Random Forest - 분류

```
import warnings
warnings.filterwarnings('ignore')
```

실습용 데이터 설정

- iris.csv

```
import seaborn as sns

DF = sns.load_dataset('iris')
```

- pandas DataFrame

```
DF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
DF.head(3)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa

I. 탐색적 데이터 분석

1) 빈도분석

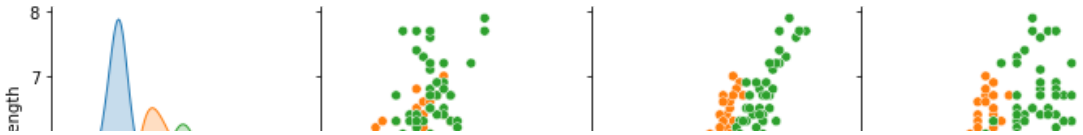
```
DF.species.value_counts()
```

```
versicolor    50
setosa        50
virginica     50
Name: species, dtype: int64
```

2) 분포 시각화

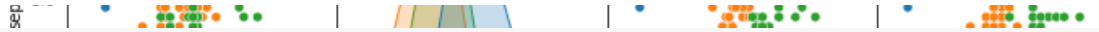
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.pairplot(hue = 'species', data = DF)
plt.show()
```



II. Data Preprocessing

1) Data Set



```
X = DF[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = DF['species']
```

2) Train & Test Split

- 7:3

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.3,
                                                    random_state = 2045)

print('Train Data : ', X_train.shape, y_train.shape)
print('Test Data : ', X_test.shape, y_test.shape)
```

```
Train Data :  (105, 4) (105,)
Test Data :  (45, 4) (45,)
```

III. Modeling

1) Train_Data로 모델 생성

- random_state : 반복 실행 시 동일한 결과 출력
- n_jobs : 모든 CPU 코어 사용

```
from sklearn.ensemble import RandomForestClassifier

Model_rf = RandomForestClassifier(n_estimators = 10,
```

```
                                    max_features = 2,
                                    random_state = 2045,
                                    n_jobs = -1)
Model_rf.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features=2,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=-1,
                       oob_score=False, random_state=2045, verbose=0,
                       warm_start=False)
```

2) Test_Data에 Model 적용

```
y_hat = Model_rf.predict(X_test)
```

3) Model Evaluate

```
from sklearn.metrics import confusion_matrix, accuracy_score

print(confusion_matrix(y_test, y_hat))
```

```
[[17  0  0]
 [ 0 14  0]
 [ 0  2 12]]
```

```
print(accuracy_score(y_test, y_hat))
```

```
0.9555555555555556
```

4) Feature Importance

- Feature Importance 값 확인

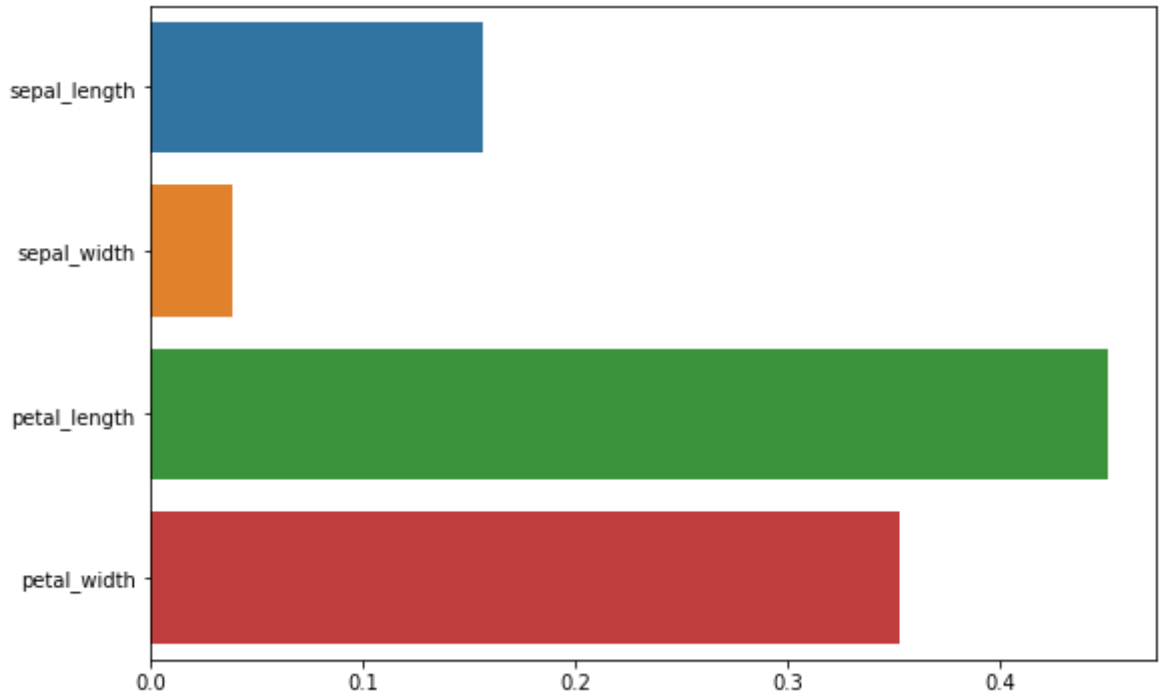
```
Model_rf.feature_importances_
```

