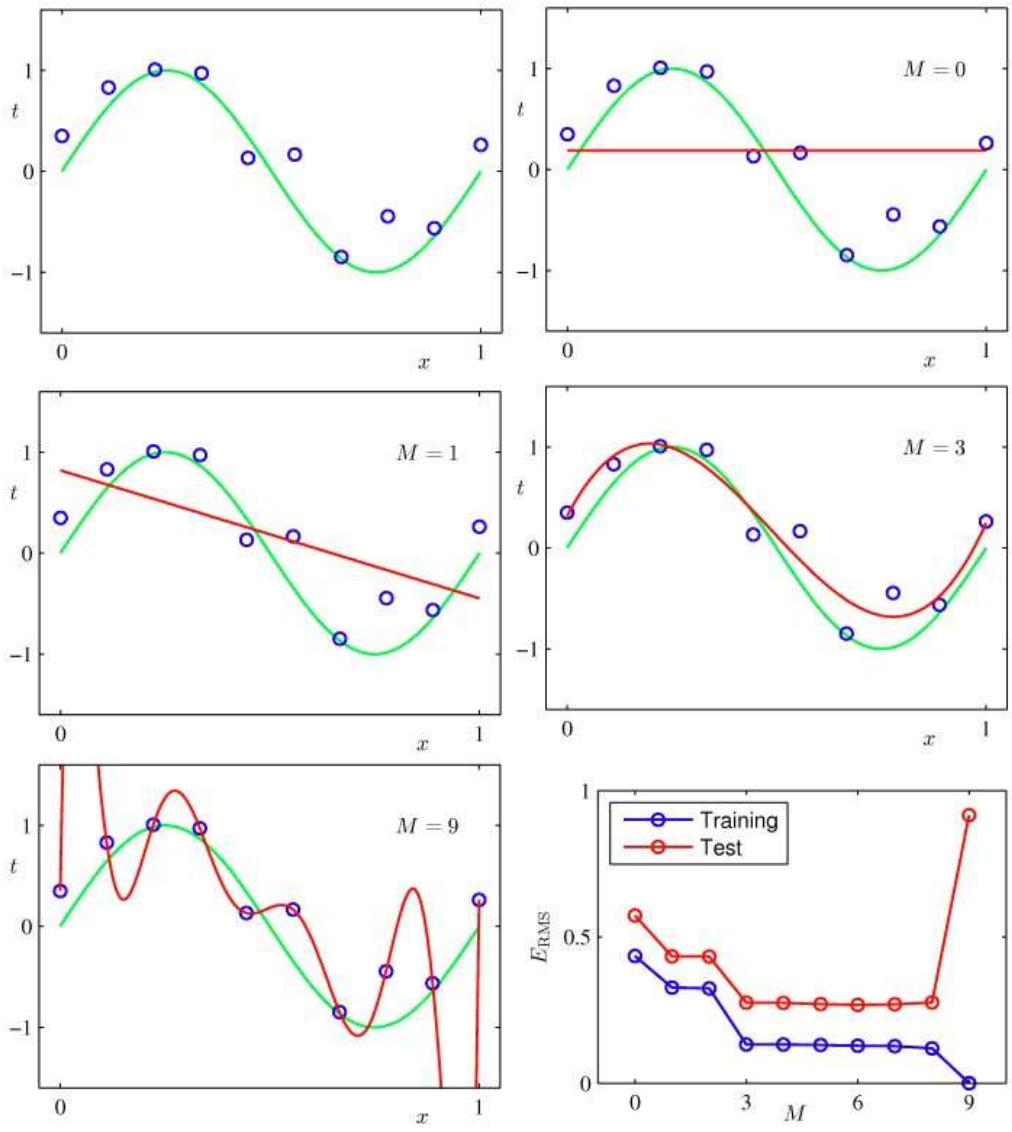
4. 과적합과 정규화

4.1. 과적합

기계학습에서 이제까지 우리가 훈련 또는 학습이라고 부른 과정은 적합(fitting)이라고도 부른다. 모형을 데이터에 맞추기(fit) 때문이다. 모형이 데이터에 잘 적합되면 예측력이 높아진다. 그러나 데이터에 지나치게 적합되면 예측력을 떨어질 수 있다. 이를 과적합(overfitting)이라 한다.

