4. 분류 (1)

goblurry · 방금 전

데이터분석 머신러닝 파완머

파이썬 머신러닝 완벽가이드

📊 파이썬 머신러닝 완벽 가이드 (위키북스, 개정 2판) 교재를 바탕으로 학습한 내용을 정리한 포스트입니다.
https://github.com/goblurry/ML-Study 에서 예시 코드를 확인할 수 있습니다.

> 1. 분류의 개요
> 2. 결정 트리
> 3. 앙상블 학습
> 4. 랜덤 포레스트

분류의 개요

분류(Classification): 지도학습의 대표 유형. 기존 데이터가 어떤 레이블에 속하는지 패턴을 알고리즘으로 인지한 후 새로 관측된 데이터에 대한 레이블을 판별.

- 학습 데이터로 주어진 데이터의 피처와 레이블 값을 머신러닝 알고리즘으로 학습해 모델 생성
- 생성된 모델에 새로운 데이터 값이 주어지면 미지의 레이블 값을 예측
- Naive Bayes(나이브 베이즈), 로지스틱 회귀, 결정 트리, 서포트 벡터 머신, 최소 근접 알고리즘, 신경망, 앙상블 등의 ML 머신러닝으로 구현 가능

앙상블(Ensemble): 정형 데이터의 예측 분석 영역에서 매우 높은 예측 성능을 보여 사랑받는 알고리즘. (음성, 영상, 이미지, NLP 영역에서는 신경망에 기반한 딥러닝이 인기 있다.)

일반적 분류: 배깅(Bagging), 부스팅(Boosting)
1. 랜덤 포레스트: 대표적인 '배깅' 방식.
2. 근래의 앙상블 방법은 부스팅 방식으로 지속해서 발전 중.

- 효시: 그래디언트 부스팅. 예측 성능이 좋지만 수행 시간이 너무 오래 걸려 최적화 모델 튜닝이 어려웠음.
- XGBoost, LightGBM 등 기존 그래디언트 부스팅의 예측 성능을 발전시키면서 수행 시간은 단축시킨 알고리즘이 등

3. 앙상블은 대부분 동일한 알고리즘을 결합하고, 기본 알고리즘으로는 **결정 트리** 를 사용한다.

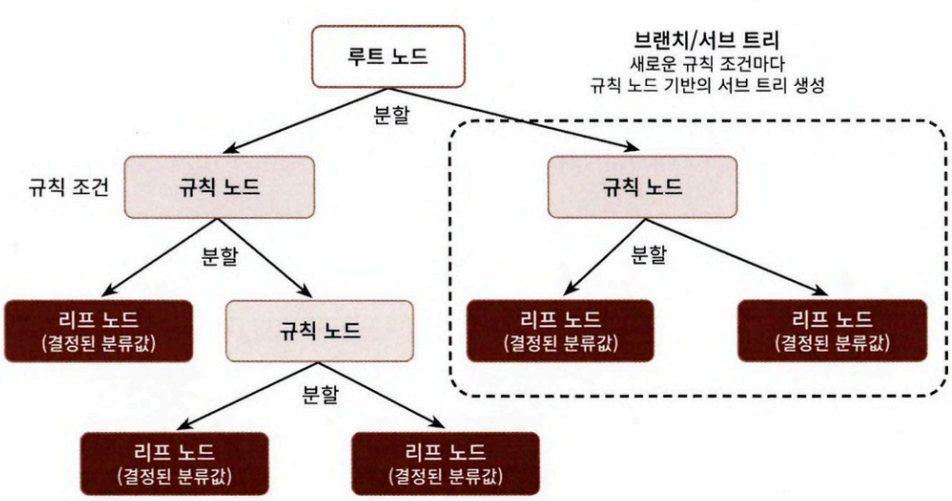
결정 트리: 쉽고 유연하게 적용되는 알고리즘. 데이터의 사전 가공(스케일링, 정규화)의 영향이 적다.
하지만 예측 성능을 향상시키기 위해 복잡한 규칙을 가져서 과적합이 발생해 오히려 성능이 떨어질 수도 있는데,

이 점이 앙상블 기법에서는 오히려 장점이 될 수 있다. (앙상블: 약한 학습기 여러 개를 결합해 오류 발생 위치에 대한 가중치를 계속 업뎃하면서 성능을 향상시키는데, 이때 결정트리가 '약한 학습기'로서 좋은 역할을 함.)

