마세마타 제??권(2023년)
단국대학교 수학과 학술제 발표논문
Mathematics of Dankook University
Vol.?? December 2023 ??~??

천체 분류를 위한 머신러닝 기법의 성능 평가 연구

단국대학교 수학과
32191235 김택균[1], 32191468 맹지주[2]

요 약 : 본 연구는 인공지능과 관련된 과제를 제시해주고 참가자가 이를 해결하는 데이콘(DACON)이 주최한 대회인 '월간 데이콘 천체 유형 분류 대회' 문제를 바탕으로 데이터 전처리와 모델 구축 및 평가를 하여 다양한 각도에서 문제를 바라보고자 한다. 특히, AutoML 라이브러리인 PyCaret를 이용하여 각 모델의 정확도를 평가하고 모델들을 앙상블하여 다양한 모델들을 비교 분석하고, 주요 모델의 선택과 튜닝에 대한 통찰을 얻고자 한다. 본 연구가 수학 및 데이터 사이언스에 관심 있는 학생들과 입문자들에게 많은 도움이 되었으면 한다.

지도교수 : 김 도 형
1) rlatg0123@naver.com
2) mjiju06@naver.com

1. 서론

머신러닝 알고리즘은 데이터를 기반으로 통계적인 신뢰도를 강화하고 예측 오류를 최소화하기 위한 다양한 수학적 기법을 적용해 데이터 내의 패턴을 스스로 인지하고 신뢰도 있는 예측 결과를 도출해 낸다. 이러한 머신러닝에는 크게 지도학습과 비지도학습으로 나뉜다. 지도학습에는 대표적으로 분류와 회귀가 있는데, 본 연구는 분류 모델을 활용하여 천체의 유형을 구분(분류)하는 알고리즘 모델을 만들 계획이다. 분류 모델이란, 기존에 존재하는 데이터의 카테고리를 파악하고, 새롭게 관측된 데이터의 카테고리를 스스로 판별하는 과정을 말한다. 분류 모델은 입력 데이터를 미리 정의된 클래스 또는 범주로 할당하는 작업에 사용되며, 입력 데이터를 여러 클래스 중 하나로 분류하는 것이 목표이다. 이진 분류를 통해 참-거짓을 판별할 수도 있지만, 다중 분류를 통해 3개의 이상의 값을 판별할 수 있다. 위 분류 모델을 활용하기 위해 '데이콘'이라는 인공지능 경진대회 플랫폼의 주제와 데이터셋을 활용하기로 하였다.

‘데이콘'은 온라인 데이터 사이언스 경진 대회와 교육을 함께 진행할 수 있는 플랫폼으로 다양한 기업의 데이터를 받아 사용자들에게 이를 학습 등 활용할 수 있는 공간을 제공하고 그곳에서 나온 좋은 프로그램을 다시 데이터 제공 기관에게 전달하여 기술 개발의 단초를 제공해 준다.

본 연구에서는 데이콘에서 제공한 '천체 유형 분류' 데이터를 사용하여 분류 모델에 적용시켰다. '천체 유형 분류' 데이터를 통해서 천체의 유형을 분류함으로써 우주의 현상과 구조를 밝혀내는 데 도움을 줄 수 있고, 우주의 다양한 현상을 이해하는 데 도움이 될 수 있다. 본 연구에서는 Python의 AutoML 라이브러리인 Pycaret를 사용한다. 천체 관측을 통해 측정된 21개의 데이터를 이용하여 이미 정의된 19개의 천체 유형을 분류하는 알고리즘을 개발한다.

2. 지도학습에서 분류

데이터 분석 모델은 학습 유형에 따라 지도학습과 비지도학습, 준지도학습, 강화학습으로 분류된다. 지도학습이란 훈련 데이터로부터 하나의 함수를 유추해 내기 위한 기계 학습의 한 방법으로 예시를 통해 학습하도록 설계된다. 즉, 간단히 말해 선생님이 문제를 내고 그다음 바로 정답까지 같이 알려주는 방식의 학습 방법인 것이다. 정답이 있는 데이터의 학습을 의미한다. 지도학습에서 분류(Classifier)는 기존에 존재하는 데이터 안에 있는 카테고리 간의 관계를 파악하고, **새롭게 관측된 데이터의 카테고리를 스스로 판별하는 과정**이다. 이러한 분류를 하기 위한 모델을 분류모델이라고 한다.

분류에서 [3]cost function은 회귀모델에서와 마찬가지로 분류모델이 얼마나 잘 예측하는지를 알 수 있어야 하는데 그 정도를 측정할 수 있는 함수이다. 하지만 회귀모델에서의 cost function의 값은 따로 제한이 없지만 [4]분류모델에서는 그 값이 0과 1사이의 값으로 나와야 한다. 따라서 이러한 결과 값을 나오게 하기 위해서 활성화 함수(activation function)를 활용한다. 신경망에서는 노드에 들어오는 값들에 대해 곧바로 다음 레이어로 전달하지 않고 활성화 함수를 통과시킨 후 전달한다. 활성화 함수는 입력 신호의 총합을 출력 신호로 변환하는 함수로, 입력받은 신호를 얼마나 출력할지 결정하고 네트워크에 층을 쌓아 비선형성을 표현할 수 있도록 해준다.

---

3) cost function 이란 머신러닝 또는 인공지능 알고리즘에서 가설함수가 얼마나 정확한지 판단하는 함수로 다음과 같이 작성한다.

$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$ 이며, 이때 $H(x) = Wx + b$를 뜻한다.

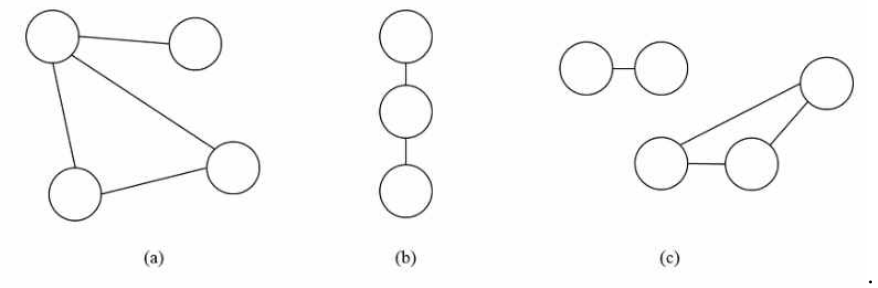
4) 로지스틱 회귀분석을 활용한 분류(classification) 문제에서는 목표변수를 직접 예측(prediction)하는 것이 아닌 2개의 값('성공(1)' or '실패(0)') 중 하나의 값으로 예측할 때 사용된다.

3. 사용된 알고리즘

그래프[5)]

앞으로 설명할 모델들은 모두 트리 구조를 가진다는 공통점이 있다. 그리고 이러한 트리 구조는 이산수학의 그래프 이론에서도 중요한 부분이기에 이에 대한 설명이 필요하다.

트리의 개념을 이해하기 위해서는 먼저 이산 수학(discrete mathematics)의 그래프(graph)를 이해할 필요가 있다. 그래프 G는 G=(V,E)와 같이 표현되며, V는 그래프의 노드를 나타내는 집합이고, E는 그래프의 간선을 나타내는 집합이다. 이때의 노드는 그래프에서 데이터를 저장하는 기본 단위로 Vertex라고도 표현하며, 그래프 내의 개별 항목을 나타낸다. 또한 간선은 노드 간의 관계를 나타내고 Edge라고 표현하며 두 노드를 연결하는 선을 의미한다. 아래의 [그림1]은 그래프에 대한 몇 가지 예시를 나타낸다.

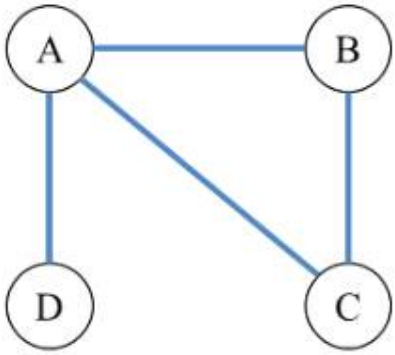


[그림 1]

이러한 그래프에는 Walk와 Path 그리고 Cycle이라는 개념이 존재한다. Walk는 그래프 내에서 Vertex와 Edge를 번갈아 가면서 순회하는 경로를 의미하고, Path는 Edge가 반복되지 않는 Walk를 의미하며, Cycle은 경로의 시작 Vertex와 마지막 Vertex가 같은 Path를 의미한다. 아래의 [그림2]를 통해서 예를 들어보겠다.