왜 과적합은 예측력을 떨어트릴까? 모든 데이터에는 패턴과 잡음이 섞여있다. 패턴은 과거의 데이터와 미래의 데이터에 반복되지만, 잡음은 그렇지 않다. 잡음이 생겨나는데는 다양한 이유가 있을 수 있다. 패턴 자체가 변하기 때문일 수도 있고, 우연이나 측정되지 않은 원인 때문일 수도 있다. 어쨌든 중요한 것은 잡음이 반복되지 않는다는 것이다.

아래 사례를 살펴보자.



이 데이터는 사인 함수에 약간의 잡음을 섞어 만든 것이다(왼쪽 위). 이 데이터를 다항 함수에 적합시켜 보자. 0차 모형 $y = b$ (오른쪽 위)는 모든 데이터를 똑같이 평균으로 예측하는 모형이다. 당

연히 잘 맞지 않는다. 1차 모형 $y = b + w_1 x$ (왼쪽 중간)은 우리가 이미 다룬 선형 모형이다. 실제 데이터는 비선형적인 패턴이기 때문에 선형 모형에는 잘 맞지 않는다. 3차 모형 $y = b + w_1 x + w_2 x^2$ (오른쪽 중간)은 실제 패턴에 가까운 형태로 적합되었다. 9차 모형 $y = b + w_1 x + w_2 x^2 + \ldots + w_8 x^8 + w_9 x^9$ (왼쪽 아래)는 모든 데이터를 관통하는 형태로 적합된다.

오차만을 보면 3차 모형에는 약간의 오차가 있지만 9차 모형은 오차가 전혀 없다. 그렇지만 직관적으로도 9차 모형은 지나치게 모형을 데이터에 끼워 맞춘 듯한 느낌이 든다. 실제로도 9차 모형은 실제 패턴인 초록선과도 동떨어져 있다. 만약 새로운 데이터가 들어온다면 9차 모형은 오히려 오차가 더 커질 것이다(오른쪽 아래)

위의 사례에서도 알 수 있듯이 대체로 단순한 모형은 과적합이 적지만 훈련용 데이터도 잘 설명하지 못한다. 복잡한 모형은 훈련용 데이터에 과적합되어 새로운 데이터를 잘 설명하지 못한다. 따라서 적절한 모형을 선정할 필요가 있다.

4.2. 교차 검증

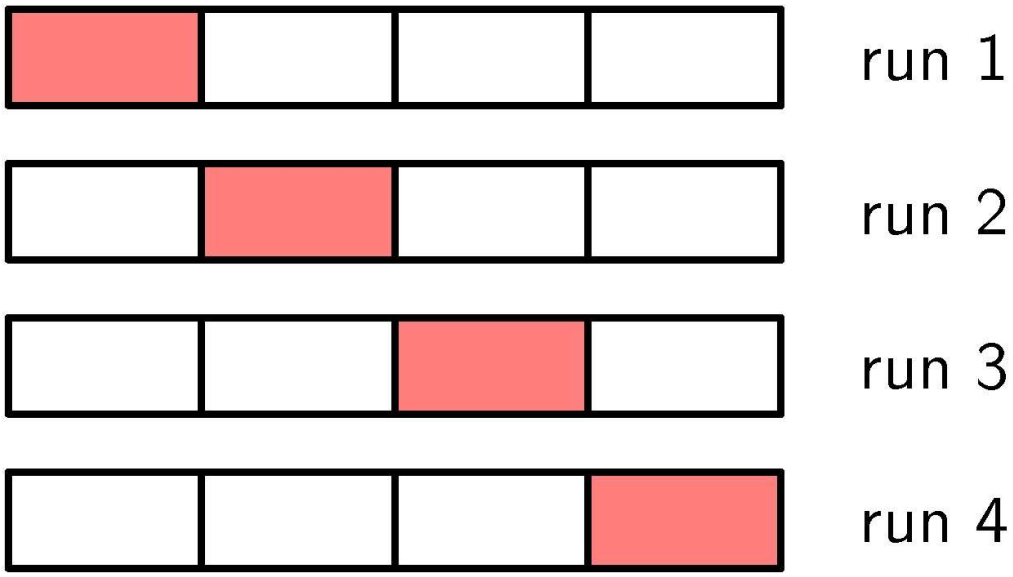
교차 검증(cross validation)은 모형이 잘 적합되었는지 알아보기 위해 훈련용 데이터와 별개의 테스트 데이터로 검증하는 것이다.