결정 트리

데이터에 있는 규칙을 학습을 통해 자동으로 찾아내 tree 기반의 분류 규칙을 만드는 것.

- if/else 기반으로 가장 쉽게 규칙을 표현할 수 있음
  데이터의 어떤 기준을 바탕으로 규칙을 만들어야 가장 효율적인 분류가 될 것인지가 알고리즘의 성능을 좌우함.



피처가 결합해 규칙 조건을 만들 때마다 규칙 노드가 생성되는데, 많은 규칙이 있다 = 분류 결정하는 방식이 더욱 복잡해진다 => 이는 과적합으로 이어진다.
즉, **트리의 깊이가 깊어질수록 결정 트리의 예측 성능이 저하될 가능성이 높다.**

최소한의 결정 노드로 높은 예측 정확도를 가지려면, 데이터를 분류할 때 최대한 많은 데이터 세트가 해당 분류에 속할 수 있도록 결정 노드의 규칙이 정해져야 함. 어떻게 트리를 분할할 것인가 가 중요 - 최대한 데이터 세트를 균일하게 구성하도록 분할해야 함.

결정 노드는 **정보 균일도가 높은** 데이터 세트를 먼저 선택하도록 규칙 조건을 만든다. (데이터 세트의 균일도는 데이터를 구분하는 데 필요한 정보의 양에 영향을 미침. 균일도가 높아야 예측이 쉬움.)
조건에 맞는 서브 데이터 세트 만들고, 그 서브 데이터 세트에서 균일도 높은 데이터 세트를 또 쪼개고... 그런 방식으로 반복해서 내려가는 방식으로 데이터 값을 예측한다.

정보의 균일도를 측정하는 방법: 엔트로피를 이용한 정보 이득 지수 & 지니 계수
결정 트리 알고리즘을 사이킷런에서 구현한 DecisionTreeClassifier: 기본으로 지니 계수를 이용해 데이터 세트를 분할함.
데이터 세트를 분할하는 데 가장 좋은 조건을 찾아서, 자식 트리 노드에 걸쳐 반복적으로 분할한 뒤, 데이터가 모두 특정 분류에 속하게 되면 분할을 멈추고 분류를 결정함.

결정 트리 모델의 특징

[장점]
1. 정보의 균일도를 rule로 하고 있어 알고리즘이 쉽고 직관적
2. 규칙/리프 노드가 어떻게 만들어지는지 알 수 있고, 시각화도 가능
3. 균일도 외 고려사항 없음: 사전 가공 작업(피처의 스케일링, 정규화) 불필요

[단점]
1. 과적합으로 알고리즘 정확도 떨어짐

- 피처가 많고 균일도가 다양할수록 트리의 깊이가 커지고 복잡해짐
- 트리의 크기를 사전에 제한하는 튜닝이 필요함

결정 트리 파라미터

사이킷런에서 제공하는 결정 트리 알고리즘 클래스: `DecisionTreeClassifier` , `DecisionTreeRegressor` 각각 분류와 회귀를 위한 클래스

파라미터 (분류, 회귀 모두 동일하게 사용)