---

5) [출처] https://untitledtblog.tistory.com/85



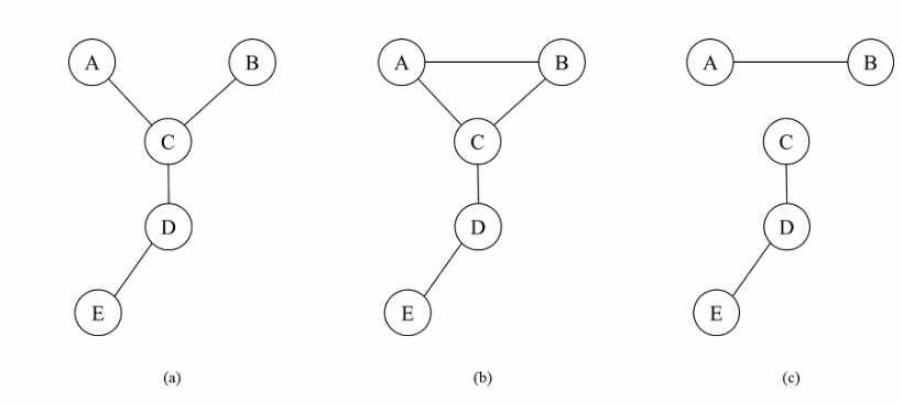
[그림 2]

만약 [그림2]의 경로가 A-B-C-A라고 한다면 이는 Vertex와 Edge를 번갈아 가면서 순회하는 경로이기에 Walk이고, 반복되는 Edge 없이 Walk의 조건을 만족했으므로 Path의 조건도 만족한다. 또한 경로의 시작 Vertex와 마지막 Vertex가 모두 A로 같기에 Cycle의 조건 또한 만족한다.

그렇다면 경로가 A-B-C-A-B라면 어떨까? 시작 Vertex는 A이고 마지막 Vertex는 B이므로 서로 달라서 이는 Cycle이 아니고 A-B의 경로에서 반복되는 Edge가 생겼으므로 이는 Path라고도 할 수 없는 것이다.

트리

이산 수학 또는 그래프 이론에서 트리는 어떤 두 Vertex가 정확히 하나의 유일한 Path로만 연결된 그래프를 의미한다. 두 Vertex가 유일한 Path로만 연결되어 있다는 것은 그래프에 Cycle이 존재하지 않는다는 것과 동치이다. 아래 [그림3]을 통해 예를 들겠다.



[그림 3]

[그림3]에서 (b)는 A에서 C로 이동하는 path가 2개(A-B-C, A-C) 존재하므로 어떤 두 Vertex가 정확히 하나의 유일한 Path로만 연결되어야 한다는 조건에 맞지 않아서 트리가 아니다. 또는 A-B-C-A로 이루어진 Cycle이 존재하므로 트리가 아니라고도 할 수 있다.

(c)는 Cycle이 존재하지는 않지만, A 또는 B에서 C로 이동하는 path가 존재하지 않는다. 이또한 어떤 두 Vertex 사이에 하나의 유일한 Path가 존재해야한다는 조건에 맞지 않으므로 (c) 또한 트리가 아니다.

마지막으로 (a)는 어떤 두 Vertex가 정확히 하나의 유일한 Path로만 연결된 그래프라는 조건에 완벽하게 부합하므로 트리라고 할 수 있다.

본 연구에서는 위에서 설명한 트리 구조를 기반으로 하는 다음의 5가지 알고리즘을 사용한다.

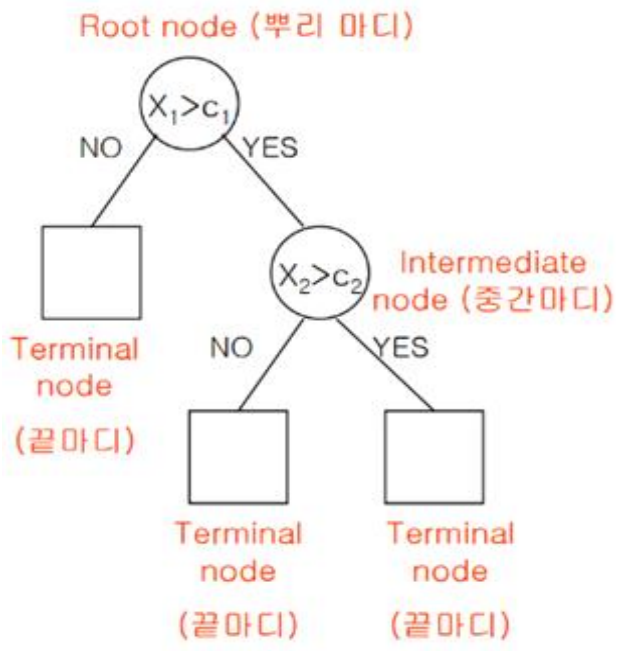
3.1 랜덤 포레스트 (Random Forest)

랜덤 포레스트(RandomForest)는 여러 개의 결정트리(Decision Tree)를 활용한

배깅방식의 대표적인 알고리즘이다. 여기서 결정트리(Decision Tree)와 배깅(bagging)에 대해서 설명이 필요하다.

3.1.1 결정 트리 (Decision Tree)

특정 기준(질문)에 따라 데이터를 구분하는 모델을 결정 트리 모델이라고 한다. 결정 트리에서 질문이나 정답을 담은 네모 상자를 노드(Node)라고 하고, 맨 처음 분류 기준 (즉, 첫 질문)을 Root Node, 맨 마지막 노드를 Terminal Node 혹은 Leaf Node라고 한다.



[그림 4]

위의 [그림 4]을 보면 이해가 더 쉬울 것이다.[6)]

---

6)[출처]
https://velog.io/@yuns_u/%EA%B2%B0%EC%A0%95%ED%8A%B8%EB%A6%AC%EC%9D%98%EC%82%AC%EA%B2%B0%EC%A0%95%EB%82%98%EB%AC%B4-Decision-Trees

Root node에서 질문을 시작하여 질문이 참이면 YES를 통해서, 거짓이면 NO를 통해서 다음 노드로 데이터를 보내는 방식이다.

3.1.2 배깅 (Bagging)

Bagging은 Bootstrap Aggregation의 약자로 복원추출을 통해서 샘플을 여러 번 랜덤하게 뽑아서 뽑은 모델들을 각각 학습시켜 그 결과물을 집계하는 방법이다.

배깅은 아래와 같은 방식으로 진행된다.

(1) 동일한 알고리즘을 사용하는 일정 수의 분류기 생성

(2) 각각의 분류기는 부트스트래핑(Bootstrapping)방식으로 생성된 샘플 데이터를 학습

(3) 최종적으로 모든 분류기의 결과를 집계해서 예측값 결정

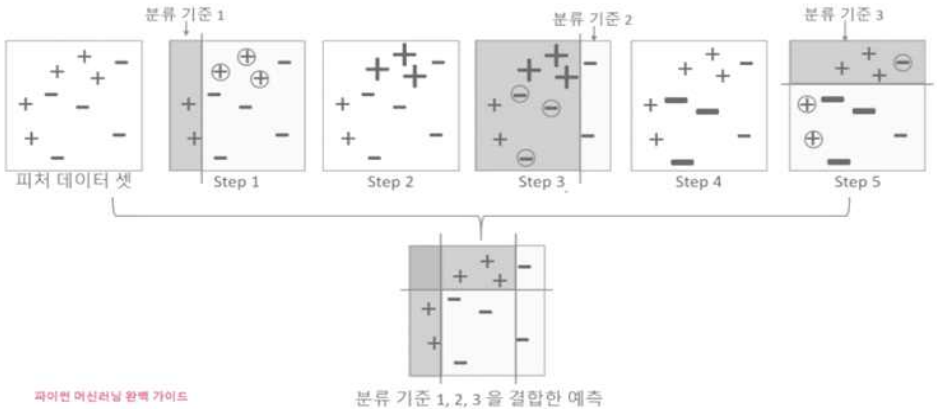
※ 부트스트래핑 샘플링은 전체 데이터에서 일부 데이터의 중첩을 허용하여 랜덤하게 샘플을 뽑는 방법이다.

따라서 랜덤포레스트는 여러 개의 결정트리를 만들고 그 결정트리들을 부트스트랩 방식으로 복원 추출하여 여러 개의 샘플을 만들고 만든 샘플들 각각의 예측값을 알아보고 그 값들의 결과를 집계해서 최종 예측값을 결정하는 모델인 것이다.

이러한 RandomForest는 쉽고 직관적이며 비교적 빠른 수행속도를 가지고 다양한 분야에서 좋은 성능을 나타낸다는 장점이 있지만 수많은 의사결정나무를 만들어야하기 때문에 학습 시간과 연산이 많이 든다는 단점 또한 존재한다.

3.2 그레디언트 부스팅 (Gradient Boosting)

부스팅 알고리즘은 여러 개의 모델을 순차적으로 학습, 예측하며 잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선해 나가면서 학습하는 방식이다. 그리고 이러한 부스팅의 대표적인 예시로는 그래디언트 부스트(Gradient Boost)가 있다.