여러 모형들은 모형의 특성을 결정하는 하이퍼파라미터(hyperparameter)를 가지고 있다. 최근접 이웃의 경우 k가 하이퍼파라미터다. 어떤 하이퍼파라미터가 적절한지 알 수 없기 때문에 다양한 하이퍼파라미터를 시도해보고 테스트 데이터에서 성능이 가장 높은 하이퍼파라미터를 고른다. 그런데, 이렇게 할 경우 하이퍼파라미터가 테스트 데이터에 과적합될 우려가 있다.

한 가지 해결책은 훈련용 데이터의 일부를 다시 떼어내 하이퍼파라미터를 위한 검증용 데이터(validation dataset)을 만드는 것이다. 검증용 데이터를 한 세트만 만들면 역시 그 데이터세트에 과적합될 우려가 있으므로 검증용 데이터도 여러 세트로 만든다. 보통은 훈련용 데이터를 k등분한 다음에 그중에 한 세트를 검증용으로, 나머지를 훈련용으로 해서 검증하는 과정을 k번 반복한다. 이를 k-fold 교차검증이라고 한다.

경우에 따라서는 훈련용 데이터, 검증용 데이터, 테스트 데이터 셋으로 나누기도 한다. 는데, 어떤 하이퍼파라미터가 적절한지 결정하기 위해 검증용 데이터를 사용한다. 여러 가지 하이퍼파라미터로 훈련을 시킨 후 검증용 데이터로 가장 성능이 높은 최적의 하이퍼파라미터를 고른 후, 훈련용 데이터와 검증용 데이터를 합쳐 최적의 하이퍼파라미터의 모형에 학습시키고 다시 테스트 데이터로 테스트하는 것이다.

아래 그림은 4-fold 교차 검증을 나타낸 것이다. 데이터를 4등분으로 나누고 run 1에서는 1번 세트를 검증용으로 2~4번 세트를 훈련용으로 사용한다. run 2에서는 2번 세트를 검증용으로하고 1번과 3, 4번 세트를 훈련용으로 사용한다. 이 과정을 총 4회 반복하여 테스트 결과를 합쳐 가장 성능이 좋은 하이퍼파라미터를 고르는 것이다.



4.3. 정규화

선형 모형의 경우에는 하이퍼파라미터가 없기 때문에 모형의 특성을 조절해줄 수 없다. 이럴 때는 모형의 특성을 조절해줄 수 있는 하이퍼파라미터를 추가하여 과적합을 방지할 수 있다. 이런 테크닉을 정규화(regularization)이라고 한다. 선형 모형에서 많이 사용되는 정규화 방법은 손실 함수에 정규화 항(regularization term)을 추가하는 것이다. 이런 방법으로 라쏘(lasso), 릿지(ridge), 엘라스틱넷(elastic net)이 있다.

4.3.1. 라쏘

선형 모형에서 흔히 쓰이는 MSE 손실함수는 다음과 같은 식이다.

$$\frac{1}{N}\sum(y-\hat{y})^2$$

- N: 데이터 건수
- y: 실제 값
- $\hat{y}$: 예측 값

즉 실제 값과 예측 값의 차이(오차)를 제곱해서 평균낸 것이다. 선형 모형 $y = wx + b$을 적합시킨다는 것은 데이터에 대해 오차를 최소화시키는 파라미터 w, b를 찾는 것과 같다. 앞의 예에서 보았듯이 훈련용 데이터의 오차만을 최소화하면 과적합을 피하기 어렵다.

