|  |  |
| --- | --- |
| 교육 제목 | **머신러닝 Random Forest** |
| 교육 일시 | 2021년 11월 23일 화요일 |
| 교육 장소 | 영우글로벌러닝 2층 |
| **교육 내용** | |
| 오전 | 1. 정형 데이터와 비정형 데이터 & 앙상블 학습    1. <정형 데이터와 비정형 데이터>       1. - 정형 데이터 : 구조가 정해져 있고 가지런히 정리된 데이터       2. - 비정형 데이터 : 데이터베이스나 엑셀로 표현하기 어려운 것들. 텍스트, 사진, 음성파일등이 해당된다. 2. <앙상블 학습>    1. - 다양한 분류 알고리즘(주로 결정 트리를 기반)을 합쳐서 학습을 진행하는 방법    2. - 랜덤 포레스트, 엑스트라 트리, 그레이디언트 부스팅 등이 있다. 3. 랜덤 포레스트    1. - 결정 트리를 랜덤하게 만들어 결정 트리의 숲을 만든다. 그리고 각 결정 트리의 예측을 사용해 최종 예측을 만든다.    2. - 랜덤 포레스트는 각 트리를 훈련하기 위한 데이터를 랜덤하게 만드는게 이때 부트스트랩 샘플을 사용한다. 4. <부트스트랩 샘플>    1. - 전체 세트에서 뽑았던 샘플을 다시 넣어서 복원 추출을 하게 하는 것. 이 경우 중복이 적용된다.    2. - 보통 부트 스트랩 방식이라고 한다.    3. - 또한 각 노드를 분할할때 전체 특성 중 일부 특성을 무작위로 고른 다음 이 중에서 최선의 분할을 찾는다.    4. - 분류 모델인 RandomForestClassifier는 기본적으로 전체 특성 개수의 제곱근 만큼 특성을 선택한다.       1. <실습 코드>       2. from sklearn.model\_selection import cross\_validate       3. from sklearn.ensemble import RandomForestClassifier       4. rf = RandomForestClassifier(n\_jobs=-1, random\_state=42)       5. scores = cross\_validate(rf, train\_input, train\_target, return\_train\_score=True, n\_jobs=-1)       6. print(np.mean(scores['train\_score']), np.mean(scores['test\_score']))       7. - 코드의 사용방법은 다른 학습모델과 큰 차이가 없다.       8. - 랜덤 포레스트는 결정 트리의 앙상블이기 때문에 특성 중요도를 계산해서 출력할 수 있다. 5. # 특성 중요도 계산    * 1. rf.fit(train\_input, train\_target)      2. print(rf.feature\_importances\_)    1. - RandomForestClassifier에는 자체적으로 모델을 평가하는 점수를 얻을 수 있다.    2. - 부트스트래핑을 할때,부트스트랩 샘플에 포함되지 않고 남은 샘플(=OOB샘플)로 훈련한 결정트리를 평가할 수 있다.(이때 OOB샘플은 검증 세트의 역할을 하는 셈이다) 이 점수를 얻으려면 매개변수 oob\_score를 True로 지정해야 한다.       1. rf = RandomForestClassifier(oob\_score=True, n\_jobs=-1, random\_state=42)       2. rf.fit(train\_input, train\_target)       3. print(rf.oob\_score\_) |
| 오후 | 1. 엑스트라 트리    1. - 랜덤 포레스트와 비슷하게 동작하지만, 엑스트라 트리는 부트스트랩 샘플을 사용하지 않고 결정트리를 만들때 전체 훈련 세트를 사용한다.    2. - 대신 노드를 분할할때 무작위로 분할하게 된다(spliter=random)    3. - 성능은 낮아질 수 있지만 많은 트리를 앙상블 하기 때문에 과대적합을 막고 검증세트의 점수를 높이는 효과가 있다.    4. - 사이킷런에서 제공하는 엑스트라 트리는 ExtraTreesClassifier이다.    5. - 그 외 매개변수는 랜덤포레스트와 거의 동일하다. 2. <실습 코드>    1. from sklearn.ensemble import ExtraTreesClassifier    2. et = ExtraTreesClassifier(n\_jobs=-1, random\_state=42)    3. scores = cross\_validate(et, train\_input, train\_target, return\_train\_score=True, n\_jobs=-1)    4. print(np.mean(scores['train\_score']), np.mean(scores['test\_score'])) 3. 