[그림 5]

위 [그림 5][7]처럼 "+"와 "-"로 구성된 피쳐 dataset이 있다면 step 1에서 일단 분류를 하고 step 2에선 잘못 분류한 data에 가중치를 부여한다. 이 과정을 step 5까지 반복하면 총 3개의 분류기(분류기준 1,2,3)를 통해 분류를 하게 되고 1개의 분류기를 사용할 때보다 정확도가 높아진 것을 알 수 있다.

GBC(Gradient Boosting Classifier)는 이러한 알고리즘을 가지는데 가중치 업데이트를 경사하강법(Gradient Descent)[8]을 이용하는 것이 가장 큰 특징이다. 오류값은 (실제값-예측값)이 되고 이 오류 식을 최소화하는 방향으로 반복적으로 가중치를 업데이트한다. gradient descent는 말 그대로 보면 gradient = 기울기, descent = 하강이라는 두 단어가 합쳐진 단어이다. 따라서 gradient descent는 주어진 함수에서 극소점을 찾기 위해 기울기가 최소가 되는 지점을 찾아가는 방식이다.

---

7) https://velog.io/@sset2323/04-05.-GBMGradient-Boosting-Machine
8) 경사 하강법은 1차 근삿값 발견용 최적화 알고리즘이다. 기본 개념은 함수의 기울기(경사)를 구하고 경사의 반대 방향으로 계속 이동시켜 극값에 이를 때까지 반복시키는 것이다.

3.3 CatBoost

CatBoost는 이름에서부터 알 수 있듯이 Boosting(부스팅) 기반의 모델이다. Boosting은 Gradient Boosting Classifier에서도 설명했듯이 여러 개의 모델을 순차적으로 학습, 예측하며 잘못 예측한 데이터에 가중치 부여를 통해 오류를 개선해 나가면서 학습하는 방식이다

대부분 boosting 기반 알고리즘들은 데이터를 가지고 모델이 학습을 한 후 예측을 하기 위해서 범주형으로 된 모든 데이터를 일련의 과정을 통해 숫자형으로 바꿔준다. 반면 CatBoost에서는 다른 boosting 기반 알고리즘과 달리 범주형 변수를 특별하게 처리한다. 범주형 변수를 One-hot Encoding, Label Encoding 등 범주형을 숫자형으로 바꿔주는 encoding 작업을 하지 않고도 범주형 변수를 그대로 모델에 넣어주면 CatBoost가 알아서 Orderd Target Encoding를 진행하여 모델이 학습을 하고 예측까지 할 수 있다는 장점이 있다. 이때 Orderd Target Encoding은 범주형 변수를 인코딩시킬 때, target의 값을 고려한 숫자를 인코딩하는데 이는 현재 데이터의 인코딩하기 전의 데이터들의 타겟값을 이용하여 인코딩을 해서 데이터의 누수와 오버피팅을 막을 수 있는 방식이다.

또한 CatBoost에서는 일반적인 Boosting 대신 Orderd Boosting을 사용한다. Ordered Boosting에서는 데이터셋이 무작위로 섞이고, 각 학습 단계에서는 이전 단계까지의 데이터만 사용하여 모델을 훈련시킨다. 그리고 각 학습 단계에서의 오차는 다음 단계에서 사용되는 데이터에 영향을 미친다. 아래 [그림 6]의[9] 예시를 통해서 더 자세하게 설명해보겠다.

9) https://dailyheumsi.tistory.com/136

time	datapoint	class label
12:00	x1	10
12:01	x2	12
12:02	x3	9
12:03	x4	4
12:04	x5	52
12:05	x6	22
12:06	x7	33
12:07	x8	34
12:08	x9	32
12:09	x10	12

[그림 6]

기존 부스팅 기법은 모든 datapoint(x1-x10)까지의 잔차를 일괄 계산한다. 반면, Ordered Boosting의 과정은 다음과 같다.

1. 먼저 x1의 잔차만 계산하고, 이를 기반으로 모델을 만든다. 그 후, x2의 잔차를 이 모델로 예측한다.

2. x1,x2의 잔차를 가지고 모델을 만든다. 이를 기반으로 x3,x4의 잔차를 모델로 예측한다.

3. x1,x2,x3,x4의 잔차를 가지고 모델을 만든다. 이를 기반으로 x5,x6,x7,x8의 잔차를 모델로 예측한다.

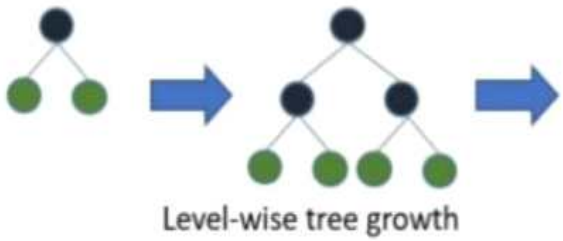
4. 이 과정을 반복한다.

즉 순서(order)에 따라 모델을 학습시키고 순차적으로 잔차를 계산하는 과정을 반복하는 것이다.

3.4 XGBoost (eXtreme Gradient Boosting)

XGBoost는 "Extreme Gradient Boosting"의 약자이다. 이는 여러 개의 Decision Tree를 조합해서 사용하는 앙상블 알고리즘으로 기존의 그래디언트 부스팅 알고리즘의 계산 속도와 모델 성능을 효과적으로 향상시킨 것이 특징이다.

대부분의 트리 기반 알고리즘은 크게 Level-wise 트리 분할과 Leaf-wise 트리 분할로 나뉘게 된다. 이 중에서 XGBoost는 Level-wise 트리 분할 알고리즘을 사용한다.



[그림 7]

일반적인 Gradiant Boosting 모델의 트리 분할은 균형 트리 분할 방법(Level Wise)을 사용한다. 균형 트리 분할은 트리의 깊이를 효과적으로 줄일 수 있다. 즉 최대한 균형 잡힌 트리를 유지하면서 분할하기 때문에 트리의 깊이가 최소화될 수 있다. 균형 잡힌 트리를 생성하는 이유는 overfitting에 보다 더 강한 구조를 가질 수 있다고 알려져 있기 때문이다. **XGBoost의 핵심적인 특징**은 다음과 같다.

1. 기존 Gradient Boosting 알고리즘은 트리의 깊이가 일정 수준 이상이면 가지치기를 하지 않고 최대한 깊게 트리를 만든 반면 XGBoost는 가지치기를 통해 더욱 간결하면서 예측 성능이 우수한 트리 구조를 만들 수 있다. 여기서 말하는 가지치기는 트리의 깊이가 깊어질수록 과적합의 위험성이 높아지는데 이때 불필요한 마디를 제거하는 과정이 바로 가지치기이다.

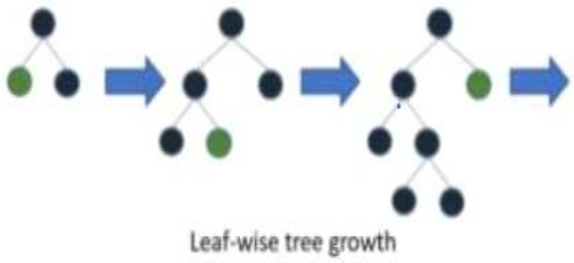
2. XGBoost는 과적합을 방지하기 위해 규제(Regularization)를 적용할 수 있다. 규제는 L1, L2 규제를 지원한다. 이때 L1규제는 모델의 가중치에 대한 절대값의 합을 계산하여 이를 손실 함수에 추가함으로써 모델의 가중치를 0으로 만들어 불필요한 특성을 제거하고 모델을 간단하게 만드는 효과가 있고, L2규제는 모델의 가중치에 대한 제곱의 합을 계산하여 이를 손실 함수에 추가하여 모델의 모든 가중치를 작게 만드는 효과가 있어, 모델이 학습 데이터에 너무 민감하게 반응하는 것을 방지할 수 있다.

이러한 특징으로 인해서 XGBoost는 **과적합을 방지하는데에 적합**해서 이를 통해 모델의 예측 성능이 좋다는 장점이 있는 반면 좋은 성능을 보이기 위해서는 충분히 많은 데이터가 있어야하며 이는 다른 말로 하면 충분치 않은 데이터에서

는 과적합을 보일 수 있다는 이야기이다.

3.5 LightGBM

LightGBM은 XGBoost와 마찬가지로 Decision Tree를 조합해서 사용하는 **앙상블 알고리즘**이다. XGBoost의 경우 매우 뛰어난 부스팅 알고리즘이지만 여전히 학습 시간이 오래 걸린다는 단점이 있는 반면 LightGBM의 가장 큰 장점은 XGBoost보다 학습에 걸리는 시간이 훨씬 적고 메모리 사용량도 상대적으로 적다. LighGBM과 XGBoost의 예측 성능은 별다른 차이가 존재하지 않는다. 또한 기능상의 다양성은 LightGBM이 약간 더 많다. 따라서 LightGBM은 XGBoost의 장점은 계승하고 단점은 보완하는 방식으로 개발되었다. 그럼에도 불구하고 LightGBM의 한 가지 단점으로는 적은 데이터 셋에 적용할 경우 과적합이 발생하기 쉽다는 것이다.



