993, 0.00203896, 0.00199127,
       0.00099897, 0.00302911, 0.00298572, 0.00099587, 0.00103307]), 'test_score': array([1.        , 0.90909091, 1.        , 0.90909091, 1.        ,
       0.9       , 1.        , 0.9       , 1.        , 1.        ]), 'train_score': array([0.96808511, 0.96808511, 0.95744681, 0.96808511, 0.95744681,
       0.97894737, 0.95789474, 0.96842105, 0.97894737, 0.95789474])}
0.9618181818181817
```

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/