EURON

4장 분류(1)

Hodge 2024. 3. 25. 12:51

01. 분류(Classification)의 개요

- **지도학습** : Label, 즉 명시적 정답이 있는 데이터가 주어진 상태에서 학습하는 머신러닝 방식
  -> 대표적인 방식 : **분류(Classification)**
  => 학습 데이터로 주어진 데이터의 피처/레이블값을 머신러닝 알고리즘으로 학습->모델 생성->미지의 레이블 값 예측

- 분류로 구현할 수 있는 다양한 머신러닝 알고리즘
  -> 나이브 베이즈(Naive Bayes), 로지스틱 회귀(Logistic Regression), 결정트리(Decision Tree),
  서포트 벡터 머신(Support Vector Machine), 최소근접(Nearest Neighbor) 알고리즘,
  신경망(Neural Network), **앙상블(Ensemble) <- 이번 장에서 집중적으로 다루는 알고리즘**

- **앙상블 방법(Ensemble Method)** : 서로 다른(또는 같은) 머신러닝 알고리즘을 결합한 방법
  -> 정형 데이터의 예측 분석 영역에서 매우 높은 예측 성능
  -> 일반적으로 **배깅(Bagging)**, **부스팅(Boosting)** 방식으로 나뉨
  -> 배깅의 대표적인 방식 : **랜덤 포레스트(Random Forest)** : 뛰어난 예측 성능, 상대적으로 빠른 수행 시간, 유연성
  -> But! 근래 앙상블 방법은 부스팅 방식으로 발전하고 있음 -> **그래디언트 부스팅(Gradient Boosting)**
  -> 단점 : 오랜 수행시간 => **XgBoost(eXtra Gradient Boost)와 LightGBM** 등으로 보완

- **결정 트리** : 앙상블의 기본 알고리즘
  -> 장점 : 매우 쉽고 유연하게 적용, 데이터의 스케일링/정규화 등 사전 가공의 영향이 적음
  -> 단점 : 복잡한 규칙 구조, 과적합으로 예측 성능 저하 가능성
  -> But 여러개의 약한 학습기를 결합하는 앙상블 특성상, 이게 오히려 장점으로 작용하기도 함.

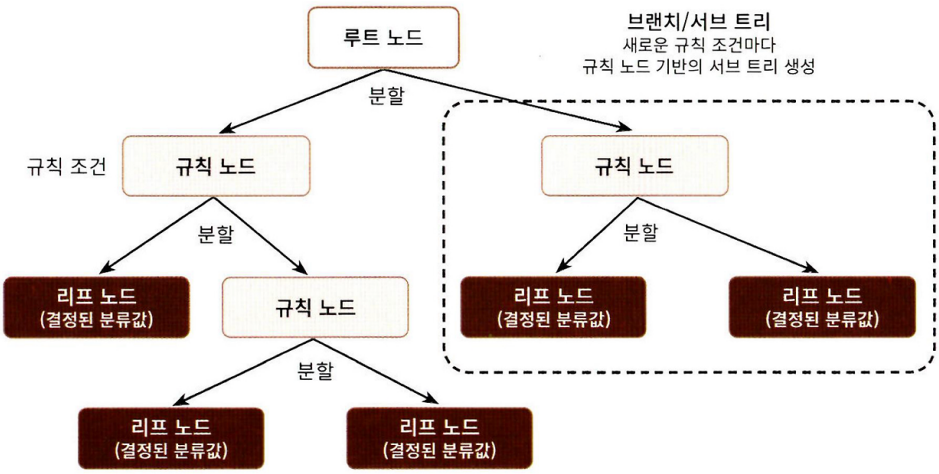
02. 결정 트리

- 결정 트리의 구조

  -> **규칙 노드(Decision Node) :** 규칙 조건

  -> **리프 노드(Leaf Node)** : 결정된 클래스 값

  -> **서브 트리(Sub Tree)** : 새로운 규칙 조건마다 생성



- 많은 규칙 -> **깊이(Depth)**가 깊어짐 -> 예측 성능의 저하 가능성 상승

- for 가능한 적은 결정 노드로 높은 예측 정확도

  -> **최대한 균일한 데이터 세트**를 구성할 수 있도록 분할

  -> 데이터 세트의 균일도 : 데이터 구분하는 데 필요한 정보의 양에 영향을 미침

- 정보 균일도 측정 방법 :