10)

[그림 8]

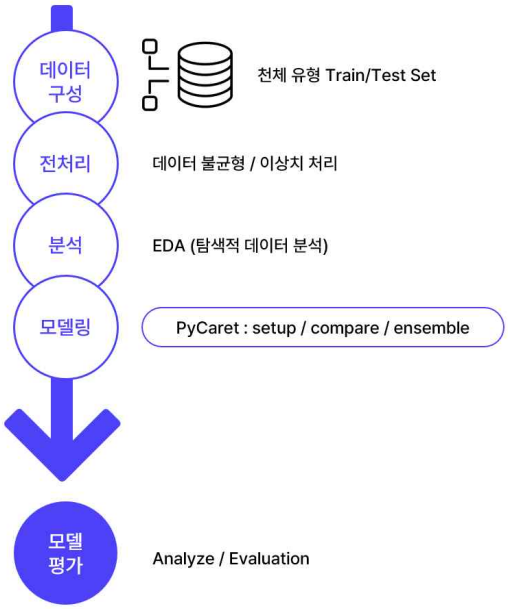
LightGBM은 일반 GBM 계열의 트리 분할 방법과 달리 리프 중심 트리 분할(Leaf Wise) 방식을 사용한다. LightGBM의 리프 중심 트리 분할 방식은 트리의 균형을 맞추지 않고, 최대 손실 값을 가지는 리프 노드를 지속적으로 분할하면서 트리의 깊이가 깊어지고 비대칭적인 규칙 트리가 생성된다. 하지만 이렇게 최대 손실값을 가지는 리프 노드를 지속적으로 분할해 생성된 규칙 트리는 학습을 반복할수록 결국은 균형 트리 분할 방식보다 예측 오류 손실을 최소화 할 수 있다.

---

10) https://m.blog.naver.com/winddori2002/221931868686

4. 본론

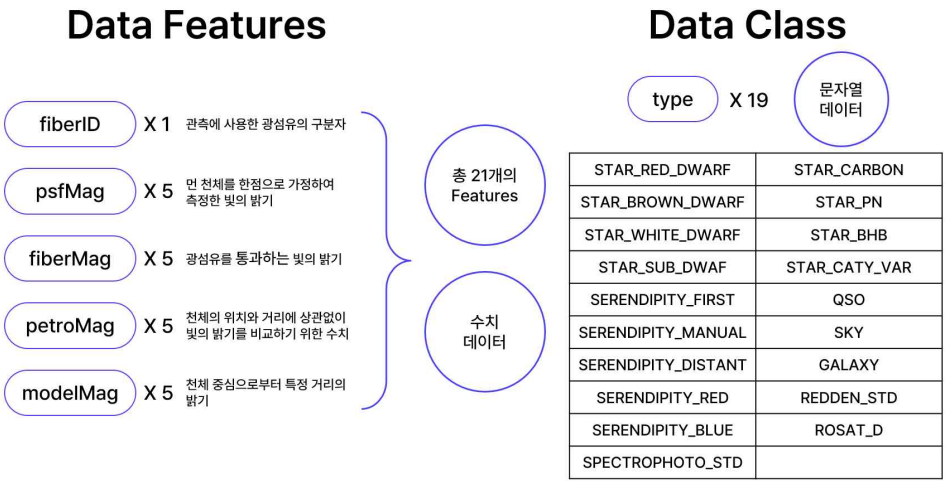
전체적인 연구 흐름은 다음 [그림 9]와 같다.



[그림 9]

4.1 분석 데이터 구성

데이콘이 주최하는 월간 데이콘 천체 유형 분류 대회에서는 fiberID, psfMag, fiberMag 등과 같은 천체를 관측하여 측정된 21개의 데이터를 포함하는 train(훈련)데이터셋과 test(시험)데이터셋을 제공한다.



[그림 10]

여기서 train 데이터셋은 다섯 종류의 천체 관측 데이터와 19개의 천체 유형을 포함하여 총 199991건의 천체 데이터를 제시한다. test 데이터셋은 train 데이터셋에서 천체 유형을 제외한 10009건의 데이터를 제시한다. 본 연구에서는 train 데이터셋을 훈련한 모델에서 찾은 알고리즘을 train 데이터셋에 적용하여 천체 10009개의 유형을 분류, 예측하는 것을 목표로 한다.

	id	type	fiberID	psfMag_u	psfMag_g	psfMag_r	psfMag_i	psfMag_z	fiberMag_u	fiberMag_g	...	petroMag_u	petroMag_g	petroMag_r	petroMag_i	petroMag_z
0	0	QSO	601	23.198224	21.431953	21.314148	21.176553	21.171444	22.581309	21.644453	...	22.504317	21.431636	21.478312	21.145409	20.422446
1	1	QSO	788	21.431355	20.708104	20.678850	20.703420	20.473229	21.868797	21.029773	...	21.360701	20.778968	20.889705	20.639812	20.646660
2	2	QSO	427	17.851451	16.727898	16.679677	16.694640	16.641788	18.171890	17.033098	...	17.867253	16.738784	16.688874	16.744210	16.808006
3	3	QSO	864	20.789900	20.040371	19.926909	19.843840	19.463270	21.039030	20.317165	...	20.433907	19.993727	19.985531	19.750917	19.455117
4	4	STAR_RED_DWARF	612	26.454969	23.058767	21.471406	19.504961	18.389096	25.700632	23.629122	...	25.859229	22.426929	21.673551	19.610012	18.376141

5 rows × 23 columns

[그림 11]

본 대회에서 제공하는 train 데이터는 [그림 11]에서 보는 바와 같이 id와 type, 그리고 21개의 feature들로 이루어져 있다. ‘id’는 데이터셋에서 지정한 천체들의 고유 번호를 의미한다. ‘type’는 문자열 데이터로 우리가 분류하고자 하는 천체의 유형을 의미한다. 이는 타겟변수로, train 데이터에서만 존재하고 test 데이터에는 존재하지 않는다.

다음의 21개의 feature들은 예측변수이다. ‘fiberID’는 관측에 사용된 광섬유의 구분자로, 천체의 위치를 정확하게 파악하기 위해 사용된 광섬유를 구분하는 식별자이다. ‘psfMag’는 point spread function magnitudes의 약자로, 먼 천체를 한 점으로 가정하여 측정한 빛의 밝기를 의미한다. ‘fiberMag’는 fiber magnitudes의 약자로, 3인치 지름의 광섬유를 사용하여 광스펙트럼을 측정하는데 이때 나타난 천체의 밝기를 나타낸다. ‘petroMag’는 Petrosian Magnitudes의 약자이다. 은하처럼 뚜렷한 표면이 없는 천체에서는 빛의 밝기를 측정하기 어려운데, 이때 천체의 위치와 거리에 상관없이 빛의 밝기를 비교하기 위한 수치이다 ‘modelMag’는 Model magnitudes로, 천체 중심으로부터 특정 거리의 밝기이다. fiberID를 제외한 모든 feature에는 u, g, r, i, z라는 알파벳이 존재하는데 해당 알파벳은 파장대를 의미하며 구체적인 의미는 다음의 [표 1]에 따른다.

Value	Definition	뜻
u	Ultraviolet	자외선
g	Green	가시광선 중 초록색의 파장
r	Red	가시광선 중 빨간색의 파장
i	Near Infrared	빨간색에 가까운 적외선
z	Infrared	적외선

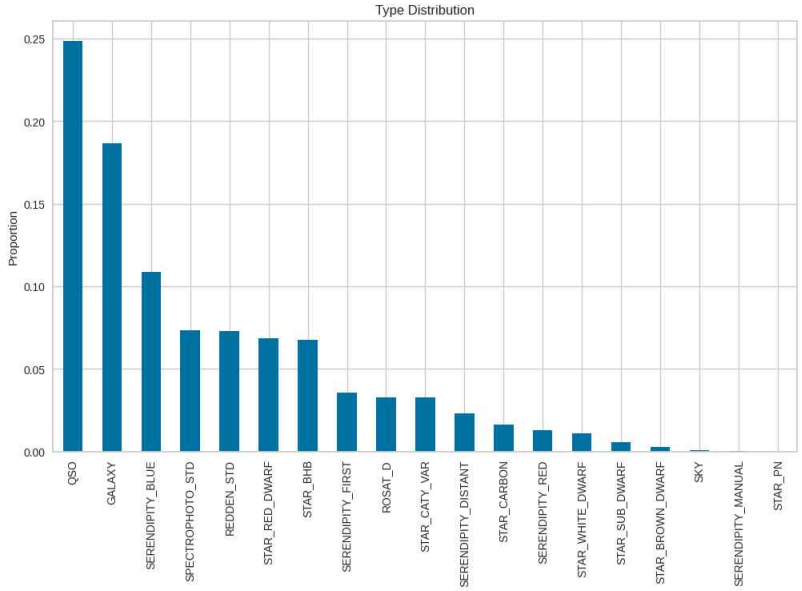
[표 1]

천체 train, test 데이터에서는 결측치가 존재하지 않아 따로 처리해줄 필요가 없다. 그러나 천체 유형별 불균형이 매우 심하고, 이상치가 존재한다. 데이터 불균형과 이상치를 처리하기 위해 데이터 전처리를 진행한다.