라쏘는 과적합을 피해주기 위해 손실 함수를 다음과 같이 정규화 항을 추가해준 것이다.

$$\frac{1}{N}\sum(y-\hat{y})^2 + \lambda\sum|w|$$

이 정규화항은 w의 절대값의 합에 가중치(λ, 람다)를 곱한 것이다. 왜 이런 항을 더해줄까? 대부분의 데이터는 매끄럽게 변하는 패턴을 가진다. 입력 x가 비슷하면 출력 y도 비슷하다. 그런데 w의

절대값이 커지면 입력이 x이 조금만 달라져도 wx가 크게 변하고 따라서 y도 크게 변한다. 이런 현상은 과적합의 한 가지 원인이 될 수 있다. 따라서 위와 같은 정규화 항을 추가하여 오차를 줄이는 동시에 파라미터도 작게 만들려는 것이다. 가중치 λ는 오차와 파라미터 중 어느 쪽에 더 무게를 둘지 결정하는 하이퍼파라미터 역할을 한다. λ는 항상 0보다 크거나 같은 값으로, $\lambda = 0$인 경우는 MSE와 동일하게 된다.

4.3.2. 릿지

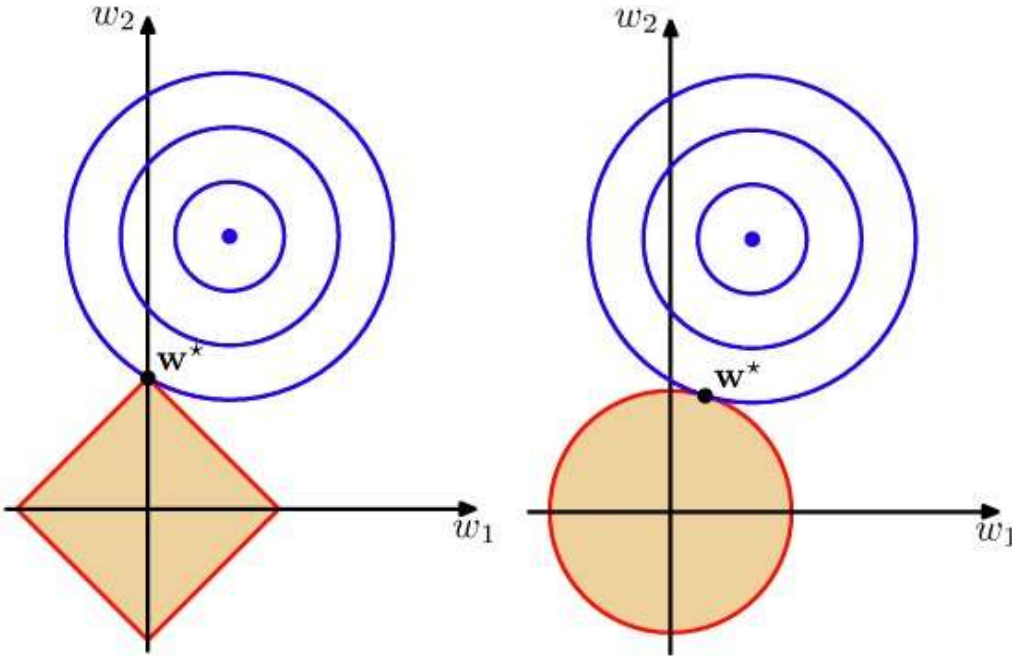
릿지는 라쏘와 거의 비슷하다. 다만 정규화 항에 파라미터의 절대값 대신 제곱을 사용한다는 점만 다르다.

$$\frac{1}{N}\sum(y-\hat{y})^2 + \lambda\sum|w|^2$$

라쏘는 1승, 릿지는 2승을 해준 것이기 때문에 이 둘을 각각 L1, L2라고 부르기도 한다. 인공신경망에서는 릿지를 weight decay라고 한다.

4.3.3. 라쏘와 릿지의 차이

절대값과 제곱에는 어떤 차이가 있을까? 그래프로 그려보면 절대값은 0에서 뾰족하게 꺾어지는 형태를, 제곱은 둥글게 변하는 형태를 띈다.



위의 그림에서 파란색 동심원은 파라미터에 따른 오차 제곱을 나타낸다. 파란점 부분이 오차가 가장 작은 부분이다. 빨간색 사각형과 원은 각각 라쏘 정규화항과 릿지 정규화항을 나타낸다. 두 도형이 만나는 w^*는 손실함수가 최소화되는 점을 나타낸다. 어느 쪽이나 오차 제곱만 사용한 경우보다 파라미터가 작아진다.

