Model Design and Training

5조 김나경 김수민 김정하 양승준 이유정

0. github Link

코드 구현은 다음 깃허브 링크의 Modeling.ipynb 파일에서 확인하실 수 있습니다.
https://github.com/urcloud/Wine_Quality.git

1. Data Preprocessing

전체 데이터의 80%를 학습용으로, 나머지 20%를 테스트용으로 분할하였습니다.
데이터 분할과 모델 초기 가중치 설정 등 여러 과정에서 난수가 사용되므로, 동일한 실험 결과를 재현하기 위해 모델 학습 과정의 재현성을 확보하기 위해 랜덤 시드를 42로 고정하였습니다.
전 주차에 정해놓은 대로 학습 데이터의 평균과 표준편차를 활용하여 Z-score 정규화를 사용해 입력 변수들의 스케일을 일관되게 조정하였습니다.

2. Baseline model training

2.0. Model Performance Evaluation Metrics

MSE는 다음과 같은 공식으로 정의되며, 실제값과 예측값의 차이 제곱을 평균한 값입니다

$$MSE = (1/n)\ \Sigma(yi - \hat{y}i)^2$$

여기서 yi는 실제 값, ŷi는 예측 값, n은 샘플 수입니다.
RMSE는 MSE의 제곱근으로, 오차의 크기를 원래 값의 단위에서 확인할 수 있다는 장점이 있습니다.
R^2는 모델이 타깃 값의 분산을 얼마나 설명하는지를 나타내며, 다음 수식으로 정의됩니다

$$R^2 = 1 - (\Sigma(yi - \hat{y}i)^2 / \Sigma(yi - \bar{y})^2)$$

여기서 ȳ는 실제 값들의 평균입니다. R^2 값이 1에 가까울수록 모델이 데이터를 잘 설명하고 있다는 의미입니다.
모델의 성능을 비교하는 지표로 RMSE, R^2값을 사용했습니다.

2.1. Linear Regression

Linear Regression은 독립 변수(X)와 종속 변수(y) 사이의 선형 관계를 모델링하는 가장 기본적인 회귀 기법입니다. 이 모델의 목표는 실제값과 예측값의 차이를 최소화하는 방향으로 최적의 계수 β를 찾는 것이며, 이때 사용되는 손실 함수가 MSE(Mean Squared Error)입니다. 모델의 형태는 다음과 같은 선형 결합으로 표현됩니다

$$y = \beta0 + \beta1 \cdot x1 + \beta2 \cdot x2 + \cdots + \beta p \cdot xp + \varepsilon$$

여기서 y는 예측하려는 타깃 값이며, x1~xp는 입력 특징(feature), β0는 절편(bias), β1~βp는 각 특징의 계수(weights), ε는 오차(residual)를 의미합니다. 이때 최적의 계수 β는 정규방정식 $\beta = (X^TX)^{-1}X^Ty$ 를 통해 계산할 수 있습니다.

앞서 언급한 정규방정식을 이용해 수식 기반으로 파라미터(회귀 계수)를 계산하는 방식으로 코드를 구현했습니다. 그 결과, 소수점 넷째 자리까지 반올림해서 RMSE: 0.6245, R2: 0.4032라는 결과가 나왔습니다.

2.2. Decision Tree

Decision Tree는 입력 특징(feature)과 임계값(threshold)을 기준으로 데이터를 반복적으로 분할해 나가는 트리 기반 모델입니다. 트리의 각 노드에서는 특정 특징 값이 어떤 기준보다 큰지 작은지를 확인하고, 그 조건에 따라 데이터가 왼쪽 또는 오른쪽 자식 노드로 나뉩니다.

노드를 분할할 때는 MSE를 최소화하는 방향으로 특징과 임계값을 찾습니다. 즉, 가능한 모든 특징과 임계값 조합을 탐색해 왼쪽·오른쪽으로 나뉜 후의 타깃값과 평균값 간의 제곱 오차합이 가장 작은 분할을 선택합니다.

이 모델은 기본적으로 트리가 데이터에 맞게 성장하도록 하며, 보통 샘플이 2개 이상이면 분할이 가능하고 리프 노드는 1개 이상의 샘플을 가질 수 있는 조건을 갖습니다. 최종 리프 노드에서의 예측값은 해당 노드에 포함된 타깃값(y)의 평균으로 계산됩니다.