4.2 데이터 전처리

4.2.1 데이터 불균형

데이터의 불균형을 살펴보면 다음과 같은 [그림 12]가 그려진다.



[그림 12]

[그림 12]는 train 데이터셋에서 data class의 분포도를 나타낸 그래프이다. 위의 그래프를 보면 'QSO'가 가장 많고, 'STAR_PN'이 가장 적으며 차이가 큼을 볼 수 있다. 이 상태로 모델 학습을 수행할 경우 'QSO'로 예측했을 때 가장 좋은 정답률을 보이기 때문에 한쪽으로 치우쳐져 정확한 모델을 구현하기 매우 어렵다.

이러한 데이터 불균형의 문제를 해결하는 방법으로는 보통 data augmentation (데이터 증강), data sampling(데이터 재분배), class weight(가중치 부여)가 있다. 데이터 증강은 새로운 데이터를 수집하지 않고 모델 훈련에 사용할 수 있는 데이터의 다양성을 증가시키기 위한 전략이다. 데이터 재분배는 데이터셋의 조합을 균형있게 맞추어 여러개의 데이터셋을 만들어 모델 학습에 사용되는 데이터가 균등하게 적용될 수 있도록 하는 방법이다. Class Weight는 해당 Class의 비율의 역수 값의 가중치를 적용하여 'QSO'처럼 빈도수가 높은 Class가 모델에 미치는 영향은 줄이고, 'STAR_PN'처럼 빈도수가 낮은 Class가 모델에 미치는 영향을 높여 균등하게 만드는 방법이다.

그러나 본 연구에서는 이러한 다중 분류에서 나타날 수 있는 문제들을 해결하

기보다, 비율이 비슷한 'SPECTROPHOTO_STD'와 'REDDEN_STD'를 가지고 이진 분류를 한다. 이 두 class만을 분류함으로써 불균형 문제에서 벗어나고 연구함에 있어서 해석하기 쉬운 방향으로 진행하고자 한다.

4.2.2 이상치 제거

```
[ ]  1  # 훈련데이터 요약
     2  df_train.describe()
```

	id	fiberID	psfMag_u	psfMag_g	psfMag_r	psfMag_i	psfMag_z	fiberMag_u	fiberMag_g	fiberMag_r	..
count	199991.00000	199991.000000	1.999910e+05	199991.000000	199991.000000	199991.000000	199991.000000	1.999910e+05	199991.000000	199991.000000	.
mean	99995.00000	360.830152	-6.750146e+00	18.675373	18.401235	18.043495	17.663526	1.084986e+01	19.072693	19.134483	.
std	57732.57318	225.305890	1.187678e+04	155.423024	127.128078	116.622194	123.735298	4.172116e+03	749.256162	90.049058	.
min	0.00000	1.000000	-5.310802e+06	-40022.466071	-27184.795793	-26566.310827	-24878.828280	-1.864766e+06	-215882.917191	-21802.656144	.
25%	49997.50000	174.000000	1.965259e+01	18.701180	18.048572	17.747663	17.425523	1.994040e+01	18.902851	18.259352	.
50%	99995.00000	349.000000	2.087136e+01	19.904235	19.454492	19.043895	18.611799	2.104910e+01	20.069038	19.631419	.
75%	149992.50000	526.000000	2.216043e+01	21.150297	20.515936	20.073528	19.883760	2.233754e+01	21.385830	20.773911	.
max	199990.00000	1000.000000	1.877392e+04	3538.984910	3048.110913	4835.218639	9823.740407	4.870154e+03	248077.513380	12084.735440	.

8 rows × 22 columns

[그림 13]

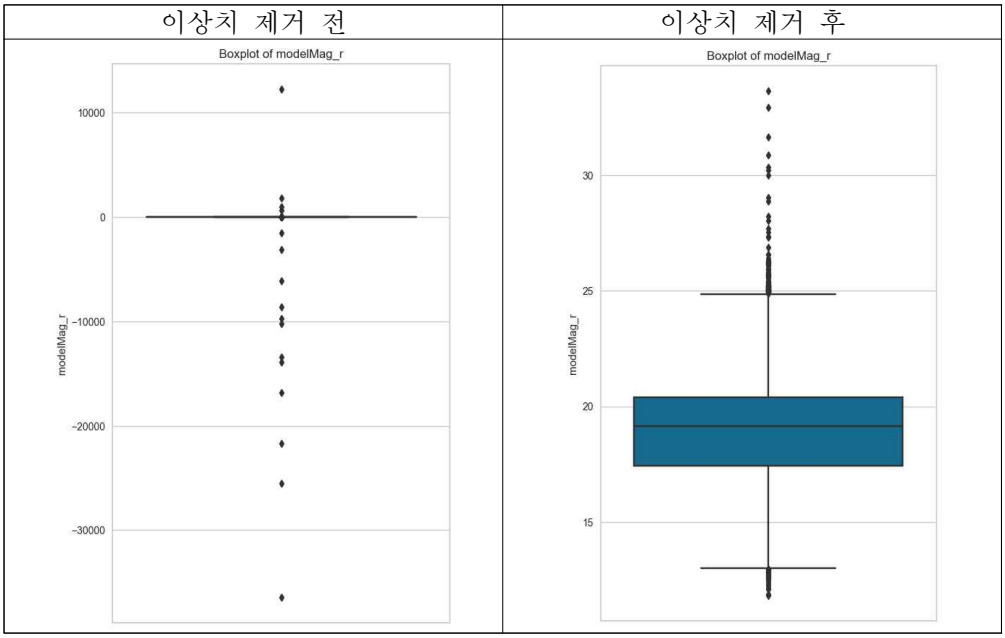
위의 [그림 13]와 같이 train 데이터셋에는 이상치라고 생각되는 높고 낮은 수치들이 존재한다. 이러한 이상치를 제거하기 위한 방법에는 여러가지가 있지만, 본 연구에서는 통계적 방법, 그중에서도 사분위수(Quartiles) 방법을 사용한다. 사분위수 방법은 데이터 분포와 값의 크기를 이용하여 대략적인 이상치의 구간을 설정해준다. 구체적으로는 데이터의 분포에 따라 4등분을 하여 각 부분의 값을 각각 1사분위, 2사분위, 3사분위, 4사분위로 나타낸다. 3사분위 수와 1사분위 수의 차로 IQR(Interquartile Range) 값을 구한 후, IQR * 1.5를 하여 그 이상 벗어난 값들을 이상치로 판별하고 제거한다.

```
[ ]  1  eda.describe()
```

	id	fiberID	psfMag_u	psfMag_g	psfMag_r	psfMag_i	psfMag_z	fiberMag_u	fiberMag_g	fiberMag_r	...
count	199626.000000	199626.000000	199626.000000	199626.000000	199626.000000	199626.000000	199626.000000	199626.000000	199626.000000	199626.000000	...
mean	99991.640362	360.893230	20.982024	19.858800	19.285440	18.870704	18.610927	21.171233	20.089826	19.505251	...
std	57738.783275	225.354843	2.073025	1.855364	1.699939	1.619982	1.677546	1.963918	1.838456	1.703446	...
min	0.000000	1.000000	3.419618	2.741681	11.017885	7.945726	3.888018	4.357522	7.251943	11.795394	...
25%	49988.250000	174.000000	19.652131	18.700760	18.048341	17.747122	17.425033	19.941380	18.903685	18.259843	...
50%	99987.500000	349.000000	20.869446	19.902966	19.453638	19.042660	18.609989	21.048516	20.069053	19.632447	...
75%	149999.750000	526.000000	22.156155	21.146650	20.513680	20.072213	19.882288	22.336302	21.384627	20.773926	...
max	199990.000000	1000.000000	40.741479	34.786670	33.937239	32.691039	32.130146	40.543025	37.745406	30.160957	...