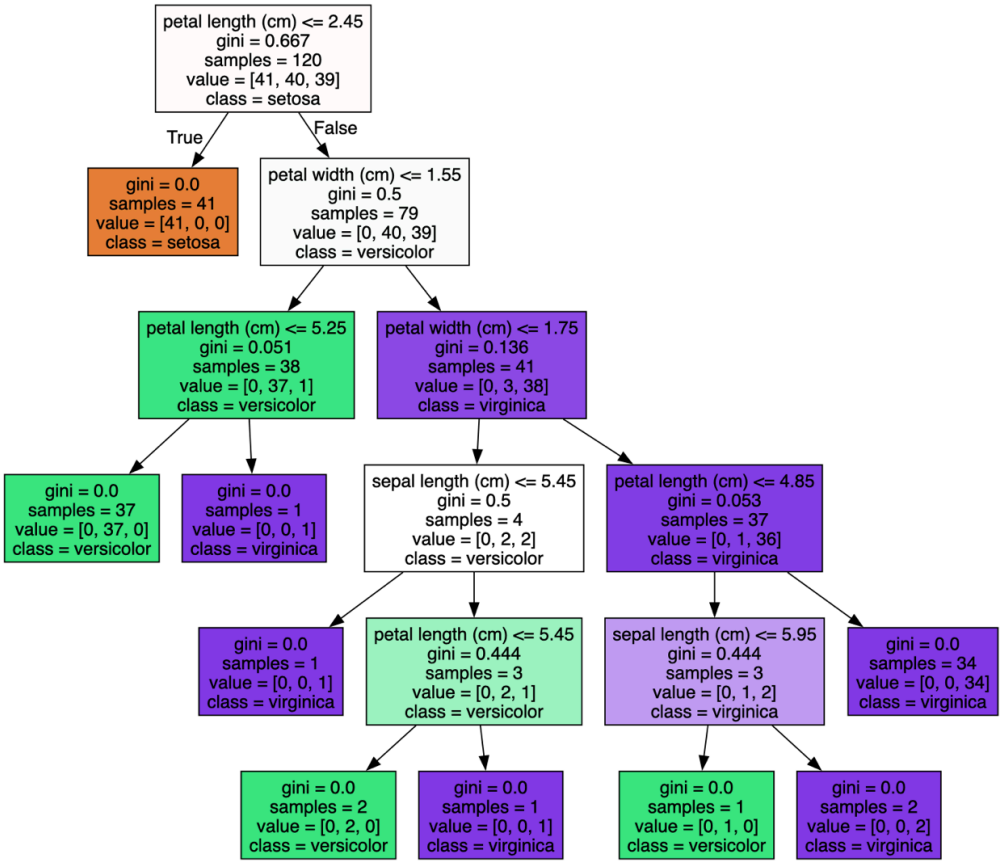
> 1. min_samples_split: 노드를 분할하기 위한 **최소한의 샘플 데이터 수**, 과적합 제어용 (디폴트 2. 작게 설정할수록 분할 노드가 많아져 과적합 가능성 증가)
> 2. min_samples_leaf: 분할될 경우 좌우의 브랜치 노드에서 가져야 할 최소한의 샘플 데이터 수, 과적합 제어용 (그러나 비대칭적 데이터의 경우에는 특정 클래스의 데이터가 극도로 작을 수 있으므로 이 경우 작게 설정해야 함)
> 3. max_features: 최적의 분할을 위해 고려할 최대 피처 개수. (디폴트 None, 데이터 세트의 모든 피처 사용해 분할)
> 4. max_depth: 트리의 최대 깊이를 규정. (디폴트 None)
> 5. max_leaf_nodes: 말단 노드(Leaf)의 최대 개수

결정 트리 모델의 시각화

Graphviz 패키지를 사용하여 결정 트리 알고리즘이 어떤 규칙으로 트리를 생성하는지 시각적으로 확인이 가능하다. 사이킷런에서 제공하는 export_graphviz() API를 활용해, 함수 인자로 **학습이 완료된 Estimator, 피처의 이름 리스트, 레이블 이름 리스트** 를 입력하면 학습된 결정 트리 규칙을 실제 트리 형태로 시각화해 볼 수 있다.

```
!pip install graphviz
```

[11]:



결정 트리 규칙 구성

자식 노드가 더 없는 노드: '**리프 노드**'

- 최종 클래스(레이블) 값이 결정되는 노드
- 하나의 클래스 값으로 최종 데이터 구성되거나, 리프 노드가 될 수 있는 하이퍼 파라미터 조건을 충족하면 됨

자식 노드가 있는 노드: '**브랜치 노드**'

- 자식 노드를 만들기 위한 분할 규칙 조건을 가짐.