코드에서는 scikit-learn의 DecisionTreeRegressor를 사용하여 결정트리 회귀 모델을 구현하는 방식을 사용했습니다. 그 결과, 소수점 넷째 자리까지 반올림해서 RMSE: 0.7826, R2: 0.0627 결과가 나왔습니다.

3. Advanced model training

3.1. Random Forest

Random Forest는 여러 개의 결정트리를 조합해 성능을 높이는 앙상블 학습 모델입니다. 개별 트리는 서로 다른 데이터 샘플과 일부 특징(feature)만을 활용해 학습되며, 최종 예측은 각각의 트리 예측값을 평균내서 결정됩니다.

이 과정에서 각 트리는 데이터를 뽑고, 다시 넣고, 또 뽑는 방식으로 샘플링하는 방식인 부트스트랩 샘플링과 무작위 특성 선택을 기반으로 한 Bagging 기법을 사용해 학습합니다.

학습 데이터에서 중복을 허용한 랜덤 샘플을 뽑아 각 트리에 제공함으로써, 트리마다 서로 다른 데이터 분포를 보도록 합니다. 또한 노드 분할 시 전체 특징을 사용하는 것이 아니라, 임의로 선택된 일부 특징만 고려해 분할을 진행합니다. 이런 방식은 트리들 간의 상관관계를 줄이고, 개별 트리가 특정 특징에 과도하게 의존하는 것을 막아 과적합을 효과적으로 방지합니다.

코드에서는 scikit-learn의 RandomForestRegressor를 이용한 Random Forest 회귀 모델을 구현하는 방식을 사용했습니다. 총 100개의 트리를 구성하며, 각 트리는 Decision Tree처럼 MSE를 최소화하는 기준으로 노드를 분할하며 학습을 수행하도록 코드를 작성했습니다. 그 결과 소수점 넷째 자리까지 반올림해서 RMSE: 0.5483, R2: 0.5399 결과가 나왔습니다.

3.2. XGBoost

XGBoost 모델은 Gradient Boosting 방식으로 여러 개의 결정트리를 순차적으로 쌓아 올리며 오차를 점진적으로 줄여 나가는 강력한 앙상블 회귀 기법입니다. 이 방식은 단일 트리의 한계를 보완하고, 이전 단계에서 발생한 오차를 다음 트리가 학습하도록 설계되어 있습니다.

먼저 첫 번째 트리로 초기 예측을 만든 뒤, 그 예측값과 실제값의 차이(잔차)를 다음 트리가 학습하도록 합니다. 이렇게 생성된 트리들은 이전 트리의 부족한 부분을 보완하며 순차적으로 더 나은 예측을 만들어내는데, 이 과정을 Gradient Boosting이라고 하며, 손실 함수의 기울기를 따라 오차를 단계적으로 줄여나가는 방식입니다.

XGBoost 회귀 모델(XGBRegressor)을 사용해 모델을 구현했습니다. 코드에서는 총 100개의 트리가 사용되며, 각 트리가 예측에 기여하는 비율(learning rate)은 0.1로 설정되어 있습니다. 낮은 학습률은 안정적인 학습을 가능하게 하고, 여러 개의 트리를 조합해 점진적으로 성능을 향상시키는 데 유리합니다. 최종 예측값은 개별 트리의 예측값을 모두 합산한 뒤 학습률을 곱해 계산합니다. 그 결과 소수점 넷째 자리까지 반올림해서 RMSE: 0.5853, R2: 0.4758라는 결과가 나왔습니다.

3.3. DNN

이 모델은 학습 데이터에서 20%를 검증 데이터로 분리해 사용하며, 분리된 검증 데이터는 Early Stopping과 모델 튜닝에 활용됩니다. 신경망 구조는 64개의 뉴런으로 시작해 32개, 마지막 1개의 뉴런으로 이어지는 형태로 층을 순차적으로 구성합니다.

활성화 함수로는 ReLU가 사용되며, 수식은 ReLU(x) = max(0, x) 로 표현됩니다. 입력값이 0보다 크면 그대로 출력하고, 0 이하인 경우 0을 출력하는 방식입니다. ReLU는 계산이 단순해 학습 속도가 빠르고, 기울기가 일정하게 유지되어 Vanishing Gradient 문제를 완화하는 데 효과적입니다.