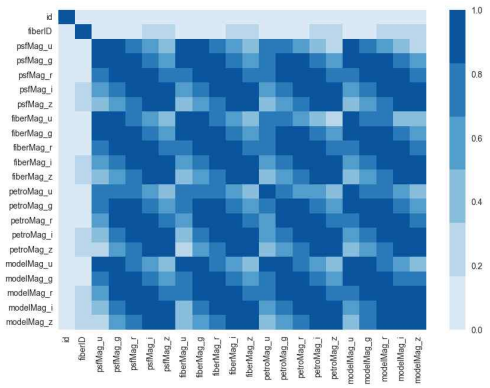
8 rows × 22 columns

[그림 14]



[표 2]

4.3 EDA



[그림 17]

데이터셋의 각 변수들 간의 상관관계를 시각적으로 보여주는 Heatmap을 생성해본다. 색상의 변화가 일정해 보이는 것을 확인해 볼 수 있다.

4.4 모델링 (Modeling)

Modeling 과정은 Python의 AutoML 라이브러리인 Pycaret을 사용하여 모델 간 비교를 통해 해당 데이터에서 좋은 성능을 기대할 수 있는 상위 5개의 모델을 선정하고 그 모델들을 앙상블한 모델들을 만들어서 Accuracy의 비료를 통해 결과적으로 개별 모델과 앙상블 모델 중 어떤 모델이 더 좋은 성능을 내는지를 알아보고자 한다.

이때 데이터의 Target 변수는 기존의 19개의 종류로 나뉘는 천체 유형에서 데이터의 양이 비슷한 SPECTROPHOTO_STD와 REDDEN_STD를 골라서 이진분류 데이터로 만들어주었다. 이때 데이터를 이진분류로 만들어준 이유는 Target 변수가 19개나 되니까 실행시간이 굉장히 오래 걸려서 시간이 제한적인 현 상황에서는 이진분류로 만들어주어서 실행시간을 줄이는 것이 낫다고 판단을 하여서이고, 데이터의 양이 비슷한 SPECTROPHOTO_STD와 REDDEN_STD로 종속변수를 고른 이유는 데이터의 분포가 한 쪽으로 치우쳐져 있으면 모델이 정확한 예측을 하기 어렵기 때문이다.

4.4.1 Model setup

```
setup(data=pd.concat([X_train, y_train], axis=1), target = 'type', session_id = 42, use_gpu = True)
```

[그림 18]

Pycaret을 사용해 단일 모델 성능을 비교하기 전 모델의 setup을 진행하여 해당 데이터에서 좋은 성능을 낼 것으로 기대되는 개별 모델을 찾는다. 고정된 성능 비교를 위해 session_id 값을 42로 고정시키고 Target 변수는 'type'으로 설정하며 use_gpu = True로 두어서 GPU를 사용하여 학습을 수행한다. 이는 gpu를 사용해야 실행이 되는 모델들을 위해서 설정한 것이다.

4.4.2 Model comparison

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.9803	0.9981	0.9795	0.9811	0.9803	0.9606	0.9606	0.3000
gbc	Gradient Boosting Classifier	0.9802	0.9980	0.9787	0.9817	0.9802	0.9603	0.9603	9.4560
catboost	CatBoost Classifier	0.9799	0.9982	0.9784	0.9814	0.9799	0.9598	0.9599	22.0090
lightgbm	Light Gradient Boosting Machine	0.9795	0.9984	0.9794	0.9797	0.9795	0.9590	0.9590	0.5990
xgboost	Extreme Gradient Boosting	0.9792	0.9983	0.9788	0.9796	0.9792	0.9584	0.9584	0.2510
ada	Ada Boost Classifier	0.9782	0.9982	0.9764	0.9801	0.9782	0.9564	0.9564	1.8510
dt	Decision Tree Classifier	0.9701	0.9701	0.9688	0.9714	0.9701	0.9402	0.9402	0.1440
lda	Linear Discriminant Analysis	0.9655	0.9962	0.9834	0.9495	0.9662	0.9310	0.9316	0.0380
ridge	Ridge Classifier	0.9653	0.0000	0.9828	0.9496	0.9659	0.9305	0.9311	0.0250
qda	Quadratic Discriminant Analysis	0.6367	0.9791	0.9943	0.5895	0.7369	0.2724	0.3824	0.0320
lr	Logistic Regression	0.5426	0.5615	0.4218	0.5564	0.4774	0.0857	0.0885	0.0490
knn	K Neighbors Classifier	0.5035	0.5049	0.5090	0.5044	0.5066	0.0069	0.0069	0.0750
nb	Naive Bayes	0.5011	0.5001	0.9999	0.5010	0.6675	0.0004	0.0060	0.0210
dummy	Dummy Classifier	0.5009	0.5000	1.0000	0.5009	0.6675	0.0000	0.0000	0.0170
svm	SVM - Linear Kernel	0.4997	0.0000	0.2993	0.1504	0.2002	0.0001	-0.0020	0.1450

[그림 19]

compare_models()를 실행한 결과 Accuracy가 높을 것으로 예상되는 모델들의 결과가 위부터 순서대로 나왔다.

따라서 우리는 상위 5개 모델인 Random Forest Classifier, Gradient Boosting Classifier, CatBoost Classifier, Light Gradient Boosting Machine, Extreme Gradient Boosting를 개별 모델로 선정하고 이 모델들을 Blending, Stacking한 모델을 만들어서 비교한다.

일단 train set에서 10-fold 교차 검증을 수행했을 때 즉, 데이터를 10개의 부분으로 나눈 뒤, 각 부분을 한 번씩 test set으로 사용하여 총 10번의 학습과 검증을 수행한 후에 도출된 개별 모델들의 Accuracy의 평균을 확인해보면 아래와 같이 나온다.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Random Forest Classifier	0.9789	0.9981	0.9789	0.9789	0.9789	0.9578	0.9578

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Gradient Boosting Classifier	0.9799	0.9986	0.9799	0.9799	0.9799	0.9598	0.9598

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	CatBoost Classifier	0.9785	0.9984	0.9785	0.9785	0.9785	0.9570	0.9570

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Light Gradient Boosting Machine	0.9778	0.9985	0.9778	0.9778	0.9778	0.9556	0.9556

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Extreme Gradient Boosting	0.9774	0.9984	0.9774	0.9774	0.9774	0.9547	0.9547

[그림 20]

위 결과에서 우리의 평가지표인 Accuracy만 본다면 Gradient Boosting Classifier, Random Forest, CatBoost Classifier, Light Gradient Boosting Machine, Extreme Gradient Boosting순으로 높게 나왔음을 알 수 있다.

4.4.3 Model Ensemble

이제 이 모델들을 앙상블(Ensemble)한 모델을 만들어서 성능비교를 해보겠다. **앙상블은 여러 개의 모델을 결합하여 더 좋은 성능을 가지는 모델을 만들어내는 기법이다.** 이러한 앙상블 기법 중 우리는 Blending과 Stacking을 선택하였다. 이때 Blending은 여러 개의 다른 모델들의 예측 결과를 가중 평균이나 투표 등의 방식으로 결합하는 방법을 말하며 Stacking은 Blending과 유사하게 모델들의 예측 결과를 결합하지만, 이를 결합하는 방식이 다르다. Stacking에서는 기본 모델의 예측 결과를 새로운 '메타 모델'의 입력으로 사용하는데, 메타 모델은 기본 모델의 예측을 입력으로 받아 최종 예측을 생성하는 역할을 한다. 이렇게 하면 기본 모델의 예측 결과 간의 복잡한 관계를 학습하여 더 좋은 성능을 얻을 수 있다. 하지만 스태킹은 구현이 복잡하고, 과적합의 위험이 존재한다.

```
# 앙상블 모델 생성 및 학습
blended = blend_models(estimator_list = [rf_model, gbc_model, cat_model, lgb_model, xgb_model], method = 'soft') # 블렌딩 모델
stacked = stack_models(estimator_list = [rf_model, gbc_model, cat_model, lgb_model, xgb_model]) # 스태킹 모델
```

[그림 21]

위와 같은 코드를 이용해서 Blending 모델과 Stacking 모델을 만들었다. 이때 블렌딩 모델의 method = 'soft'로 두어 모델들이 예측한 확률 값을 가중 평균하여 최종 예측을 하게 된다. 그리고 train set에서 Accuracy를 확인해봤더니 아래와 같은 결과가 나왔다.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Voting Classifier	0.9786	0.9986	0.9786	0.9786	0.9786	0.9573	0.9573

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Stacking Classifier	0.9775	0.9941	0.9775	0.9775	0.9775	0.9550	0.9550

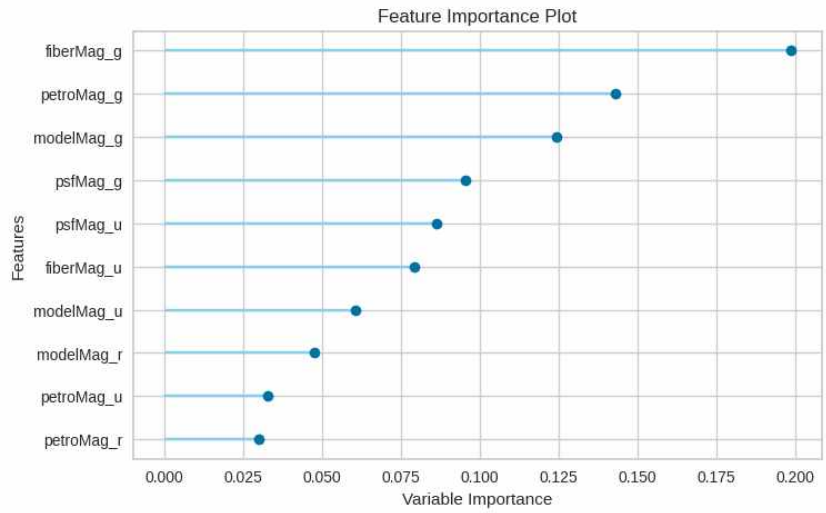
[그림 22]

Voting Classifier과 Stacking Classifier 중 train set에서만 예측을 해봤을 때 Voting Classifier 즉, Blending한 모델이 더 높은 정확도가 나왔음을 알 수 있다.