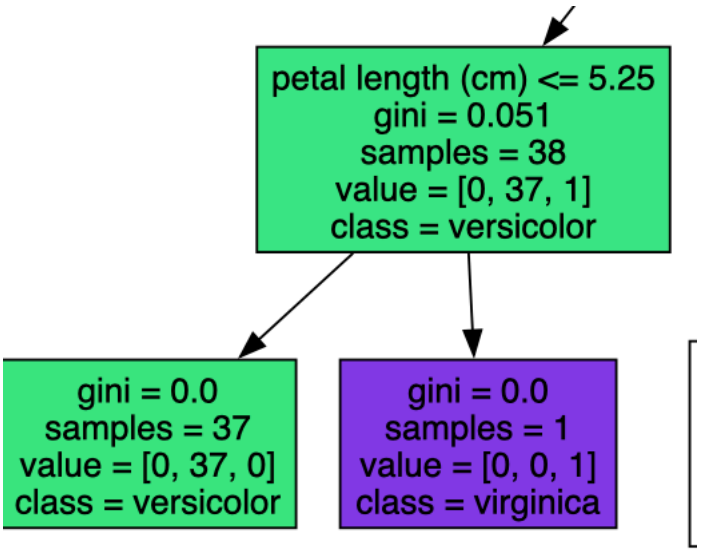
노드 내 기술된 지표 의미 알아보기

petal length <= 2.45 ... 같은 피처 조건: 자식 노드 만들기 위한 규칙 조건. 이게 없으면 리프 노드.

gini: value=[]로 주어진 데이터 분포에서의 지니 계수

samples: 현 규칙에 해당하는 데이터의 건수
value = []: 클래스 값 기반의 데이터 건수.

- 붓꽃 데이터셋: 0, 1, 2의 클래스 값 가지고 있었음. 만약 Value = [41, 40, 39] 라면? 순서대로 0번 꽃(Setosa) 41개, 1번 꽃 40개, 2번 꽃 39개로 구성되어 있는 데이터라는 의미!
- 각 노드의 색은 붓꽃 데이터의 '레이블 값'을 의미한다. (주황은 0, 초록은 1, 보라는 2) 색깔이 **짙을수록** 지니 계수 낮고 해당 레이블에 속하는 샘플 데이터가 **많다**는 뜻이다.



이 노드(위에 있는 초록색 노드): 38개 샘플 중 1개가 Virginica고 나머지는 Versicolor이지만 이를 구분하기 위해 한번 더 자식 노드를 생성해야 한다.
=> 이렇게 결정트리는 규칙 생성 로직을 미리 제어하지 않으면 클래스 값을 완벽하게 구현하기 위해 계속해서 트리 노드를 만들어가고, 이는 모델이 과적합되는 문제를 유발한다.

하이퍼 파라미터: 복잡한 트리 생성 방지를 위한 용도.
앞서 본 max_depth, min_samples_split, min_samples_leaf 같은 하이퍼 파라미터 변경으로 결정 트리를 변화시킬 수 있다.
노드가 분할되기 위한 조건을 설정함으로써, 노드가 분할될 경우 자식 노드들이 그 조건을 정말 만족하는지를 확인. 분할되기 어렵게 조건을 설정하면 자연스럽게 분할 횟수가 줄어들며 트리가 간결해진다.

feature*importances*

결정 트리: 균일도에 기반해 어떤 속성을 규칙 조건으로 선택하는지가 중요!
사이킷런: 피처의 중요한 역할 지표를 DTC 객체의 feature*importances* 속성으로 제공 - ndarray 형태로 값 반환하며 피처 순서대로 값을 할당함

- 피처가 트리 분할 시 정보이득/지니계수를 얼마나 효율적으로 잘 개선시켰는지 정규화된 값으로 표현한 것.
- 일반적으로 값이 높을수록 피처의 중요도가 높다.

결정 트리는 알고리즘이 직관적이어서 규칙 트리의 시각화와 이런 중요도 속성을 통해 알고리즘의 동작 방식을 직관적으로 이해할 수 있다.

결정 트리 과적합, 결정 트리 실습

해당 부분은 코드에 첨부한 주석과 함께 개념을 정리하였다.