1) **정보 이득(Information Gain) 지수** : **엔트로피** 개념을 기반으로 함

  -> 엔트로피 : 주어진 데이터 집합의 혼잡도

  정보 이득 지수 : 1 - 엔트로피 지수

  -> **정보 이득이 높은 속성을 기준**으로 분할

2) **지니 계수**

  -> 지니 계수가 낮을수록 데이터 균일도 높음

  => **지니 계수가 낮은 속성을 기준**으로 분할

  -> DecisionTreeClassifier : 지니 계수를 이용해 데이터 세트 분할

1. 결정 트리 모델의 특징

- 결정 트리

  - 장점 : 균일도를 기반으로 하고 있어 쉽고 직관적인 알고리즘

    룰이 명확하고, 시각화로 표현 가능, 전처리 작업 필요 없음

  - 단점 : 과적합으로 정확도 떨어짐. 이를 극복하기 위해 트리의 크기를 사전에 제한하는 튜닝 필요

2. 결정 크리 파라미터

- 사이킷런에서 제공하는 결정 트리 알고리즘 구현 클래스
  - **DecisionTreeClassifier** : 분류를 위한 클래스
  - **DecisionTreeRegressor** : 회귀를 위한 클래스
  => **CART(Classification And Regression Trees)** 알고리즘 기반

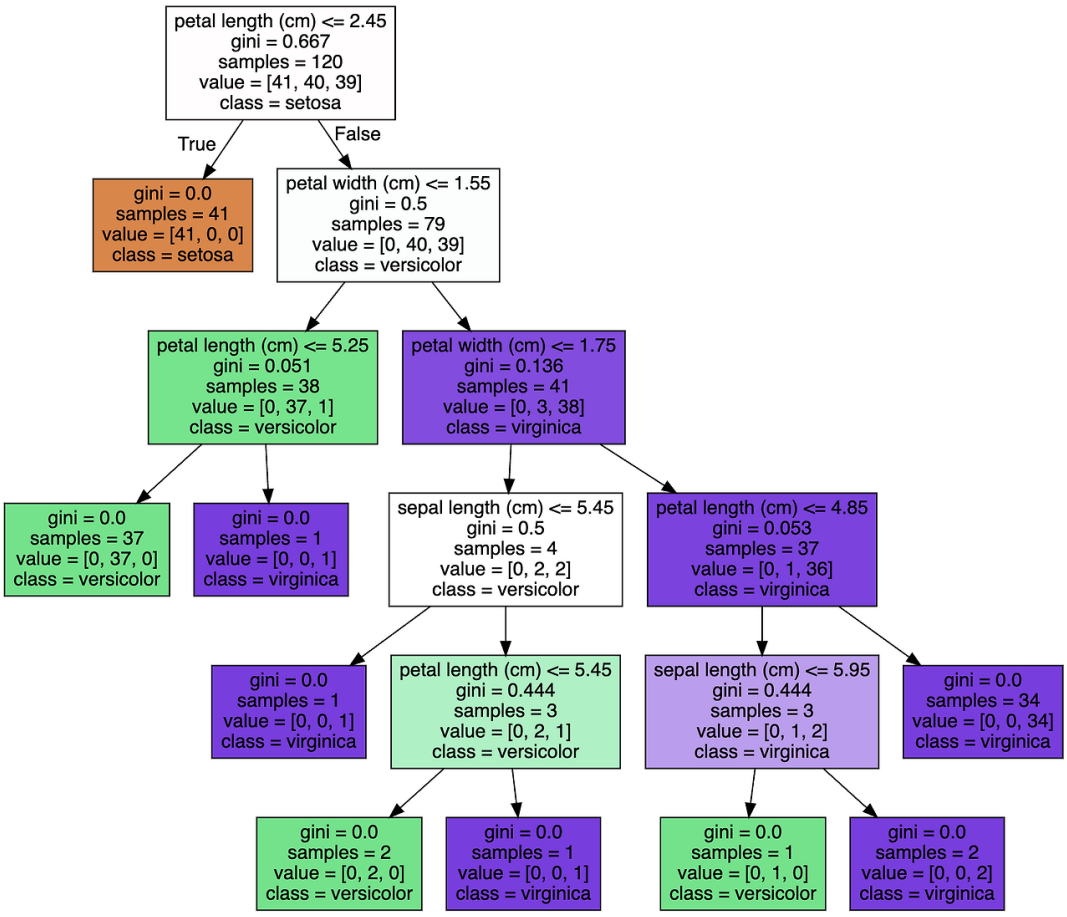
- DecisionTreeClassifier & DecisionTreeRegressor 의 공통 파라미터
  - **min_samples_split** : 노드를 분할하기 위한 최소한의 샘플 데이터 수 , 과적합 제어 용도
  - **min_samples_leaf** : 말단 노드(Leaf)가 되기 위한 최소한의 샘플 데이터 수 , 과적합 제어 용도
  - **max_features** : 최적의 분할을 위해 고려할 최대 피처 개수
  - **max_depth** : 트리의 최대 깊이 규정
  - **max_leaf_nodes** : 말단 노드(Leaf)의 최대 개수