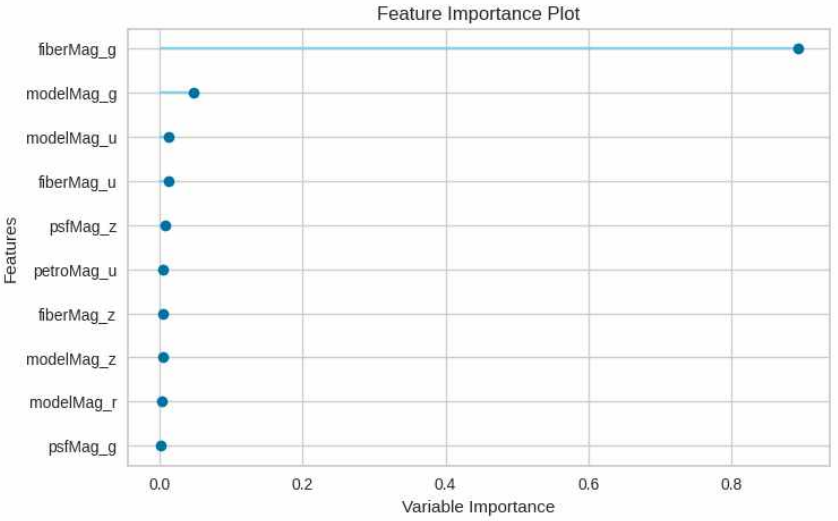
4.5 모델 분석 (Model Analysis)

이후 pycaret의 plot_model(model, plot = 'feature')을 통해서 개별 모델들의 변수 중요도를 파악한다.

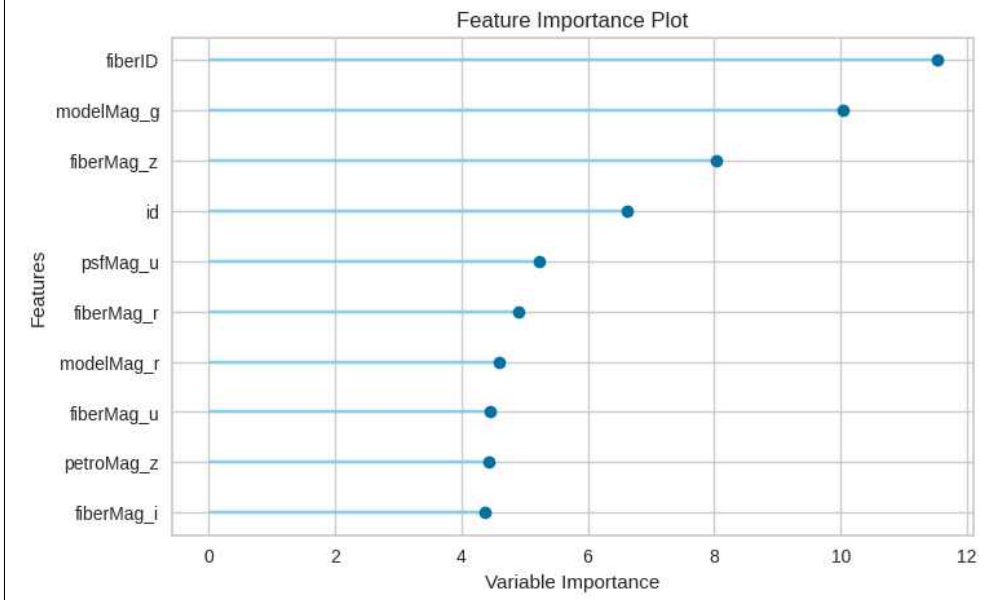
Random Forest Classifier



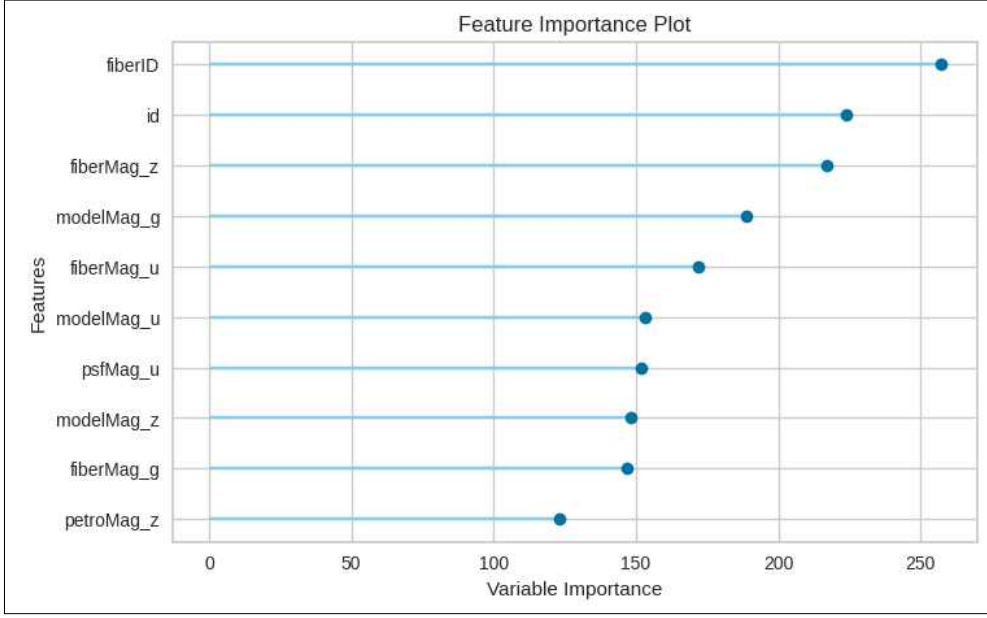
Gradient Boosting Classifier

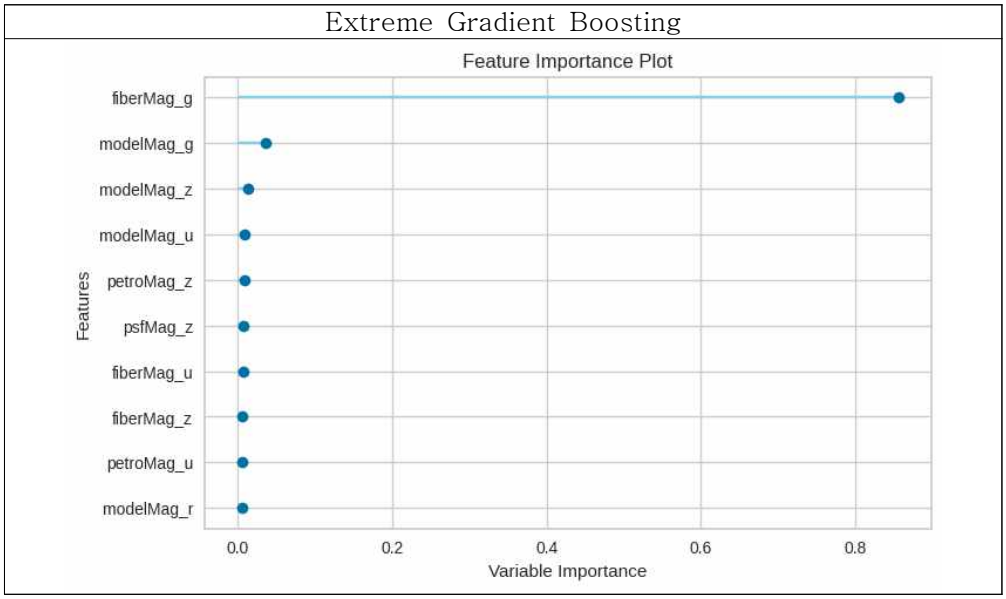


CatBoost Classifier



Light Gradient Boosting Machine

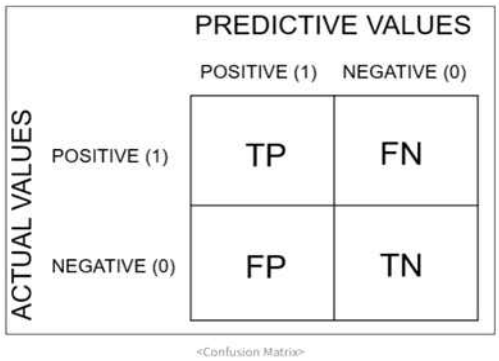




[표 3]

