6.1.1

최적 부분집합 선택:

예측 변수 p개에서 모든 집합 고려, least squared regression 수행 해서 가장 좋은 모델선택 방식.

최적부분집합 선택 과정:

1. M\_0: 예측 변수 없는 null model 시작. 🡺 단순 데이터 평균
2. 예측할거 K개가 있다고 하면, 모든 모델 pCk 저장해놓고
3. 그중에서 RSS가장 작은거나 R^2 가장 큰거 선택하고 M\_k로 지정.

거기서 여러 M\_0, M\_1 …, M\_p 있다고 하면 교차검증, k-fold, R^2 보고 가장 좋은거 하나 선택.

Best Subset Selection:

모델 전부 만든거 중에서 가장 좋은 모델 선택하는 방법.

다만, 예측할거(p) 많아질수록 모델 수 2^p로 만아져서 계산할거 많아짐.

그래서 Branch-and-bound 방식 나왔는데, 선형회귀에만 적용 가능.

뭐 이런 문제들 해결 하기 위해서

Best Subset Selection 방식에 Forward Stepwise Selection 알고리즘 적용된거 나옴.

Predictors 하나씩 추가하면서 모델 만드는 방식.

1. 기본 M\_0 모델(아무것도 없는)에서 시작.
2. 현재 모델 M\_k에 추가할 수 있는거 고려해서 가장 좋은거(RSS, R^2고려) M\_(k+1) 생성.
3. 이런식으로 나온 모델 M\_p 중에서 C\_p, BIC, R^2, 교차검증 해서 가장 좋은 모델 선택.

근데 Forward Stepwise Selection은 계산량만 봤을 때는 더 좋지만, 변수 놓칠 수 있어서 그닥.

6.1.2

Forward Stepwise Selection:

위의 Best Subset Selection은 계산량 너무 높아서 별로 안좋음. 개선책으로 모든 모델 탐색 방식 대신 점진적 모델 확장 방식.

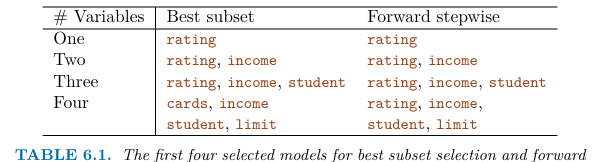
처음엔 동일하게 M\_0처럼 null model로 시작. 거기다가 변수 하나씩 추가하여 모델 확장하는 방식.

1+p(p+1)/2 모델만 계산하면 돼서 충분히 계산량 줄어듦.

고차원 데이터에서도 당연히 쓸수 있고.

단점: 1번째에서 X\_1 선택했다고 하면 2번째에소 X\_1 들어가야 함.

최적 변수가 X\_2, X\_3 이라고 하면 선택 못할 가능성 있음.





Forward Stepwise Selection이랑 Best Subset Selection 결과가 위의 방식처럼 나옴.

봐보면 거의 비등비등 해서 둘 다 써도 된다고 함.

근데 이것도 n(데이터 row수) < p(데이터 column 수) 경우, Least Squares 썼을 때, M\_(n-1) 까지만 가능.

Backward Stepwise Selection:

Forward Stepwise Selection 반대 방식. 전체모델 M\_p에서 시작해서 변수 하나씩 없애면서 모델 축소.

1. P개 변수 다 포함한 모델 시작점.
2. M\_k에서 변수 하나씩 제거한 K개 모델 생성.
3. RSS, R^2 보고 M\_k-1로 선택.
4. 여기서 C\_p, BIC, R^2, 교차검증 보고 최적 모델 선택.

계산량은 1+p(p+1)/2 로 동일. Best Subset Seletion 보다 훨씬 계산량 적음.

위랑 반대로 n > p 에서만 적용 가능.

Hybrid Approaches:

Forward랑 Backward Stepwise Selection 결합방식.

Forward처럼 변수 추가하면서, 여기서 덜 필요한거 제거하는 방식.

성능 자체는 Best Subset Selection과 성능은 비슷하게, 계산량은 Stepwise Selection처럼.

6.1.3

RSS, R^2는 predictors 추가하면 할수록 training 오류는 감소.

근데 이게 테스트 오류가 좋을 거라는 보장 X

RSS 낮을수록, R^2 높을수록 학습 잘됐다는 건데, 과적합 발생 가능성, 테스트 오류 반영 X

그래서 테스트 오류 보기 위해서 나온게 C\_p, AIC, BIC, Adjusted R^2 나옴.

C\_p:



d:predictors(선형회귀의 X\_1, X\_2 생각)

n: 데이터 개수

: 오차 분산 추정값.

