

▽ 앙상블

: 앙상블이란 여러 개의 모델을 결합하여 성능을 향상시키는 방법

이를 통해 개별 모델보다 높은 예측 성능 및 일반화 성능이 높음

- 배깅(Bagging)
 - Bootstrap Aggregating의 줄임말로, 동일한 유형의 모델 여러 개를 병렬로 학습시키고, 그 예측 결과를 평균 또는 투표하여 최종 예측을 생성
 - 여러 개의 부트스트랩 샘플을 사용

부트스트랩이란?

샘플을 뽑는데 중복을 허용해서 데이터를 뽑는 방법

- 보팅(Voting)
 - 여러 개의 모델을 학습시키고, 각 모델의 예측을 분류 모델일 때는 다수결 투표, 회귀일 경우 평균을 통해 결합
 - 하드 보팅 : 각 모델의 결과 중 가장 많은 표를 받은 클래스를 최종 예측으로 선택
 - 소프트 보팅 : 각 모델의 클래스 확률의 평균을 구하여 최종 예측을 선택
- 부스팅(Boosting)
 - 약한 학습기를 순차적으로 학습시키고, 이전 학습기의 오류를 보정하면서 강한 학습기를 만드는 방법 (잘못 예측한 데이터에 더 큰 가중치를 부여)
 - AdaBoost, Gradient, XGBoost 등이 있음
- 랜덤 포레스트(Random Forest)
 - 배깅의 한 종류
 - 여러 개의 의사 결정 트리를 학습시키고, 그 예측을 평균(회귀)하거나 투표(분류)하여 최종 예측을 생성

```
import seaborn as sns
data = sns.load_dataset('penguins')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   species               344 non-null   object
1   island                344 non-null   object
2   bill_length_mm        342 non-null   float64
3   bill_depth_mm         342 non-null   float64
4   flipper_length_mm     342 non-null   float64
5   body_mass_g           342 non-null   float64
6   sex                   333 non-null   object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

```
data.isnull().sum()
```

```
species      0
island       0
bill_length_mm  2
bill_depth_mm  2
flipper_length_mm  2
body_mass_g   2
sex          11
dtype: int64
```

```
from sklearn.impute import KNNImputer
import pandas as pd
imputer = KNNImputer(n_neighbors = 5)
data_imputed = pd.DataFrame(imputer.fit_transform(data.select_dtypes(include = 'float')), columns = data.select_dtypes(include = 'float').columns)
```

앙상블.ipynb - Colab

```
# 배깅 모델
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
bagging_model = BaggingClassifier(base_estimator = DecisionTreeClassifier(), n_estimators= 10, random_state=42)
bagging_model.fit(X_train, y_train)

y_pred = bagging_model.predict(X_test)
y_pred
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```

precision    recall  f1-score   support

0           0.98      0.96      0.97         50
1           0.86      0.95      0.90         20
2           1.00      0.97      0.99         34

accuracy          0.96         104
macro avg         0.95      0.96      0.95         104
weighted avg      0.96      0.96      0.96         104
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2
warnings.warn(
```

```
# 부스팅
from xgboost import XGBClassifier
xgb_model = XGBClassifier(n_estimators = 100, random_state = 42)
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)
print(classification_report(y_test, y_pred_xgb))
```

```

precision    recall  f1-score   support

0           0.98      0.98      0.98         50
1           1.00      0.95      0.97         20
2           0.97      1.00      0.99         34

accuracy          0.98         104
macro avg         0.98      0.98      0.98         104
weighted avg      0.98      0.98      0.98         104
```

```
from sklearn.ensemble import GradientBoostingClassifier

gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_model.fit(X_train,y_train)
y_pred_gb = gb_model.predict(X_test)

print(classification_report(y_test, y_pred_gb))
```

```

precision    recall  f1-score   support

0           1.00      0.98      0.99         50
1           1.00      1.00      1.00         20
2           0.97      1.00      0.99         34

accuracy          0.99         104
macro avg         0.99      0.99      0.99         104
weighted avg      0.99      0.99      0.99         104
```