```
array([0.1571031 , 0.03897972, 0.45102744, 0.35288974])
```

- Feature Importance 시각화

```
plt.figure(figsize = (9, 6))
```

```
sns.barplot(Model_rf.feature_importances_,
            ['sepal_length', 'sepal_width', 'petal_length', 'petal_width'])
plt.show()
```



IV. Hyperparameter Tuning

- n_estimators : 모델에 사용되는 의사결정나무의 개수
- max_features : 분할에 사용되는 Feature의 개수
- max_depth : 트리모델의 최대 깊이를 지정
- max_leaf_nodes : 말단 노드의 최대 개수
- min_samples_split : 분할을 위한 최소한의 샘플데이터 개수
- min_samples_leaf : 말단 노드가 되기 위한 최소한의 샘플데이터 개수

1) RandomForestClassifier 객체 생성

```
from sklearn.ensemble import RandomForestClassifier

Model_rf = RandomForestClassifier()
```

2) GridSearchCV Hyperparameters 설정

```
params = {'n_estimators':[100, 300, 500, 700],
          'max_features':[1, 2, 3, 4],
```

```
          'max_depth':[1, 2, 3, 4, 5],
          'random_state':[2045]}
```

3) GridSearchCV 객체 생성

- 5-Fold Cross Validation

```
from sklearn.model_selection import GridSearchCV, KFold

grid_cv = GridSearchCV(Model_rf,
                       param_grid = params,
                       scoring = 'accuracy',
                       cv = KFold(n_splits = 5,
                                  random_state = 2045),
                       refit = True,
                       n_jobs = -1)
```

4) GridSearchCV 수행

- 약 3분

```
from datetime import datetime

start_time = datetime.now()

grid_cv.fit(X_train, y_train)

end_time = datetime.now()
print('Elapsed Time : ', end_time - start_time)
```

```
Elapsed Time :  0:03:12.971505
```

5) 최적 Hyperparameter 확인

- Best Accuracy

```
grid_cv.best_score_
```

```
0.9523809523809523
```

- Best Hyperparameter

```
grid_cv.best_params_
```

```
{'max_depth': 3, 'max_features': 1, 'n_estimators': 100, 'random_state': 2045}
```

6) 최적 모델 생성 및 평가

- Best Model

```
Model_CV = grid_cv.best_estimator_
```

- Evaluation

```
y_hat = Model_CV.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix, accuracy_score

print(confusion_matrix(y_test, y_hat))
```

```
[[17  0  0]
 [ 0 13  1]
 [ 0  1 13]]
```

```
print(accuracy_score(y_test, y_hat))
```

```
0.9555555555555556
```

#

#

#

The End

#

#

#