Optimizer는 Adam을 적용하며, 이는 Gradient Descent의 변형 알고리즘으로 학습률을 자동 조절하면서 빠르고 안정적으로 최적점에 접근하는 특징을 가집니다. Adam은 기울기의 1차 모멘트를 통해 과거 gradient의 평균을 반영해 방향을 안정화하고, 2차 모멘트를 통해 gradient 제곱 평균을 기반으로 학습률을 조절함으로써 더 효율적인 업데이트를 수행합니다.

또한 Early Stopping 기법을 사용해 검증 데이터의 MSE가 10 epoch 동안 개선되지 않으면 학습을 종료하고, 가장 성능이 좋았던 시점의 가중치로 복원합니다. 이는 과적합을 방지하며 불필요한 장시간 학습을 줄이는 데 크게 도움됩니다. 학습 과정에서는 batch size 32로 데이터를 나누어 순차적으로 학습을 진행하며, 검증 성능을 지속적으로 확인해 성능 향상이 없을 경우 즉시 학습을 멈추도록 설계되어 있습니다.

모델은 Keras로 구현했으며, 원래 학습 데이터에서 20%를 검증용(validation) 데이터로 분리해 사용하도록 설계했습니다. 64개 뉴런을 갖는 첫 번째 층, 32개의 뉴런을 갖는 두 번째 층, 1 개의 층을 갖는 마지막 층으로 구성되어 있으며, 최대 200번 반복 학습, 한 번에 32개 샘플씩 학습하도록 설정이 되어 있습니다. 그 결과 소수점 넷째 자리까지 반올림해서 RMSE: 0.6258, R2: 0.4007라는 결과가 나왔습니다.

4. Comparing each models

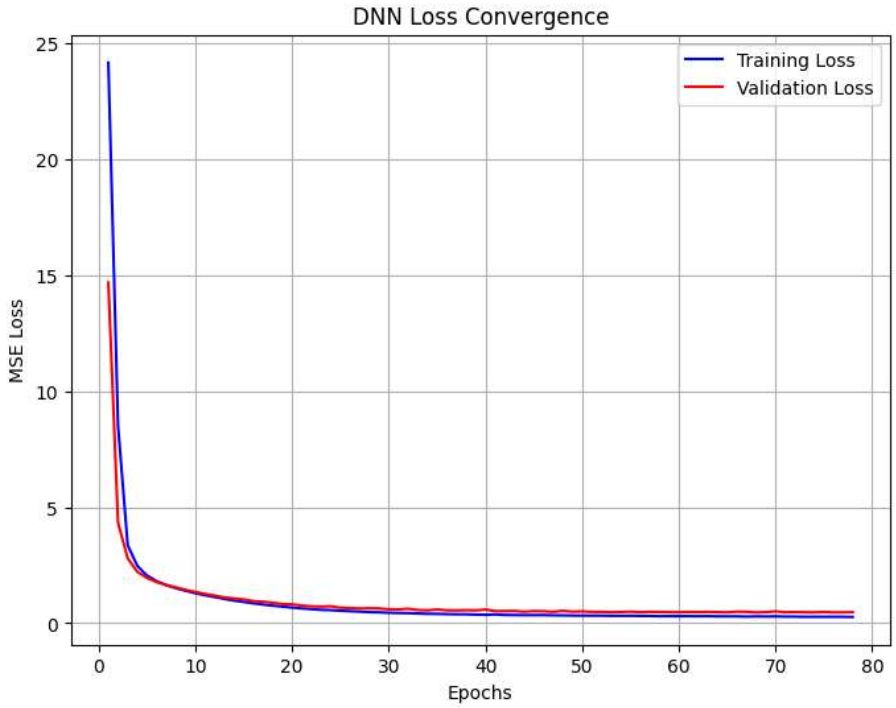
Model	RMSE	R2
Linear Regression	0.6245	0.4032
Decision Tree	0.7826	0.0627
Random Forest	0.5483	0.5399
XGBoost	0.5853	0.4758
DNN	0.6207	0.4105

RMSE : Decision Tree > Linear Regression > DNN > XGBoost > Random Forest

R^2 : Random Forest > XGBoost > DNN > Linear Regression > Decision Tree

정리하자면, 위와 같은 성능이 나왔습니다. RMSE는 적을수록 모델의 오차가 적다는 뜻이고, R^2는 1에 가까울수록 해당 모델의 설명력이 높다는 뜻이므로 Random Forest > XGBoost > DNN > Linear Regression > Decision Tree 순으로 모델의 성능이 좋다고 결과를 해석할 수 있었습니다.

