<개념 세션 대본>

안녕하세요 이어서 릿지라쏘 회귀에 대해 설명하겠습니다. 릿지라쏘를 처음 접하시는 분들이라면, 내용이 다소 어렵게 느껴지실 수 있는데, 구체적인 공식에 집중하기보다는, 릿지라쏘를 "왜" 사용하는 것인지 전체적인 흐름을 기억해주시면 좋을 것 같습니다.

먼저, bias과 variance의 개념에 대해 살펴보겠습니다. 모델을 적합했을 때 발생하는 error는 크게 bias와 variance로 나눌 수 있습니다. Bias, 즉 편향은 모델이 데이터를 제대로 설명하지 못해서 발생하는 에러이고, variance, 즉 분산은 모델이 데이터의 작은 변화에도 너무 과도하게 반응해서 발생하는 에러입니다. 그림에서 가운데의 빨간색 점이 정답이고, 파란색 점들은 우리가 모델을 사용해서 예측한 것들입니다. 1행에 low bias 부분을 보면 2행 high bias에 비해 빨간색 점에 가까이 위치해 있음을 알 수 있고, 1열에 low variance는 2열 high variance에 비해 점들이 퍼져있지 않고 모여있음을 확인할 수 있습니다.

우리는 앞서 실습 세션에서 train set과 test set으로 나누는 과정을 거쳤습니다. Train set으로 모델을 학습시킨 후, 학습에 사용된 train set으로 평가할 뿐만 아니라, 학습에 사용하지 않은 새로운 데이터인 test set을 사용하여 모델 성능을 평가하게 되는데, 이때 underfitting, overfitting 문제를 확인할 수 있습니다. 회귀뿐만 아니라 거의 대부분의 모델을 적합하고 평가하는 과정에서, 우리는 underfitting, overfitting 문제가 있는지를 반드시 확인해야 합니다. 먼저 underfitting은 모델이 train set을 너무 조금 공부해서 간단한 모델이 되어 데이터의 특징을 제대로 반영하지 못할 때, 즉 bias가 높을 때 발생합니다. 동그라미, 엑스를 구분하는 문제가 주어졌다고 했을 때, 가운데의 그림의 빨간선은 이상적인 구분 모델입니다. 하지만 왼쪽 일직선으로 된 모델은 오분류된 것들이 가운데에 비해 훨씬 많고, 직선 형태라 매우 단순합니다.

다음으로 overfitting 문제는 모델이 train set을 너무 열심히 공부해서, train set의 특성을 매우 잘 보여주지만, 새로운 test set에서는 제대로 성능을 발휘하지 못할 때 발생합니다. 모델이 복잡해 variance가 높아졌기 때문입니다. 오른쪽 그림을 보면 모델이 매우 복잡하죠? 그런데 o와 x는 아주 완벽하게 분류하고 있습니다. 여기에 o, x 분포가 다른 test set을 적용했을 때는 이 너무 복잡한 부분들 때문에, 에러로 변하는 것들이 많아질 것입니다.

정리하자면, 모델을 적합할 때 우리는 크게 두 가지의 목표를 설정합니다. 첫째, test set의 에러와 train set의 error가 비슷할 것, 둘째, train error는 0에 가까울 것. Test error가 train error보다 높아서 첫번째 목표가 실패했을 때, overfitting 문제가 발생합니다. Train error가 0보다 커서 두번째 목표가 실패했을 때, underfitting 문제가 발생합니다. Bias와 variance는 그림처럼 trade-off 관계 즉, 하나가 커지면 하나는 작아지는 이런 상반된 관계를 가지기 때문에, 두 가지 목표를 모두 완벽하게 달성할 순 없고, 두 목표를 모두 적절하게 만족하는 어느 이상적인 지점을 찾아야 합니다. 그 이상적인 부분은 이 점으로 된 부분, 즉 bias와 variance를 더한 test set의 error가 가장 최소화되는 지점입니다.

