# 보팅(Voting)

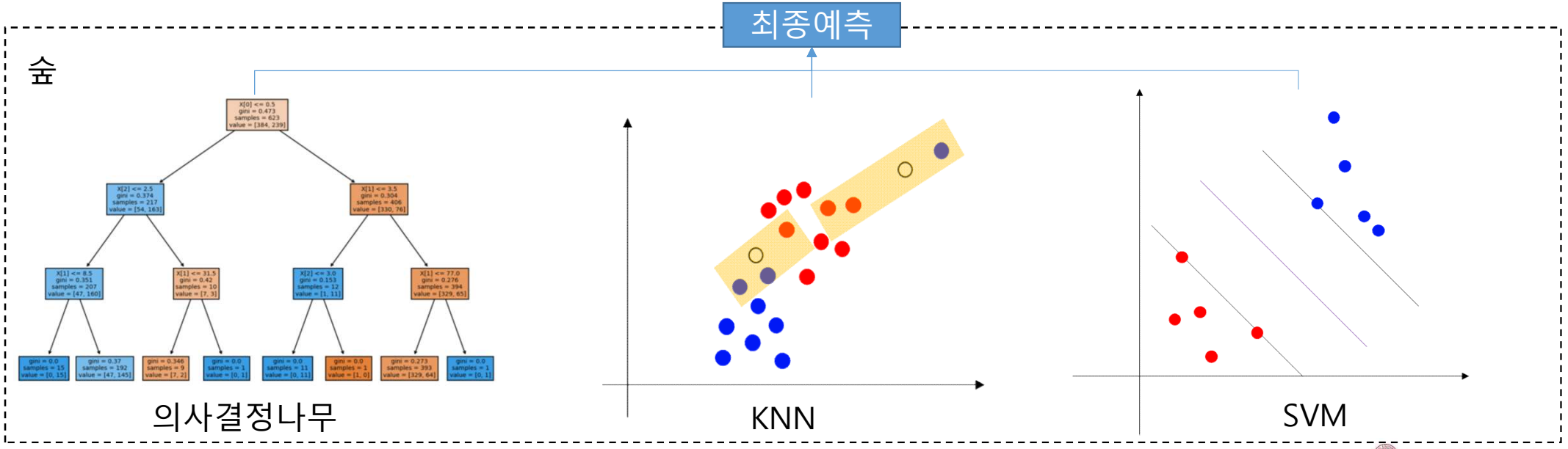
# 앙상블(Ensemble)

- 단일 모델의 성능을 보완하기 위하여 여러 개의 모델을 사용하여 최종 결과를 도출하는 방식
- 복수 개의 모델들을 결합하여 강력한 성능 제공
- 대표적으로 보팅(Voting), 배깅(Bagging), 부스팅(Boosting) 유형이 있음

| 앙상블 유형 | 대표적 모델 |
|---|---|
| 보팅 | A+B, A+B+C, A+B+C+... |
| 배깅 | 랜덤포레스트, 엑스트라트리, ... |
| 부스팅 | AdaBoost, GBM, ... |

# 보팅(Voting)

- 모델들로부터 얻은 결과들을 이용하여 하드(Hard)보팅 혹은 소프트(Soft)보팅으로 최종 결정
- 하드보팅은 다수결원칙으로 최종 결과 예측
- 소프트보팅은 각 모델의 예측 확률을 평균하여 최종 결과 예측



# sklearn.ensemble.VotingClassifier

*class* sklearn.ensemble.**VotingClassifier**(*estimators*, *, *voting='hard'*, *weights=None*, *n_jobs=None*, *flatten_transform=True*, *verbose=False*) [source]

Soft Voting/Majority Rule classifier for unfitted estimators.

**estimators : *list of (str, estimator) tuples***

Invoking the `fit` method on the `VotingClassifier` will fit clones of those original estimators that will be stored in the class attribute `self.estimators_`. An estimator can be set to `'drop'` using `set_params`.