앙상블 학습

앙상블 학습 개요

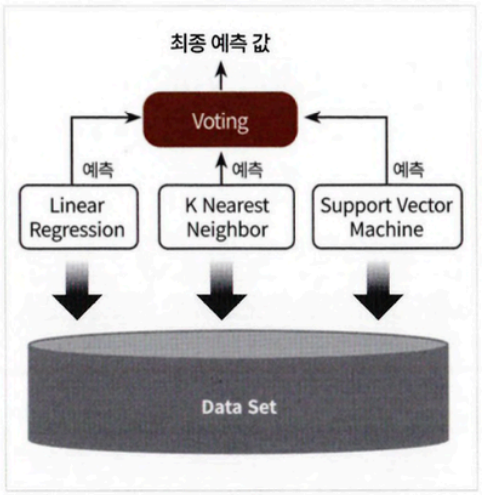
앙상블 학습을 통한 분류: 여러 개의 분류기(Classifier)을 생성하고, 그 예측을 **결합**함으로써 정확한 최종 예측을 도출하는 기법

- 정형 데이터 분류에 뛰어난 성능을 나타냄

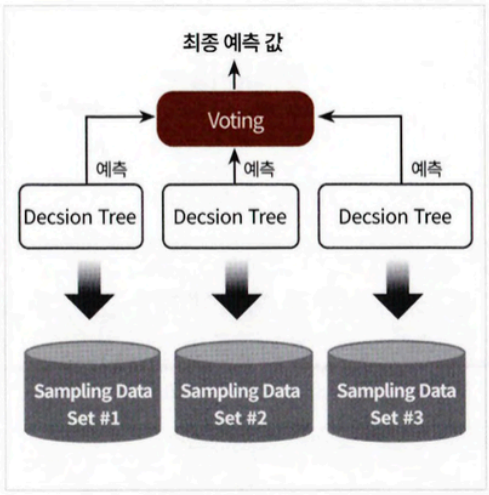
- 대표적인 앙상블 알고리즘: **Random forest, Gradient Boosting**
  - 쉬운 사용, 뛰어난 성능, 다양한 활용도
    - 부스팅 계열의 앙상블 알고리즘 인기 - 새로운 알고리즘 개발 가속화
    - XGBoost, LightGBM, Stacking 등... : 정형 데이터 분류나 회귀 분야에서 예측 성능 뛰어난 모델 개발 가능

앙상블 학습의 유형

- 보팅 Voting: 여러 개의 분류기가 투표를 통해 예측 결과 결정. **서로 다른 알고리즘**을 가진 분류기를 결합.
- 배깅 Bagging: 기본 방식은 보팅과 동일. 각각의 분류기가 **같은 유형의 알고리즘**을 기반으로 하지만 **데이터 샘플링을 다르게** 가져가며 학습을 수행해 보팅 수행 (대표: 랜덤 포레스트 알고리즘)
- 부스팅 Boosting
  등...



Voting 방식



Bagging 방식

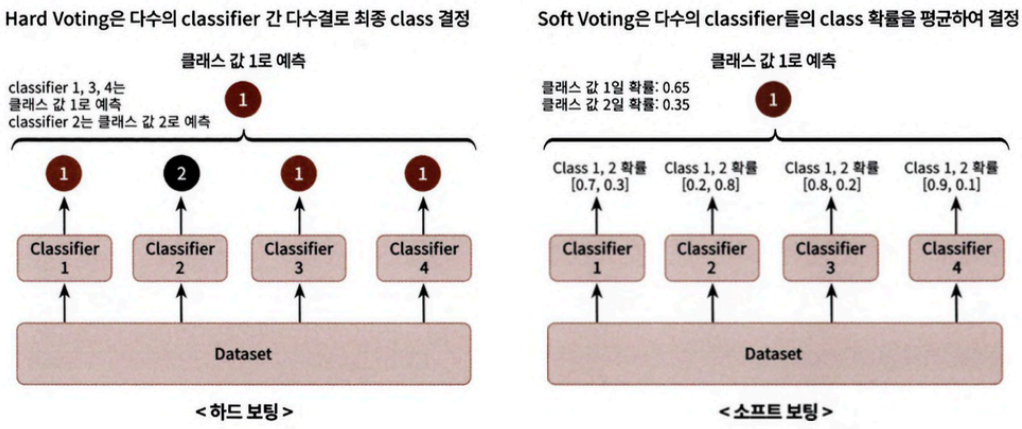
좌: 보팅 분류기 도식화. 3개의 다른 머신러닝 알고리즘이 동일한 데이터셋에 대해 학습하고 예측한 결과를 가지고 보팅을 통해 최종 예측 결과를 선정함.
우: 배깅 분류기 도식화. 단일 머신러닝 알고리즘으로 여러 분류기가 학습으로 개별 예측함.