Overfitting 문제를 해결하기 위한 방법 중 하나가 바로 regularization, 즉 규제화입니다. 일반적으

로 회귀 모델에서 계수의 크기가 큰 경우, 모델은 overfitting될 가능성이 높아지는데, 규제화는 계수의 크기를 제한함으로써 모델이 예측할 때 사용하는 변수의 개수를 줄이거나 모델의 복잡도를 줄입니다. 이를 통해 overfitting을 방지하고, 일반화 성능이 향상될 수 있습니다.

규제화 방법에는 Lasso, Ridge 두 가지가 있습니다. 이 페이지는 앞으로 말할 내용을 요약해 둔 부분입니다. 지금 여러분께서 반드시 기억하셔야 할 부분은, lasso는 계수 값을 0으로 줄일 수 있으나, ridge는 계수 값을 0으로 만들지 못한다는 내용입니다.

먼저, 라쏘에 대해 알아보겠습니다. 식에서 yi-wixi+b의 제곱합이라고 적힌 부분이 지금까지 봐왔던 일반적인 선형 회귀의 에러 공식이고, 파란색으로 밑줄 친 부분이 라쏘에 해당하는 부분입니다. 이처럼 L1 규제화 방법인 라쏘는 계수의 절댓값 합을 최소화합니다.

최적화 과정에서 라쏘는 일부 계수가 0으로 수렴하는 경향이 있습니다. 이는 모델에서 해당 변수를 제외하는 역할을 합니다. 즉, 변수 선택이 이루어집니다. 앞의 식을 계수 w에 대해 정리하면 이러한 식이 되는데, 분자에서 알파를 빼는 부분이 보이시죠? 상수인 알파를 빼면 분자는 0이 될 수 있습니다. 즉, 계수 w가 0이 될 수 있다는 것입니다. 그래서 변수 간 상관관계가 낮을 때, 특정 변수들이 예측에 중요한 역할을 할 때, 라쏘가 유용합니다. (상관관계가 높은 상황에서, 라쏘는 상관관계가 높은 애들을 다 0으로 보내버리는 경향이 있음. 특정 변수들이 예측에 중요한 역할을 한다는 것은 변수 선택이 필요하다는 것인데, 라쏘는 변수 선택 기능이 있음.)

이 공식을 다시 보면, 계수 앞에 알파가 붙어 있는 것을 볼 수 있는데요, 이 알파는 규제화 강도를 조절하는 하이퍼파라미터입니다. 즉, 모델이 학습하는 파라미터와 달리, 우리가 직접 값을 지정해주어야 함을 의미합니다. 알파가 커질수록 규제화 강도가 강해져서 계수의 크기를 작게 만들고, 이게 0이면 규제화 없는 일반적인 선형 회귀와 동일합니다. 그림에서 알파가 커질수록 모델의 모양이 점차 단순해지는 것을 볼 수 있습니다.

다음으로, 릿지에 대해 알아보겠습니다. 식에서 맨 뒤의 w 부분이 절댓값에서 제곱으로 바뀌었습니다. L2 규제화 방법 릿지는 계수의 제곱합을 최소화합니다. 계수의 제곱을 하는 것이므로, 계수가 커질수록 패널티가 기하급수적으로 증가하고, 예를 들어 0.5를 제곱하면 크기가 더 작아지니까, 계수가 1보다 작을 때는, 라쏘에 비해 규제를 덜 가합니다.

앞의 식을 w에 대해 정리하면 이러한 식이 되는데, 분모에서 알파를 더하는 부분이 보이시죠? 알파가 무한대에 가까워져야 분모가 0이 될 수 있습니다. 알파의 크게 함에 따라 w를 작게 만들 순 있으나, 0으로 만들지는 못합니다. 그래서 변수 간 상관관계가 높을 때, 특정 변수들이 예측에 미치는 영향이 크게 다르지 않을 때 유용합니다.