5. DNN Loss Convergence Graph



DNN이 가장 성능이 좋을 것이라 예측했지만 성능이 생각보다 잘 나오지 않아 시각화해서 원인을 유추해보았습니다.

학습 초기 몇 epoch 동안 Training Loss와 Validation Loss가 매우 빠르게 감소하는 모습을 통해 모델이 기본적인 패턴을 빠르게 학습하고 있다는 것을 알 수 있었습니다.

약 10~20 epoch 이후에 Loss가 안정화되는데, 손실이 거의 변하지 않고 평평하게 수렴하는 모습도 관찰할 수 있었습니다. Training Loss와 Validation Loss 거의 동일해 과적합(overfitting)은 발생하지 않음을 확인할 수 있었습니다.

학습률이 너무 커서 loss가 평탄한 지점에 갇혔거나, 더 복잡한 비선형 패턴을 학습해야 하는데 모델의 크기가 적합하지 않았거나, 데이터 수가 적은 것 그 자체로 성능 저하가 발생한 것 같다는 의견이 나왔습니다.

5. Methods for Improving DNN Performance

따라서 DNN 모델의 일반화 성능을 향상시킬 수 있도록 다음과 같은 방법을 사용하였습니다.

5.1. K-fold

전체 데이터를 K개의 폴드로 분할한 뒤, 각 폴드를 검증용으로 교차 사용함으로써 모델의 일반화 성능을 보다 안정적으로 평가하는 방법입니다. 훈련/검증 데이터 편향을 감소해주는 역할을 합니다. 코드에서는 데이터를 5개의 폴드로 나눠서 학습/검증 반복을 수행하도록 설정했습니다.

소수점 넷째 자리까지 반올림해서 Average RMSE: 0.6176, Average R2: 0.4161라는 결과를 얻었는데, 모델 구조나 학습 전략은 그대로이기 때문에 모델의 성능이 크게 좋아지지 않는 모습을 확인할 수 있었습니다.

5.2. Reducing learning rate and Using ReduceLROnPlateau

ReduceLROnPlateau는 딥러닝에서 학습률(learning rate)을 동적으로 조정해주는 기법 중 하나입니다. 모델 학습 중에 검증 성능(val_loss 등)이 개선되지 않으면 학습률을 줄여서 학습을 더 안정적으로 만들도록 도와줍니다. 학습이 정체될 때 학습률을 자동으로 낮춰서 최적화가 더 잘 되게 하는 콜백입니다.

기본 Adam LR = 0.001인데, 코드에서는 0.0005로 낮춰서 더 안정적이고 천천히 수렴하도록 설정하였습니다. 또한, Loss가 더 이상 줄지 않을 때 학습률을 떨어뜨려 더 미세한 방향으로 최적화하기 위해 ReduceLROnPlateau를 5 Epoch 동안 val_loss가 개선되지 않으면 LR을 절반으로 감소하고, 최소 learning rate는 0.000001까지로 설정했습니다.

소수점 넷째 자리까지 반올림해서 RMSE: 0.6152, R2: 0.4208라는 결과를 얻었습니다. 따라서 모델의 성능이 더 좋아졌음을 확인할 수 있었습니다.

5.3. Adding hidden layers/nodes

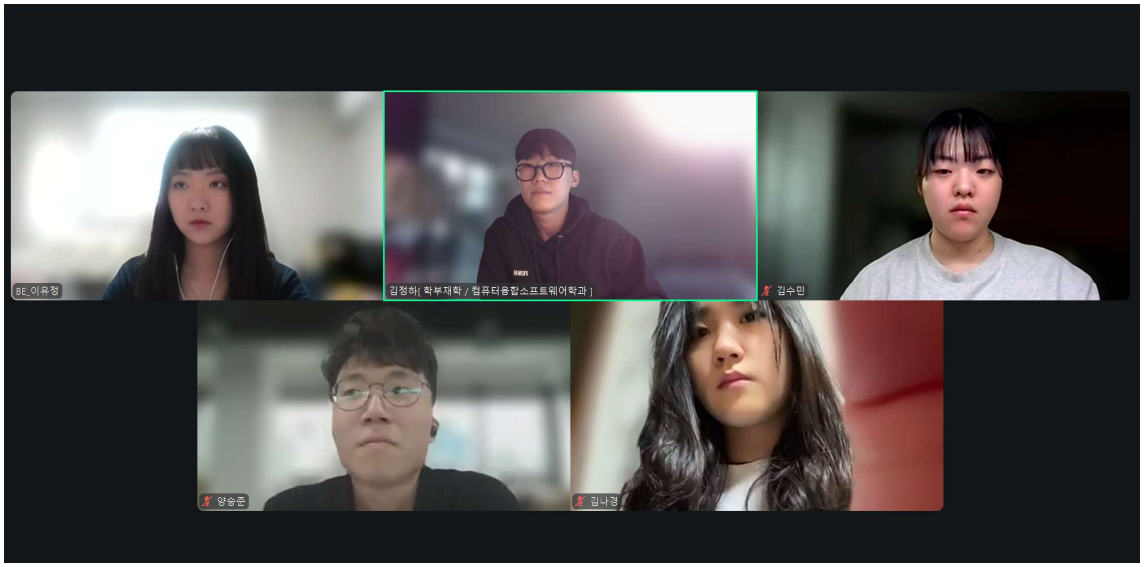
기존 DNN 모델은 64개 뉴런을 갖는 첫 번째 층, 32개의 뉴런을 갖는 두 번째 층, 1 개의 층을 갖는 마지막 층으로 구성되어 있었는데, 이를 128, 62, 1개의 뉴런을 갖도록 뉴런을 증가시켰습니다.