- 개별 분류기에 할당된 학습 데이터는 원본 학습 데이터를 샘플링해 추출 - 이렇게 개별 분류기에 데이터를 샘플링해 추출하는 방식을 **부트스트래핑 bootstrapping 분할 방식** 이라고 부름.
- 개별 분류기가 이 방식으로 샘플링된 데이터셋에 대해 학습을 통해 개별 예측을 수행한 결과를 보팅을 통해 최종 예측 결과를 선정하는 방식이 배깅 앙상블 방식.

부스팅: 여러 개의 분류기가 순차적으로 학습 수행, **앞에서 예측이 틀린 데이터에 대해서는 다음 분류기에 가중치를 부여하면서 학습과 예측을 진행하는 것** (계속해서 가중치를 부스팅하며 학습을 진행한다!) 예측 성능이 뛰어나 앙상블 학습을 주도한다.

보팅 유형: 하드 보팅, 소프트 보팅

보팅 방법은 하드 보팅, 소프트 보팅 두 가지가 있다.



일반적으로는 **소프트 보팅** 방식이 예측 성능이 좋아서 더 많이 사용된다.

보팅 분류기

`VotingClassifier` : 사이킷런에서 제공하는 보팅 방식의 앙상블 구현 클래스

- 주요 생성 인자로 estimators와 voting 값을 입력받음
  - estimators: 리스트 값. 보팅에 사용될 여러 개의 classifier 객체를 튜플 형태로 입력받음
    - voting: hard / soft (기본은 하드)

```
# 로지스틱 회귀, KNN을 기반으로 소프트 보팅 방식으로 새롭게 보팅 분류기 생성

# 개별 모델: 로지스틱 회귀, KNN
lr_clf = LogisticRegression(solver='liblinear')
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)))
```

```
Voting 분류기 정확도: 0.9561
LogisticRegression 정확도: 0.9474
KNeighborsClassifier 정확도: 0.9386
```

KNN, 로지스틱 회귀 기반의 분류기와 보팅 분류기를 비교했을 때 보팅 분류기의 정확도가 더 높다. 하지만

무조건 보팅으로 여러 개의 분류기를 결합한다고 예측 성능이 향상되는 것은 아니다! 데이터의 특성과 분포 등 요건에 따라 성능이 달라질 수 있다.

정리

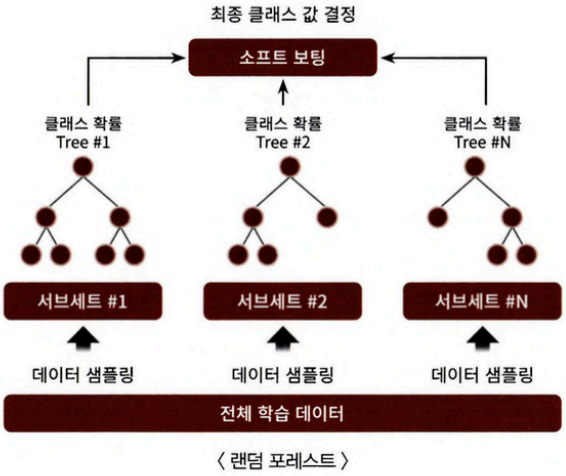
이렇게 ML 모델의 성능은 여러 테스트 데이터에 의해 검증되므로 어떻게 유연하게 대처하는지가 모델의 평가요소가 된다.
결정 트리 알고리즘은 쉽고 직관적인 분류 기준을 가지지만 정확한 예측을 위해 데이터를 계속 나누다 오히려 과적합이 발생해 예측 성능이 떨어지는 현상이 발생하기 쉽다.
앙상블 학습에서는 이런 결정 트리 알고리즘의 한계를 매우 많은 분류기를 결합해, 다양한 상황을 학습하게 함으로써 극복하고 있다.