3. 결정 트리 모델의 시각화

- **Graphviz 패키지**
  - 사이킷런은 Graphviz 패키지와 쉽게 인터페이스 할 수 있도록 export_graphviz() API 제공
    -> export_graphviz() 의 함수 인자 : 학습이 완료된 Estimator, 피처의 이름 리스트, 레이블 이름 리스트

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

#DecisionTree Classifier 생성
dt_clf = DecisionTreeClassifier(random_state=156)

#붓꽃 데이터를 로딩하고, 학습과 테스트 데이터 세트로 분리
iris_data = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris_data.data, iris_data.target,
                                                    test_size=0.2, random_state=11)

#DecisionTreeClassifier 학습
dt_clf.fit(X_train,y_train)



from sklearn.tree import export_graphviz
```

```
#export_graphviz()의 호출 결과로 out_file로 지정된 tree.dot 파일 생성
export_graphviz(dt_clf, out_file="tree.dot", class_names=iris_data.target_names, \
                feature_names = iris_data.feature_names, impurity=True, filled=True)


import graphviz
#위에서 생성된 tree.dot 파일을 Graphviz가 읽어서 주피터 노트북상에서 시각화
with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```



출력결과

- 출력결과 설명
  - petal length(cm)<=2.45 처럼 피처의 조건이 있는 것은 자식 노드를 만들기 위한 규칙 조건
    이러한 규칙 조건이 없으면 리프 노드임
  - gini : 다음의 value=[ ] 로 주어진 데이터 분포에서의 지니 계수
  - samples : 현 규칙에 해당하는 데이터 건수
  - value = [ ] : 클래스 값 기반의 데이터 건수
  - 색이 짙어질수록 지니 계수가 낮고 해당 레이블에 속하는 샘플 데이터가 많다는 뜻임.

- 하이퍼 파라미터 변경에 따른 트리 변화
  - **max_depth**

-> 결정 트리의 최대 트리 깊이 제어 => 더 간단한 결정 트리가 됨

- **min_samples_split**

-> 자식 규칙 노드를 분할해 만들기 위한 최소한의 샘플 데이터 개수 설정

- **min_samples_leaf**

-> 리프 노드가 될 수 있는 샘플 데이터 건수의 최솟값 지정

-> 값을 키우면 더 이상 분할하지 않고, 리프 노드가 될 수 있는 조건 완화

- **feature_importances**

-> 사이킷런이 중요한 역할 지표로 제공하는 DecisionTreeClassifier의 속성

- ndarray 형태로 값을 반환하며 피처 순서대로 값이 할당 됨

```
import seaborn as sns
import numpy as np
%matplotlib inline

#feature importance 추출
print("Feature importances:\n{0}".format(np.round(dt_clf.feature_importances_, 3)))

#feature별 importance 매핑
for name, value in zip(iris_data.feature_names, dt_clf.feature_importances_):
    print('{0} : {1:.3f}'.format(name, value))

#feature importance를 column 별로 시각화하기
sns.barplot(x=dt_clf.feature_importances_, y=iris_data.feature_names)
```

4. 결정 트리 모델의 과적합(Overfitting)

- **make_classification()** : 사이킷런이 분류를 위한 테스트용 데이터를 쉽게 만들 수 있도록 제공하는 함수
- visualize_boundary() : 유틸리티 함수

```
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt
%matplotlib inline

plt.title("3 Class values with 2 Features Sample data creation")

#2차원 시각화를 위해 피처는 2개, 클래스는 3가지 유형의 분류 샘플 데이터 생성
X_features, y_labels = make_classification(n_features=2, n_redundant=0, n_informative=
2,
```

```
                                                    n_classes=3, n_clusters_per_class=1, random
_state=0)

#그래프 형태로 2개의 피처로 2차원 좌표 시각화, 각 클래스 값은 다른 색으로 표시됨
plt.scatter(X_features[:,0], X_features[:,1], marker='o', c=y_labels, s=25, edgecolor
='k')
```

```
from sklearn.tree import DecisionTreeClassifier

#특정한 트리 생성 제약 없는 결정 트리의 학습과 결정 경계 시각화
dt_clf = DecisionTreeClassifier().fit(X_features, y_labels)
visualize_boundary(dt_clf, X_features, y_labels)
```

```
#min_samples_leaf=6으로 트리 생성 조건을 제약한 결정 경계 시각화

dt_clf = DecisionTreeClassifier(min_samples_leaf=6).fit(X_features, y_labels)
visualize_boundary(dt_clf, X_features, y_labels)
```