이 결과를 통해서 Random Forest 모델에서는 psfMag_g, fiberMag_g, petroMag_u 순으로 중요한 변수라고 나왔고, Catboost 모델에서는 fiberID, modelMag_g, fibermag_z순으로 Light Gradient Boosting 모델에서는 fiberID, ID, fibermag_z가 중요한 모델이라는 결과가 나왔다. 이때까지 모델들은 각 변수마다 중요도가 큰 차이가 나지 않았는데 Gradient Boosting 모델과 Extreme Gradient Boosting 모델에서는 fiberMag_g 변수가 다른 변수들에 비해서 중요도가 압도적으로 높음을 볼 수가 있다. 이에 대한 이유는 변수들에 대해서 더 파악하고 이해를 한 후에 알 수 있을 것 같다. 또한 위 변수들 중에서 fiberMag_g가 대부분의 모델에서 중요한 변수로 꼽힌다는 점은 눈여겨 볼만한 내용이라고 생각한다. 다음은 comfusion matrix[11]를 사용하여 모델의 정확도를 측정한 내용을 시각화하여 보여준다.

11) comfusion matrix는 모델에서 구한 분류의 예측값과 데이터의 실제값의 발생빈도를 나열한 행렬을 뜻하며 분류모델에서 예측결과를 평가하는데 사용되는 행렬이다.



[그림 23]

TP는 참이라고 예측했는데 실제 값이 참인 것의 비율

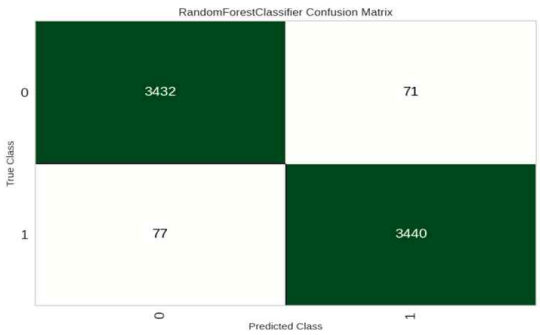
FN은 거짓이라고 예측했는데 실제 값이 참인 비율

FP는 참이라고 예측했는데 실제 값이 거짓일 비율

TN은 거짓이라고 예측했는데 실제 값이 거짓일 비율이다.

$$\text{정확도} = \frac{TP + TN}{TP + FN + FP + TN}$$

이때의 정확도는 (모든 확률 중에서 참이라고 예측했는데 실제 값이 참인 것의 비율) + (TN은 거짓이라고 예측했는데 실제 값이 거짓일 비율)이다. 즉 예상한 값이 맞았을 때 정확도가 높은 것이다. Random Forest Classifier의 comfusion matrix로 예를 들어보면



[그림 24]

모든 사각형 안의 값 = $3432 + 71 + 77 + 3440 = 7020$

초록색 사각형 안의 값 = $3432 + 3440 = 6872$

이때 정확도는 $\frac{6872}{7020} = 0.9789$ 이고 이는 위에서 구했던 Random Forest Classifier의 값과 정확히 일치한다.

4.6 모델 평가 (Model Evaluation)

지금까지는 train set에서만 테스트를 하면서 이에 해당하는 Accuracy를 구했지만 실제 우리가 수행해야 하는 건 **학습된 train set을 통해서 test set의 값을 예측하는 것이다**. 따라서 pycaret의 finalize_model을 이용하여 평가를 하기 위한 최종 모델을 만든다.

Model	Accuracy
Random Forest Classifier	0.9802

Model	Accuracy
Gradient Boosting Classifier	0.9791

Model	Accuracy
CatBoost Classifier	0.9803

Model	Accuracy
Light Gradient Boosting Machine	0.9815

Model	Accuracy
Extreme Gradient Boosting	0.9807

Model	Accuracy
Stacking Classifier	0.9812

Model	Accuracy
Voting Classifier	0.9810

[표 4]

train set으로만 Accuracy를 확인했을 때는 Gradient Boosting Classifier, Random Forest, Voting Classifier, CatBoost Classifier, Light Gradient Boosting Machine, Stacking Classifier, Extreme Gradient Boosting순으로 높았는데 실제 데이터를 통해서 분류과정을 거쳐보니 Light Gradient Boosting Machine, Stacking Classifier, Voting Classifier, Extreme Gradient Boosting, CatBoost Classifier, Random Forest, Gradient Boosting Classifier순으로 순서가 바뀜을 확인 할 수 있다.

따라서 이 데이터에서는 Light Gradient Boosting Machine이 가장 높은 정확도를 보임을 알 수 있다. 또한 개별 모델들은 모델마다 정확도의 차이가 있는데 앙상블 모델의 경우 Blending한 모델과 Stacking한 모델이 각각 0.9810과 0.9812로 높은 성능과 비슷한 정확도를 가짐을 알 수 있다.

5. 결론 및 향후 계획

앙상블 모델이란 “여러 개의 개별 모델을 조합하여 최적의 모델로 일반화하는 방법”으로 대부분의 예측 문제나 분류 문제에서 개별 모델보다 성능이 좋게 나오는 경우가 많다. 이번 분류 문제에 대해서도 앙상블 모델이 개별 모델보다 더 좋은 성능을 낼 것으로 예상했지만 결과적으로 LightGBM모델, 즉 개별 모델이 더 좋은 성능을 냈다.

한편 앙상블 모델을 구성할 때 중요한 요소 중 하나는 weak classifier(분류율이 50%를 조금 넘기는 분류기)들을 결합하여 각 classifier 들로부터 최대한 다양한 결과 값을 얻어낸 후 각 모델에 대한 오류를 서로 상쇄시켜 모델의 전체적인 성능을 향상시키는 것이다. 그러나 이번 프로젝트에서 결합한 모델들은 결과 값이 가장 높은 [12]모델들 5가지를 선정한 것이므로 이미 개별 모델들의 성능이 매우 뛰어나서 앙상블의 효과를 얻기 어려웠던 것으로 추정된다. 따라서 다음에 다시 앙상블 모델을 구성해본다면 weak classifier의 기준에 맞는 모델들을 선정하여 (이 데이터를 예로 들면 Quadratic Discriminant Analysis, Logistic Regression, K-Nearest Neighbors, Naive Bayes, Dummy Classifier) 앙상블 모델을 구성하면 더욱 좋은 성능을 낼 것으로 기대된다.

pycaret을 이용하여 하이퍼파라미터 튜닝을 하여 각 모델의 성능을 높였다면 각 모델에서 현재보다 더 높은 정확도를 가지게 되었을 것이다. 또한 프로젝트 전

12) 앞에서 선정한 5가지 모델인 Random Forest Classifier, Gradient Boosting Classifier, CatBoost Classifier, Light Gradient Boosting Machine, Extreme Gradient Boosting를 의미한다.

체적으로 더 높은 정확도를 가질 수 있었을 것으로 예상되는데, 그러지 못했던 것에 아쉬움이 남는다.

본 연구에서 진행한 머신러닝 성능 평가 연구가 우리 수학과 학생들과 데이터 사이언스 입문자들에게 많은 도움이 되기를 바란다.

참고 문헌

[1] PyCaret Docs, https://pycaret.gitbook.io/docs

[2] DACON, https://dacon.io/competitions/official/235573

[3] SDSS, https://www.sdss4.org/dr17/algorithms

[4] 유혜인, **SDSS 측광 자료를 이용한 우리 은하 내 구상성단들의 수평계열항성 연구**, 이화여자대학교 대학원, 2014.

[5] 권철민, **파이썬 머신러닝 완벽 가이드**, 위키북스, 2022.

[6] 오렐리앙 제롱, **핸즈온 머신러닝**, 박해선 옮김, 한빛미디어, 2023.

[7] 민성환, **부도 예측을 위한 앙상블 분류기 개발**, 한국산업정보학회논문지, 2011

[8] 홍진혁 & 조성배, **분류 성능 향상을 위한 다양성 기반 앙상블 유전자 프로그래밍**, 정보과학회논문지, 2005