라쏘는 0에서 뾰족하게 꺾어지는 형태이기 때문에 손실함수의 최소값이 파라미터가 0인 경우가 많다. 그래서 라쏘를 사용하면 파라미터가 0으로 딱 떨어지는 경향이 있다. 위의 그림에서도 왼쪽 라

쏘의 경우 w^*는 $w_1 = 0$인 점인 것을 볼 수 있다.

선형 모형에서 파라미터가 0이 되면 그 파라미터에 해당하는 입력은 출력에 아무런 영향을 미치지 못하게 된다. 0에 무엇을 곱해도 0이 되기 때문이다. 따라서 라쏘는 특정 입력을 예측에서 아예 배제해버리는 효과를 나타낸다. 이런 것을 변수 선택(variable selection) 또는 특징 선택(feature selection)이라고 한다. 라쏘는 특정 변수를 예측에서 제외하기 때문에, 추후에도 예측을 위해 해당 변수를 수집하는 수고를 덜어준다는 장점이 있다.

다만 일반적으로는 라쏘에 비해 릿지가 좀 더 나은 성능을 보여주는 경향이 있다.

4.3.4. 엘라스틱넷

라쏘와 릿지에 각각의 장점이 있기 때문에 이 둘을 합친 엘라스틱 넷도 있다.

$$\frac{1}{N}\sum(y-\hat{y})^2 + \lambda(\sum \alpha|w| + (1-\alpha)|w|^2)$$

엘라스틱넷은 라쏘의 $|w|$와 릿지의 $|w|^2$을 모두 최소화하는 방법이다. 여기서 α는 라쏘와 릿지 중에 어느 쪽에 더 무게를 줄지 결정하는 하이퍼파라미터이다. α는 $[0, 1]$ 범위의 값으로 만약 $\alpha = 1$이라면 라쏘만 하는 것과 같고, $\alpha = 0$이면 릿지만 하는 것과 같다.

4.4. 하이퍼파라미터 탐색

대부분의 경우 하이퍼파라미터를 약간 달리하더라도 모형의 동작이 크게 변하지 않는다. 여러 가지 하이퍼파라미터를 시도하는데는 많은 시간이 들기 때문에, 하이퍼파라미터는 보통 크게 바꿔가며 탐색한다. 엘라스틱넷에서 λ와 처럼 $[0, \infty)$ 범위의 값을 가지는 경우 10^{-3}001, 10^{-2}, 10^{-1}, 10^0, 10^1, 10^2과 같은 식으로 10배씩 곱해가면서 시도한다.

4.4.1. 격자 탐색과 무작위 탐색

하이퍼파라미터가 여러 개일 때는 시도하는 방식에는 격자 탐색(grid search)과 무작위 탐색(random search) 2가지가 있다. 격자 탐색은 각각의 하이퍼파라미터에 시도할 값들을 정해놓고, 조합을 바꿔가면서 시도해보는 것이다. 엘라스틱넷의 α는 .3, .6으로 2가지를 시도하고 λ는 10^{-2}, 10^2으로 2가지를 시도한다면 다음과 같이 4가지 조합을 시도하는 것이다.

- $\alpha = .3, \lambda = 10^{-2}$
- $\alpha = .3, \lambda = 10^2$
- $\alpha = .6, \lambda = 10^{-2}$
- $\alpha = .6, \lambda = 10^2$

무작위 탐색은 각각의 하이퍼파라미터를 무작위로 시도해보는 방법이다. 예를 들어 똑같이 4번을 시도하더라도 α는 $[0, 1]$ 범위에서 무작위로, λ는 $[10^{-3}, 10^3]$ 범위에서 무작위로 시도하는 것이

다.

- $\alpha = .23, \lambda = 10^{2.2}$
- $\alpha = .79, \lambda = 10^{-1.5}$
- $\alpha = .46, \lambda = 10^{.06}$
- $\alpha = .15, \lambda = 10^{-.8}$

똑같이 4번을 시도하지만 격자 탐색은 각각의 하이퍼파라미터를 2개씩 시도하고, 무작위 탐색은 4개씩 시도한다. 무작위 탐색이 더 다양한 시도를 해볼 수 있기 때문에 격자 탐색보다 더 나은 경향 하이퍼파라미터를 찾게 해주는 경향이 있다.