5. 결정 트리 실습 - 사용자 행동 인식 데이터 세트

03. 앙상블 학습

1. 앙상블 학습 개요

- **앙상블 학습(Ensemble Learning)**을 통한 분류 : 여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법
  - 대부분의 **정형 데이터 분류**에서 뛰어난 성능
  - 알고리즘 : **랜덤 포레스트, 그래디언트 부스팅, XGBoost, LightGBM, 스태킹(Stacking)**
  - 유형 : **보팅(Voting), 배깅(Bagging), 부스팅(Boosting). +) 스태킹**
    -> 보팅 == 배깅 : 여러 개의 분류기가 투표를 통해 최종 예측 결과 결정
    but 보팅 : **서로 다른 알고리즘**을 가진 분류기를 결합(ex: 선형 회귀, K 최근접 이웃, 서포트 벡터 머신)
    배깅 : 각각의 분류기가 **모두 같은 유형의 알고리즘** 기반, **데이터 샘플링을 다르게**(ex: 랜덤 포레스트)
    부트스트래핑(Booststrapping) 방식 : 개별 classifier에게 데이터를 샘플링해 추출하는 방식
    => **배깅 앙상블 방식** : 중첩 허용(교차검증과의 차이점)
    -> 부스팅 : 여러 개의 분류기가 순차적으로 학습을 수행하되, 다음 분류기에는 **가중치(weight)**를 부여
    ex: XGBoost, LightGBM

-> 스태킹 : 여러 가지 다른 모델의 예측 결괏값을 다시 학습 데이터로 만들어 메타 모델로 재학습시켜 결과 예측

2. 보팅 유형 - 하드 보팅(Hard Voting)과 소프트 보팅(Soft Voting)

- **하드 보팅** : **다수결의 원칙**과 비슷
  -> 예측한 결괏값들 중 다수의 분류기가 결정한 예측값을 최종 보팅 결괏값으로 선정

- **소프트 보팅** : **일반적으로 쓰임**
  -> 분류기들의 레이블 값 결정 확률을 모두 더해 **평균**해서 이들 중 확률이 가장 높은 레이블 값을 최종 보팅 결괏값으로 선정

3. 보팅 분류기(Voting Classifier)

사이킷런은 보팅 방식의 앙상블을 구현한 VotingClassifier 클래스를 제공함

ex: 위스콘신 유방암 데이터 세트를 이용한 예측 분석
load_breast_cancer()
로지스틱 회귀와 KNN을 기반으로 보팅 분류기 생성

```
import pandas as pd

from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()

data_df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
data_df.head(3)
```

로지스틱 회귀와 KNN을 기반으로 해 소프트 보팅 방식으로 새롭게 보팅 분류기 생성
VotingClassifier 클래스 이용 : 주요 생성 인자 estimators, voting

```
# 개별 모델은 로지스틱 회귀와 KNN 임.
lr_clf = LogisticRegression()
knn_clf = KNeighborsClassifier(n_neighbors=8)
```

```
# 개별 모델을 소프트 보팅 기반의 앙상블 모델로 구현한 분류기
vo_clf = VotingClassifier( estimators=[('LR',lr_clf),('KNN',knn_clf)] , voting='soft'
)

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target,
                                                    test_size=0.2 , random_state= 156)

# VotingClassifier 학습/예측/평가.
vo_clf.fit(X_train , y_train)
pred = vo_clf.predict(X_test)
print('Voting 분류기 정확도: {0:.4f}'.format(accuracy_score(y_test , pred)))

# 개별 모델의 학습/예측/평가.
classifiers = [lr_clf, knn_clf]
for classifier in classifiers:
    classifier.fit(X_train , y_train)
    pred = classifier.predict(X_test)
    class_name= classifier.__class__.__name__
    print('{0} 정확도: {1:.4f}'.format(class_name, accuracy_score(y_test , pred)))
```