랜덤 포레스트

랜덤 포레스트의 개요 및 실습

앞서 앙상블 학습 유형 설명에서 배깅의 대표적인 알고리즘 중 하나로 랜덤 포레스트를 소개했다. (배깅: 같은 알고리즘으로 여러 분류기를 만들어 보팅으로 최종 결정하는 알고리즘)

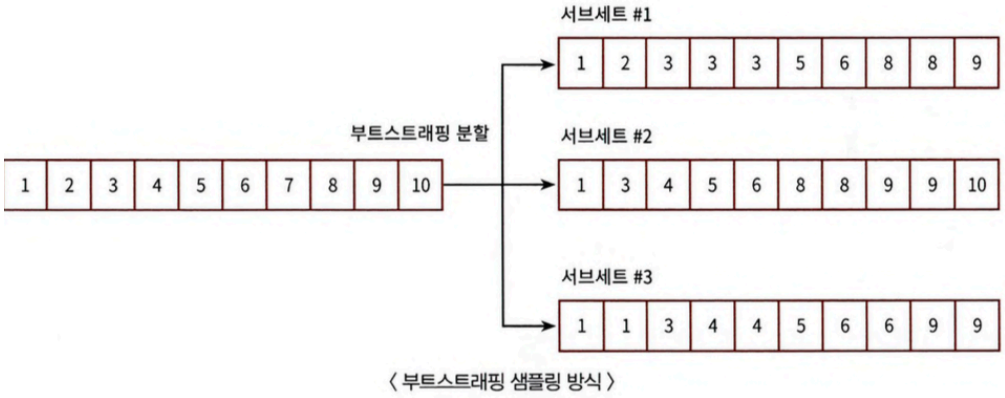
- 빠른 수행 속도
- 여러 영역에서 높은 성능 예측
- 기반 알고리즘: 결정 트리 (쉽고 직관적)



〈 랜덤 포레스트 〉

> **여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별 학습을 수행한 뒤, 최종적으로 모든 분류기가 보팅을 통해 예측 결정**

랜덤 포레스트는 개별 분류기의 기반 알고리즘은 결정 트리지만, 개별 트리가 학습하는 데이터셋은 전체 데이터에서 일부가 중첩되게 샘플링된 데이터셋이다. 이렇게 여러 개의 데이터셋이 중첩되도록 분리하는 것을 '부트스트래핑' 분할 방식이라고 한다.
랜덤 포레스트의 subset 데이터는 이런 부트스트래핑으로 데이터가 임의 생성된다.



〈 부트스트래핑 샘플링 방식 〉

위와 같이 **데이터가 중첩된 개별 데이터셋에 결정 트리 분류기를 각각 적용** 하는 게 랜덤 포레스트이다.

`RandomForestClassifier` : 사이킷런이 지원하는 랜덤 포레스트 기반 분류 클래스

```
# 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))
```

랜덤 포레스트 하이퍼 파라미터 및 튜닝

파라미터

> 1. n_estimators: 결정 트리의 개수 지정. 디폴트 10개. 많이 설정할수록 성능이 좋아지지만 계속 증가시킨다고 무조건 향상되는 건 아니고, 늘릴수록 학습 수행 시간이 늘어난다.
> 2. max_features: 결정 트리에서 쓰인 것과 동일. 하지만 여기서 디폴트는 auto (결정트리는 None)
> 3. max_depth
> 4. min_samples_leaf
> 5. min_samples_split
>
> (3~5는 결정트리에서의 사용 방식과 동일하다.)

goblurry

이전 포스트
3. 평가

0개의 댓글

댓글을 작성하세요

댓글 작성