4.5. 정규화와 교차 검증

4.5.1. 데이터 불러오기

```
data.url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/w
```

```
red.wine = read.csv(url(data.url), sep = ';')
```

```
head(red.wine)
```

```
fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
1           7.4             0.70        0.00            1.9     0.076
2           7.8             0.88        0.00            2.6     0.098
3           7.8             0.76        0.04            2.3     0.092
4          11.2             0.28        0.56            1.9     0.075
5           7.4             0.70        0.00            1.9     0.076
6           7.4             0.66        0.00            1.8     0.075
  free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
1                  11                   34  0.9978 3.51      0.56     9.4
2                  25                   67  0.9968 3.20      0.68     9.8
3                  15                   54  0.9970 3.26      0.65     9.8
4                  17                   60  0.9980 3.16      0.58     9.8
5                  11                   34  0.9978 3.51      0.56     9.4
6                  13                   40  0.9978 3.51      0.56     9.4
  quality
1       5
2       5
3       5
4       6
5       5
6       5
```

4.5.2. caret

```
library(caret)
```

4.5.3. 데이터 분할

```
set.seed(1234)
```

```
idx = createDataPartition(red.wine$quality, p=.8, list=F)
```

```
data.train = red.wine[idx, ]
data.test = red.wine[-idx, ]
```

4.5.4. 엘라스틱넷

격자 탐색

```
grid = expand.grid(.alpha=c(0, 1), .lambda=10^(-2:2))
```

```
grid
```

```
.alpha .lambda
1       0   1e-02
2       1   1e-02
3       0   1e-01
4       1   1e-01
5       0   1e+00
6       1   1e+00
7       0   1e+01
8       1   1e+01
9       0   1e+02
10      1   1e+02
```

```
model = train(
    quality ~ .,
    data = data.train,
    tuneGrid = grid,
    trControl = trainControl(method='cv', search='grid', number=5),
    method = 'glmnet')
```

```
model
```

```
glmnet

1281 samples
  11 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 1025, 1025, 1024, 1026, 1024
Resampling results across tuning parameters:

  alpha  lambda  RMSE       Rsquared
  0      1e-02   0.6583289  0.3500847
  0      1e-01   0.6587848  0.3506078
  0      1e+00   0.6903650  0.3453881
  0      1e+01   0.7820735  0.3354015
  0      1e+02   0.8108526  0.3324614
  1      1e-02   0.6586834  0.3496837
  1      1e-01   0.6829313  0.3259486
  1      1e+00   0.8146809        NaN
  1      1e+01   0.8146809        NaN
  1      1e+02   0.8146809        NaN

RMSE was used to select the optimal model using  the smallest value.
The final values used for the model were alpha = 0 and lambda = 0.01.
```

무작위 탐색

```
model = train(
    quality ~ .,
    data = data.train,
    tuneLength = 10,
    trControl = trainControl(method='cv', search='random', number=5),
    method = 'glmnet')
```

```
model
```

```
glmnet

1281 samples
  11 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 1024, 1025, 1025, 1025, 1025
Resampling results across tuning parameters:

  alpha       lambda       RMSE       Rsquared
  0.01532525  0.001842998  0.6595963  0.3482384
  0.05277008  3.023523604  0.7784200  0.3289008
  0.06167188  0.018538550  0.6593045  0.3487311
  0.11442353  0.006261803  0.6594562  0.3484752
  0.33475335  2.117664856  0.8144725        NaN
  0.43134921  0.563342381  0.7585711  0.2965496
  0.56273060  0.014917301  0.6588098  0.3499361
  0.70078405  2.128938029  0.8144725        NaN
  0.70376171  2.660648411  0.8144725        NaN
  0.81673573  0.001771485  0.6594215  0.3485671

RMSE was used to select the optimal model using  the smallest value.
The final values used for the model were alpha = 0.5627306 and lambda
 = 0.0149173.
```

4.5.5. 성능 검증

테스트용 데이터

```
y.pred = predict(model, data.test)
```

```
RMSE(data.test$quality, y.pred)
```

```
[1] 0.622837
```

```
R2(data.test$quality, y.pred)
```

```
[1] 0.3583355
```