```
from sklearn.ensemble import VotingClassifier
clf1 = GradientBoostingClassifier(random_state = 42)
clf2 = XGBClassifier(random_state = 42)

voting_clf = VotingClassifier(
    estimators = [
        ('gb', clf1),
        ('xgb', clf2)
    ], voting = 'soft'
)
voting_clf.fit(X_train, y_train)
y_pred_vt = voting_clf.predict(X_test)

print(classification_report(y_test, y_pred_vt))
```

```

precision    recall  f1-score   support

0           0.98      0.98      0.98         50
1           1.00      0.95      0.97         20
```

	2	0.97	1.00	0.99	34
accuracy				0.98	104
macro avg	0.98	0.98	0.98	0.98	104
weighted avg	0.98	0.98	0.98	0.98	104

```
# model_svm = SVC(probability = True) -> 보팅 모델 넣을 때 꼭 옵션 줘야하는 SVC
```

```
# 랜덤 포레스트
```

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators = 100, random_state = 42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
```

```
print(classification_report(y_test, y_pred_rf))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	50
1	1.00	1.00	1.00	20
2	1.00	1.00	1.00	34
accuracy			1.00	104
macro avg	1.00	1.00	1.00	104
weighted avg	1.00	1.00	1.00	104

✓ 특성 중요도

각 특성이 모델의 예측 결과에 미치는 영향을 측정할 수 있음

어떤 특성이 더 중요한 역할을 하는지 이해할 수 있음

- 트리 기반 모델 특성 중요도(Tree-Based Feature Importance)
 - 트리 기반 모델은 학습 과정에서 노드 분할에 사용된 특성의 중요도를 계산
- 피쳐 퍼뮤테이션 중요도(Feature Permutation Importance)
 - 각 특성의 값을 무작위로 섞어 모델 성능 변화를 평가
- 드롭-컬럼 중요도(Drop-Column Importance)
 - 각 특성을 하나씩 제거하고 모델의 성능 변화를 평가하여 중요도를 측정
 - 특정 특성을 제거했을 때 모델 성능이 크게 감소하면, 해당 특성은 중요한 특성임

```
from sklearn.metrics import accuracy_score
from sklearn.inspection import permutation_importance
```

```
# 1. 트리기반 특성 중요도
```

```
tree_importances = rf_model.feature_importances_
```

```
# 2. 피쳐 퍼뮤테이션 중요도
```

```
permutation_importance = permutation_importance(rf_model, X_test, y_test, n_repeats= 10, random_state=10)
permutation_importances = permutation_importance.importances_mean
```

```
# 3. 드롭컬럼 중요도
```

```
drop_importances = []
```

```
for col in X.columns:
```

```
    X_train_drop = X_train.drop(columns = [col])
```

```
    X_test_drop = X_test.drop(columns = [col])
```

```
    model_drop = RandomForestClassifier(random_state = 10)
```

```
    model_drop.fit(X_train_drop, y_train)
```


```
    y_pred = model_drop.predict(X_test_drop)
```

```
    drop_accuracy = accuracy_score(y_test, y_pred)
```

```
    drop_importances.append(accuracy_score(y_test, rf_model.predict(X_test)) - drop_accuracy)
```

```
importance_df = pd.DataFrame({"Feature" : X.columns,
                             "Tree importance" : tree_importances,
                             "Permutation importance" : permutation_importances,
                             "Drop-column importance" : drop_importances})
```

```
importances_df
```



	Feature	Tree Importance	Permutation Importance	Drop-Column Importance
0	bill_length_mm	0.333822	0.175000	0.115385
1	bill_depth_mm	0.204393	0.011538	0.000000
2	flipper_length_mm	0.236143	0.021154	0.000000
3	body_mass_g	0.078559	0.000000	0.000000
4	island_Biscoe	0.063846	0.006731	0.000000
5	island_Dream	0.068908	0.047115	0.000000
6	island_Torgersen	0.008657	0.010577	0.000000
7	sex_Female	0.003065	0.003846	0.000000
8	sex_Male	0.002608	0.003846	0.000000

코딩을 시작하거나 AI로 코드를 생성하세요.