여기서도 알파는 규제화 강도를 조절하는 하이퍼파라미터입니다. 마찬가지로, 알파가 커질수록 규제화 강도가 강해져서 계수의 크기를 작게 만들고, 이게 0이면 규제화 없는 일반적인 선형 회귀와 동일합니다. 그림에서 알파가 커질수록 모델의 모양이 점차 단순해지는 것을 볼 수 있습니다.

세번째 규제화 방법인 엘라스틱넷은 릿지와 라쏘를 합친 방법입니다. 라쏘에 대한 알파, 릿지에

대한 알파, 총 2개의 하이퍼파라미터를 가집니다. 상관관계가 큰 변수들을 모두 선택하거나 제거 가능해 다수의 변수 간에 상관관계가 존재할 때 유용합니다.

<실습 세션 대본>

릿지 라쏘 회귀를 위해 hitters라는 새로운 데이터를 불러오겠습니다. 메이저 리그 야구선수들에 관련된 데이터이고, 구체적으로 열이 무엇을 의미하는지는 모르셔도 괜찮고, Salary가 y이고, 나머지가 변수들이 다 x입니다. 짜잘한 얘기긴 한데, 제가 df.head 뒤에 transpose를 의미하는 T를 붙여놨는데, 이렇게 하면 변수들이 많은 데이터를 스크롤바 사용할 필요 없이 한눈에 보기가 좀 더 편리해서 전 개인적으로 많이 사용하고 있습니다.

각 변수마다의 결측치의 개수를 nasum 변수에 저장해 nasum이 1 이상인 것만 출력했더니, salary 변수에 결측치가 59개 있는 것으로 확인되었습니다.

결측치가 있는 행을 다 삭제해 info를 해봤더니, 263행, 20열로 데이터가 이루어져 있습니다.

이제 전처리를 해보겠습니다. 릿지라쏘가 중심이므로 복잡한 전처리는 사용하지 않았습니다. 우선, y에 타겟변수인 salary를 저장했습니다. League, division, newleague 변수가 범주형 변수이기 때문에, pd.get_dummies로 더미화를 진행해 dummies에 그 결과값을 저장하였습니다. 원래 데이터프레임에서는 y인 salary와 더미화했던 범주형 변수들을 드랍해 숫자형 변수만 존재하는 X_numerical이라는 새로운 데이터셋을 만들어주었습니다. 그리고 숫자형 변수들의 이름을 list_numerical에 저장했습니다. 원래 league 변수에 대해 더미화를 하면 league_A와 league_N가 만들어지는데, 두 개를 다 사용하면 다중공선성 문제가 발생하므로, league_N만 사용하려고 합니다. 마찬가지로, division_W, Newleague_N만 선택하고, 이를 숫자형 변수들과 concat해서 독립변수 X를 만들었습니다.

다음으로 X와 y를 7:3의 비율로 train, test set으로 랜덤하게 나누었습니다. X_train은 이렇게 생겼습니다.

마지막으로, X의 숫자형 변수들에 대해 스케일링을 진행하겠습니다. 위와 달리, standardscaling 방법을 사용할건데, 이는 원본 값에서 평균을 뺀 후, 표준편차로 나누어서, 평균 0, 분산 1로 만드는 방법입니다. Standardscaler 객체를 만든 후, X_train의 숫자형 변수들로 학습시켜 scaler에 저장합니다. 학습했으니 이제 값을 바꿔야겠죠? X_train을 transform으로 바꾸고, X_test도 transform으로 바꿉니다.

값은 이렇게 바뀝니다.

그럼 이제 라쏘를 사용해보겠습니다. 일단 알파를 임의로 1로 설정하겠습니다. Lasso 객체에 알파 1을 넣어 reg에 저장하고, 여기에 X_train, y_train으로 학습시킵니다.

그럼 이제 모델의 성능을 평가해봅시다. Score 함수로 결정계수를 확인할 수 있는데, train set으로

확인해보니 60.43이고, test set은 33.01입니다.