앞에 RSS나오는데 이게 훈련 RSS. 이거 좀 휘뚜루빠뚜루 해서 test 에러 봐보는 것.

기존 훈련 RSS에 d값, 변수 추가될수록 결국 값 높아지는거.

결국 C\_p값이 낮은게 테스트 오류 가장 낮은 모델 의미.

텍스트, 스크린샷, 라인, 도표이(가) 표시된 사진

자동 생성된 설명

그래프 봐보면 변수 6개 들어간게 최적

AIC:

폰트, 화이트, 서예, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

C\_p랑 동일한 식.

최대우도 쓴다는 차이 존재.

C\_p는 선형회귀 모델에 제한되는데,

AIC는 선형이든 비선형이든 다양한 곳에 사용 가능. 🡺 좀 더 다양하게 쓸 수 있게 유연하다는 장점 존재.

그래프 형태 얼추 비슷해보임. 식도 큰 차이 없어서.



C\_p랑 비교했을 때, 폰트, 서예, 화이트, 텍스트이(가) 표시된 사진

자동 생성된 설명를 패널티 항으로 바뀜.

Log(n)에서 n >7일 때, 2보다 커져서(앞에 2d시그마 봐볼 것) 결국 BIC는 변수 작은걸 좋아라 함.

텍스트, 스크린샷, 라인, 도표이(가) 표시된 사진

자동 생성된 설명이거처럼 최적 변수 개수 4개로 줄어든게 보임.

R^2 = 

TSS =  # Y-은 평균값

RSS = 폰트, 화이트, 디자인, 타이포그래피이(가) 표시된 사진

자동 생성된 설명 Y^ 개별 예측값.

변수 추가하면 추가할수록 R^2값은 증가하지만, 이는 과적합 위험.

텍스트, 폰트, 화이트, 라인이(가) 표시된 사진

자동 생성된 설명

그래서 나온게 Adjusted R^2.

변수 d 늘어날수록 최종 Adjusted R^2값은 줄어듦.

텍스트, 스크린샷, 라인, 도표이(가) 표시된 사진

자동 생성된 설명

Validation set & Cross validation:

둘다 위에처럼 테스트 오류 보고 최적 모델 찾는 방식 중 하나.

분산값처럼 기존에 알아야 하고, 이런거 필요 없음.

Vali set:

훈련셋으로 학습하고, 검증셋으로 오류값 계산.

검증셋 나눈 것 중에서 오류 가장 작은거 고르고,

테스트 오류를 직접 알 수 있음.

근데 데이터 많이 없음 훈련셋 자체가 줄어들어서 성능 안좋아짐.

Cross validation:

k개로 나눠서 교차 검증 하는 것.

각 폴드에서 모델 학습하고, 나머지 폴드에서 오류 계산하는 것.

폴드 전체에서 나온 오류값 평균내서 최적 모델 선택.

데이터 상대적으로 작을 때 유리한데, 계산 비용 높음.

Best subset selection에 적용해서 써보면,

모델 M\_k에 대해 교차검증 하고,

그럼 폴드마다 정확도 약간씩 달라지니까,

여기서 좋은 k값 해서 모델 다시 학습 🡺 결국 최적변수 개수 해서 다시 학습.

데이터 크기랑 변수 개수 많을수록 계산량 많아서 AIC, BIC, C\_p, Adjusted R^2 같은거 더 좋아했었는데,

꼼쀼타 좋아져서 교차검증이 좀 더 직관적으로 볼수 있기도 하고 해서 이것도 좋아라 해짐.

텍스트, 라인, 도표, 그래프이(가) 표시된 사진

자동 생성된 설명

위의 그림

Validation: 75% train set, 25% vali set 으로.

Cross validation: k = 10(10 폴드)

각각 이런식으로 뜨는데, BIC는 변수 4개, vali, Cross vali는 6개 동일.

근데 (cross) validation은 데이터 분할에 따라 결과 쪼금씩 달라짐. 그래서 최적 모델 선택에 좀 애매함.

그래서 쓰는 방식이

One-Standard-Error Rule: 테스트 오류 가장 작은거 뜬거에서 SE(standard Error)더한 범위에서 가장 변수 개수 작은 모델 선택하는 방식.

1. 각 모델에서 테스트 MSE 계산
2. 임계값 최소 테스트 오류에 SE 더한값으로 설정
3. 임계값 내에서 가장 작은 변수 개수 가진거 모델로 선택.

* 결국 성능 비슷한 경우, 가장 간단한 변수 개수 가진거 선택해서 overfitting 방지