**voting : *{'hard', 'soft'}, default='hard'***

If 'hard', uses predicted class labels for majority rule voting. Else if 'soft', predicts the class label based on the argmax of the sums of the predicted probabilities, which is recommended for an ensemble of well-calibrated classifiers.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html#sklearn.ensemble.VotingClassifier

# 보팅분류

```
import seaborn as sns
titanic = sns.load_dataset("titanic")
data = titanic[["sex", "age", "sibsp", "adult_male", "parch"]].copy()
t = titanic["survived"]
data["age"].fillna(30, inplace=True)
data["sex"].replace("male", 1, inplace=True)
data["sex"].replace("female", 0, inplace=True)

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)
```

구현(random_state=42)
** DT, KNN, LR 모델 생성

```
from sklearn.ensemble import VotingClassifier
model = VotingClassifier(estimators=[("DT", dt), ("KNN", kn), ("LR", lr)])
model.fit(train_data, train_target)
```

```
print("Train-Eval:", model.score(train_data, train_target))
print("Test-Eval :", model.score(test_data, test_target))
```

```
Train-Eval: 0.8459069020866774
Test-Eval : 0.8022388059701493
```

```
print(model.estimators_)
print(model.feature_names_in_)
```

```
[DecisionTreeClassifier(random_state=42), KNeighborsClassifier(), LogisticRegression()]
['sex' 'age' 'sibsp' 'adult_male' 'parch']
```

# 하드보팅 vs 소프트보팅

타이타닉

```
Train-Eval: 0.8459069020866774
Test-Eval : 0.8022388059701493
```

하드보팅

```
Train-Eval: 0.8812199036918138
Test-Eval : 0.7798507462686567
```

소프트보팅

---

아이리스

```
Train-Eval: 0.9809523809523809
Test-Eval : 0.9555555555555556
```

하드보팅

```
Train-Eval: 1.0
Test-Eval : 0.9555555555555556
```

소프트보팅

```
\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

# (하드)보팅분류 vs 개별분류

하드보팅

```
Train-Eval: 0.8459069020866774
Test-Eval : 0.8022388059701493
```

```python
dt.fit(train_data, train_target)
print("Train-Eval:", dt.score(train_data, train_target))
print("Test-Eval :", dt.score(test_data, test_target))

kn.fit(train_data, train_target)
print("Train-Eval:", kn.score(train_data, train_target))
print("Test-Eval :", kn.score(test_data, test_target))

lr.fit(train_data, train_target)
print("Train-Eval:", lr.score(train_data, train_target))
print("Test-Eval :", lr.score(test_data, test_target))
```

```
Train-Eval: 0.8956661316211878
Test-Eval : 0.753731343283582
Train-Eval: 0.8170144462279294
Test-Eval : 0.753731343283582
Train-Eval: 0.8170144462279294
Test-Eval : 0.8097014925373134
```

# 그리드서치(Grid Search)

- 가장 높은 성능을 갖는 최적의 하이퍼파라미터 조합을 찾는 방법
- 하이퍼파라미터 그리드는 하이퍼파라미터 조합 후보를 의미
- 모든 조합 후보들에 대해 검증 데이터를 활용한 교차검증 시도
- 하이퍼파라미터 간 모든 조합에 대해 검증하기 때문에 복잡도가 매우 높음

| p \ K | 1 | 3 | 5 | 7 | 10 |
|---|---|---|---|---|---|
| 1 | (1, 1) | (1, 3) | (1, 5) | (1, 7) | (1, 10) |
| 2 | (2, 1) | (2, 3) | (2, 5) | (2, 7) | (2, 10) |

KNN (n_neighbors, p)

# 그리드서치+보팅분류

구현(random_state=42)
** DT, KNN, LR 모델 생성

```
print("Train-Eval:", model.score(train_data, train_target))
print("Test-Eval :", model.score(test_data, test_target))
```

```
Train-Eval: 0.8426966292134831
Test-Eval : 0.7985074626865671
```

구현(KNN그리드서치)

KNN

```
Train-Eval: 0.8170144462279294
Test-Eval : 0.753731343283582
```

```
Train-Eval: 0.826645264847512
Test-Eval : 0.7798507462686567
```

그리드서치

```
from sklearn.ensemble import VotingClassifier
model = VotingClassifier(estimators = [("DT", dt), ("KNN", kn), ("LR", lr)])
model.fit(train_data, train_target)
```

# 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/