이번에는 mse를 확인해보겠습니다. Mse는 예측값에서 참값을 빼야 하므로, 예측값을 먼저 구해야 합니다. Predict 함수를 써서 X_train에 대한 y의 예측값을 구하고, 참값인 y_train과 예측값인 pred_train을 mean_squared_Error 함수에 넣어 mse를 구했습니다. 동일한 과정을 test set으로 바꿔 진행합니다. 결과는 다음과 같습니다.

그럼 이제 알파를 바꿔가며 모델의 성능을 확인해봅시다. Np.linspace를 사용하여, 0.0001부터 500까지의 수를 동일한 구간으로 나누어 200개의 숫자를 추출하고, 이 200개의 숫자는 알파들입니다. Lasso 객체를 새롭게 만들어주는데, 최적화를 위한 최대 반복 횟수를 10000번으로 설정합니다. 각 알파값을 lasso 객체에 넣고, train set에 적합한 후, 거기서 도출된 계수, 결정계수, mse를 coefs, r2, mse 리스트에 각각 저장합니다.

저장한 값들을 그래프로 그리려고 하는데, 그래프 그리는 함수를 반복하기는 귀찮으므로, show_me_plot이라는 함수를 만들어보겠습니다. x축을 alpha, y축을 k에 해당하는 계수, 결정계수 또는 mse로 잡고 선그래프를 그립니다. 알파의 범위가 매우 넓기 때문에, set_xscale('log')를 통해 x축을 로그 변환해줍니다.

먼저, 계수들이 변화하는 모습입니다. 알파가 커질수록, 계수들이 작아지고, 개념 시간에 배웠던 것처럼 결국 0이 되는 것을 확인할 수 있습니다.

결정계수는 알파가 커질수록 작아지고 있습니다. 규제화는 모델의 학습을 막는 것이기에, 학습을 막을수록 결정계수가 작아지는 것은 당연한 것입니다.

Mse는 알파가 커질수록 커집니다. 마찬가지로, 규제화가 학습을 막을수록, 예측값과 참값의 차이는 점차 커질 것입니다.

이 세 그래프만 보고, 어떤 것이 적절한 알파인지 찾기는 어렵습니다. 최적의 알파를 찾아 주는 함수가 바로 LassoCV입니다. 5 fold cross validation은 데이터를 다섯 폴드로 나누어서 첫 번째 폴드를 test set으로 사용하고, 나머지 4개 폴드는 train set으로 사용합니다. 이를 다섯 번 반복한 후, 각 반복마다 나오는 성능의 평균을 최종 성능으로 보는 방법입니다. 이렇게 하면 좀 더 안정적인 결과를 도출할 수 있습니다. LassoCV 객체에 cv, 즉 폴드 개수 5개, 최대 반복 횟수 10000번을 넣어주고, train set에 적합해 model에 저장합니다.

Model 뒤에 Alpha_를 붙이면 lassocv가 찾은 최적의 알파값을 알려주는데, 2.34입니다.

그럼, 이 최적의 알파값으로 lasso를 적합한 lasso_best의 성능을 확인해봅시다.

계수를 보니, catbat, chits는 완전히 0이 되었음을 알 수 있습니다.

결정계수는 이렇게 나오고,

Test set의 mse는 이렇습니다.

그럼 이제 릿지를 해볼 건데요, 과정은 라쏘와 동일합니다. 다만, lasso 대신 ridge로 바꿔줘야 합니다. 임의로 알파를 1로 설정해 적합하면 다음과 같은 결과들이 나옵니다.

그럼 이제 알파를 변화시켜 그래프를 그려보겠습니다.

알파가 커질수록 계수가 작아지긴 하지만, 0이 되지는 않습니다.

결정계수는 알파가 커질수록 작아지고,

Mse는 알파가 커질수록 커집니다.

이번에는 ridgeCV로 최적의 알파값을 찾아보겠습니다.

적합했더니, 최적의 알파는 10이네요.

이걸로 다시 적합시켜 ridge_best를 만들어보니,

계수들은 다음과 같은데, 0이 되지는 않았습니다.

결정계수, mse입니다.

실습은 여기까지고요, 모두 들으시느라 고생많으셨습니다!