그레이디언트 부스팅(Gradient Boosting)    1. - 깊이가 얕은 결정 트리를 사용하여 이진트리의 오차를 보완하는 방식으로 앙상블 하는 방법    2. - 경사하강법을 사용하여 트리를 앙상블에 추가하며, 분류에서는 로지스틱 손실 함수를, 회귀에서는 평균 제곱 오차 함수를 사용한다.    3. - 일반적으로 그레이디언트 부스팅이 랜덤 포레스트보다 조금 더 높은 성능을 얻을 수 있다. 하지만 순서대로 트리를 추가하기 때문에 훈련 속도가 느리다.(그래서 GradientBoostingClassifier에는 n\_jobs가 없다.)    4. - 사이킷런에서 제공하는 그레이디언트 부스팅 클래스는 GradientBoostingClassifier이다.       1. from sklearn.ensemble import GradientBoostingClassifier       2. gb = GradientBoostingClassifier(random\_state=42)       3. scores = cross\_validate(gb, train\_input, train\_target, return\_train\_score=True, n\_jobs=-1)       4. print(np.mean(scores['train\_score']), np.mean(scores['test\_score'])) 4. - 결정 트리에는 아래와 같은 매개변수들이 있다.    1. n\_estimators : 사용할 결정 트리의 개수, 기본값은 100이다.    2. learning\_rate: 학습률, 기본값은 0.1이다. 5. 히스토그램 기반 그레이디언트 부스팅    1. - 입력 특성을 256개의 구간으로 나누고, 그 구간중에서 하나를 떼어놓고 누락된 값을 위해서 사용한다.    2. - 사이킷런에서 제공하는 히스토그램 기반 그레이디언트 부스팅 클래스는 HistGradientBoostingClassifier이다.       1. from sklearn.experimental import enable\_hist\_gradient\_boosting       2. from sklearn.ensemble import HistGradientBoostingClassifier       3. hgb = HistGradientBoostingClassifier(random\_state=42)       4. scores = cross\_validate(hgb, train\_input, train\_target, return\_train\_score=True, n\_jobs=-1)       5. print(np.mean(scores['train\_score']), np.mean(scores['test\_score']))    3. - 히스토그램 기반 그레이디언트 부스팅의 회귀버전은 HistGradientBoostingRegressor에 구현되어 있다. 6. - 사이킷런 말고도 히스토그램 기반 그레이디언트 부스팅 알고리즘을 구현한 라이브러리로 XGBoost, LightGBM등이 있다. 7. <XGBoost>    * 1. from xgboost import XGBClassifier      2. xgb = XGBClassifier(tree\_method='hist', random\_state=42)      3. scores = cross\_validate(xgb, train\_input, train\_target, return\_train\_score=True, n\_jobs=-1)      4. print(np.mean(scores['train\_score']), np.mean(scores['test\_score']))    1. - 여러개의 Decision Tree를 조합해서 사용하는 Ensemble 알고리즘    2. - XGBoost에서 tree\_method를 hist로 지정하면 히스토그램 기반 그레이디언트 부스팅 알고리즘을 쓸 수 있다. 8. <LightGBM>    * 1. from lightgbm import LGBMClassifier      2. lgb = LGBMClassifier(random\_state=42)      3. scores = cross\_validate(lgb, train\_input, train\_target, return\_train\_score=True, n\_jobs=-1)      4. print(np.mean(scores['train\_score']), np.mean(scores['test\_score']))    1. - LightGBM의 경우에는 최대 손실값을 가지는 노드를 중심으로 계속해서 분할하는 '리프 중심 트리 분할(leaf-wise)' 방식을 사용한다.    2. - 따라서 트리가 깊어지기위해 소요되는 시간과 메모리를 많이 절약할 수 있다.    3. - 다만 적은 데이터에 대한 과적합(overfitting)이 발생하기 쉽다 |