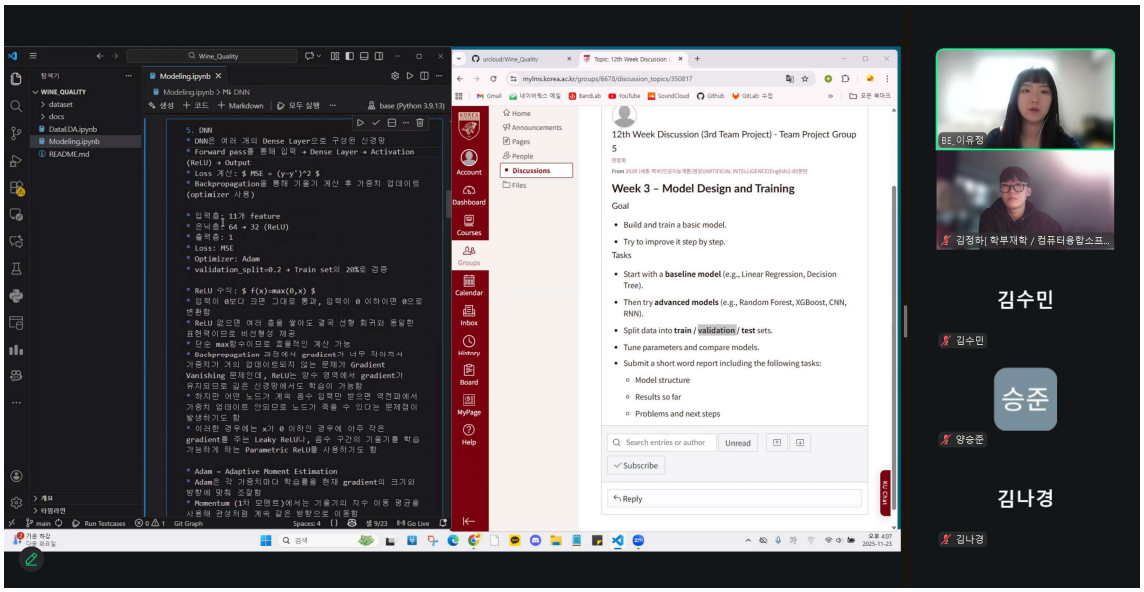
소수점 넷째 자리까지 반올림해서 RMSE: 0.6086, R2: 0.4332라는 결과를 얻었습니다. 모델의 성능이 가장 크게 향상됨을 확인했는데, 데이터의 패턴이 복잡해서 더 큰 모델이 효과적이었을 가능성이 높다는 것을 유추할 수 있었습니다.

6. Conclusion

learning rate가 너무 크면 loss가 평탄한 지점에 갇힐 수 있으므로 learning rate를 적절히 줄이고, 데이터의 패턴이 복잡한 경우 더 큰 모델이 효과적일 수 있으므로 노드 수나 깊이 증가를 통해 복잡한 패턴을 더 잘 학습할 수 있도록 해야 함을 확인할 수 있었습니다. 또한 적절한 파라미터 값 조정을 통해 모델의 성능을 향상시킬 수 있음을 확인했습니다.

7. Team Meating





이번 주도 줌으로 팀 미팅을 진행했습니다.