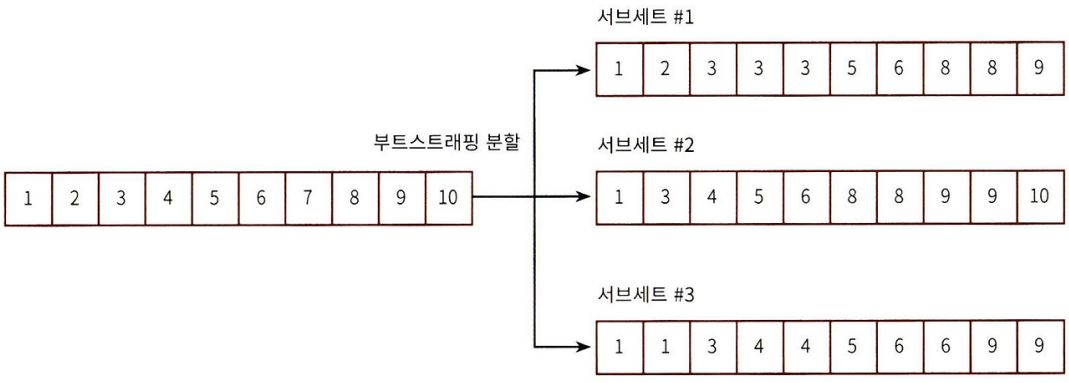
04. 랜덤 포레스트

1. 랜덤 포레스트의 개요 및 실습

랜덤 포레스트

- **결정 트리**를 기반 알고리즘으로 함
- 개별적인 분류기의 기반 알고리즘은 결정 트리이지만 개별 트리가 학습하는 데이터 세트는 전체 데이터에서 일부가 중첩되게 샘플링된 데이터 세트임

=> **부트스트래핑(bootstrapping) 분할 방식** : 이렇게 여러 데이터 세트를 중첩되게 분리하는 것.



n_estimators = 3으로 하이퍼 파라미터를 부여하는 경우의 데이터 서브세트

- 사이킷런은 **RandomForestClassifier 클래스**를 통해 랜덤 포레스트 기반의 분류를 지원함

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

# 결정 트리에서 사용한 get_human_dataset( )을 이용해 학습/테스트용 DataFrame 반환
X_train, X_test, y_train, y_test = get_human_dataset()

# 랜덤 포레스트 학습 및 별도의 테스트 셋으로 예측 성능 평가
rf_clf = RandomForestClassifier(random_state=0)
rf_clf.fit(X_train , y_train)
pred = rf_clf.predict(X_test)
accuracy = accuracy_score(y_test , pred)
print('랜덤 포레스트 정확도: {0:.4f}'.format(accuracy))
```

2. 랜덤 포레스트 하이퍼 파라미터 및 튜닝

GridSearchCV를 이용해 랜덤 포레스트의 하이퍼 파라미터 튜닝

```
from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators':[100],
    'max_depth' : [6, 8, 10, 12],
    'min_samples_leaf' : [8, 12, 18 ],
    'min_samples_split' : [8, 16, 20]
}
# RandomForestClassifier 객체 생성 후 GridSearchCV 수행
rf_clf = RandomForestClassifier(random_state=0, n_jobs=-1)
grid_cv = GridSearchCV(rf_clf , param_grid=params , cv=2, n_jobs=-1 )
grid_cv.fit(X_train , y_train)

print('최적 하이퍼 파라미터:\n', grid_cv.best_params_)
print('최고 예측 정확도: {0:.4f}'.format(grid_cv.best_score_))
```

```
#n_estimators를 300으로 증가시키고, 최적화 하이퍼 파라미터를 다시 RandomForestClassifier로 학습시킨
뒤
#별도의 테스트 데이터 세트에서 예측 성능 측정

rf_clf1 = RandomForestClassifier(n_estimators=300, max_depth=10, min_samples_leaf=8, \
                                 min_samples_split=8, random_state=0)
```

```
rf_clf1.fit(X_train , y_train)
pred = rf_clf1.predict(X_test)
print('예측 정확도: {0:.4f}'.format(accuracy_score(y_test , pred)))
```

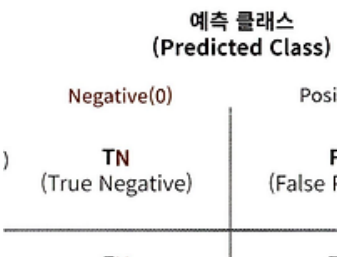
```
#feature_importances_ 속성으로 알고리즘이 선택한 피처의 중요도를 막대그래프로 시각화

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

ftr_importances_values = rf_clf1.feature_importances_
ftr_importances = pd.Series(ftr_importances_values,index=X_train.columns  )
ftr_top20 = ftr_importances.sort_values(ascending=False)[:20]

plt.figure(figsize=(8,6))
plt.title('Feature importances Top 20')
sns.barplot(x=ftr_top20 , y = ftr_top20.index)
plt.show()
```

공감

'EURON' 카테고리의 다른 글

3장 평가 (0) 2024.03.18
2장 사이킷런으로 시작하는 머신러닝 (0) 2024.03.18

'EURON' Related Articles



3장 평가

2장 사이킷런으로 시작...