06.02 기저함수 모형과 과최적화

1) 비선형 모형

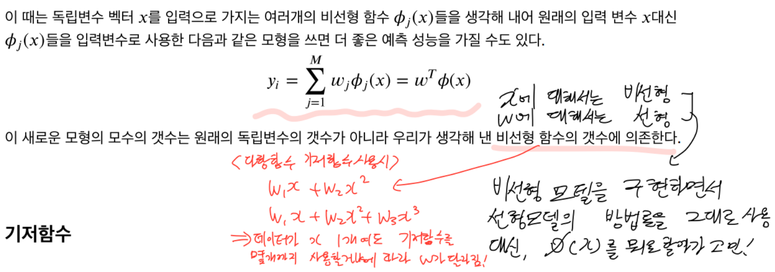
기본적인 선형회귀모형 : 입력변수의 선형조합 (~~$w^{T}x$~~) $w^T x$

선형회귀모형의 한계 : 비선형 데이터의 회귀모형을 만들 수 없음

대안 : 비선형 회귀모형

비선형 회귀모형 : x에 대해선 비선형, w에 대해선 선형 (2page 상단)

    *비선형 모델을 구현하면서 선형모델의 방법론을 그대로 사용.
    *대신, 어떤 비선형함수를 얼마나 사용할지가 중요


이 때는 독립변수 벡터 x를 입력으로 가지는 여러개의 비선형 함수 $\phi_j(x)$들을 생각해 내어 원래의 입력 변수 x대신 $\phi_j(x)$들을 입력변수로 사용한 다음과 같은 모형을 쓰면 더 좋은 예측 성능을 가질 수도 있다.

$$y_i = \sum_{j=1}^{M} w_j \phi_j(x) = w^T \phi(x)$$

x에 대해서는 비선형
w에 대해서는 선형

이 새로운 모형의 모수의 갯수는 원래의 독립변수의 갯수가 아니라 우리가 생각해 낸 비선형 함수의 갯수에 의존한다.

<다항함수 기저함수사용시>
$w_1x + w_2x^2$
$w_1x + w_2x^2 + w_3x^3$
⇒ 데이터가 x 1개여도 기저함수를 몇개까지 사용할거냐에 따라 w가 달라짐!

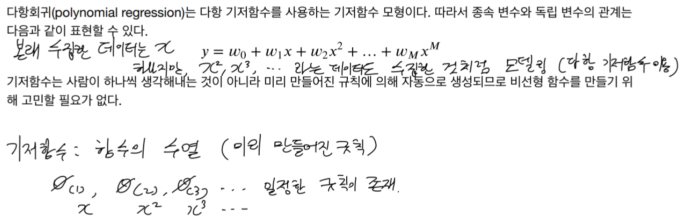
비선형 모델을 구현하면서 선형모델의 방법론을 그대로 사용
대신, $\phi(x)$를 뭐로 할지가 고민!

기저함수

비선형 함수 생성 => **기저함수 활용**

2) 기저함수

기저함수 : 함수의 수열 (규칙이 정해져 있어, 규칙에 따라 여러개의 비선형함수를 만들어낼 수 있음)
    ex) 다항 기저함수 (2page 하단)


다항회귀(polynomial regression)는 다항 기저함수를 사용하는 기저함수 모형이다. 따라서 종속 변수와 독립 변수의 관계는 다음과 같이 표현할 수 있다.

본래 수집한 데이터는 x

$$y = w_0 + w_1x + w_2x^2 + \ldots + w_M x^M$$

하지만, $x^2, x^3, \cdots$ 라는 데이터도 수집한 것처럼 모델링 (다항 기저함수 이용)

기저함수는 사람이 하나씩 생각해내는 것이 아니라 미리 만들어진 규칙에 의해 자동으로 생성되므로 비선형 함수를 만들기 위해 고민할 필요가 없다.

기저함수 : 함수의 수열 (미리 만들어진 규칙)

$\phi(1), \phi(2), \phi(3) \cdots$ 일정한 규칙이 존재.
$x \quad x^2 \quad x^3 \cdots$

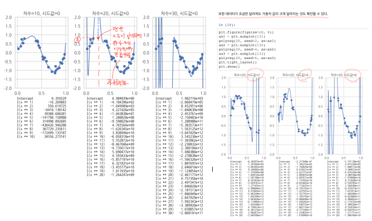
*비선형모형 : 가중치(모수) 갯수는 독립변수의 갯수가 아닌, 비선형함수의 갯수에 의존
    ex) 다항 기저함수 사용 시, 2차까지 하면 가중치 갯수는 3개, 10차까지 하면 가중치 갯수는 11개

기저함수 종류 : 체비셰프 다항식, 방사 기저함수, 삼각 기저함수, 시그모이드 기저함수

3) 과최적화

1. 과최적화의 이유
    1) 모형의 모수(parameter)가 과도하게 많거나
    2) 다중공산성

2. 과최적화가 만드는 문제
    1) non-training data 입력 시, 오차가 커짐 (cross-validation 오차)
    2) 샘플이 조금만 달라져도 가중치 계수의 값이 크게 달라짐 (추정의 불안정성)



06.03 교차검증

- in-sample testing VS outofsample testing
- 과최적화 ==>> 교차 검증 결과, 두 경우의 성능 testing 결과가 크게 다름(R^2)

1. sklearn 교차검증

1) 단순데이터 분리 " tarin_test_split( ) "

2) 교차검증

3) 교차검증 반복 " cross_val_score( ) "

`# 교안의 statsmodelsOLS 클래스 생성해, statsmodels 패키지 모형 객체 사용 가능하도록 변환`

K-폴드 교차검증 데이터 수가 적을 때, 데이터를 나눠 여러번 testing!

데이터의 수가 적은 경우에는 이 데이터 중의 일부인 검증 데이터의 수도 적기 때문에 검증 성능의 신뢰도가 떨어진다. 그렇다고 검증 데이터의 수를 증가시키면 학습용 데이터의 수가 적어지므로 정상적인 학습이 되지 않는다. 이러한 딜레마를 해결하기 위한 검증 방법이 **K-폴드(K-fold)** 교차검증 방법이다.

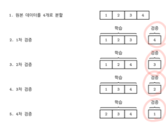
K-폴드 교차검증에서는 다음처럼 학습과 검증을 반복한다.

1. 전체 데이터를 K개의 부분 집합($\{D_1, D_2, \cdots, D_K\}$)으로 나눈다.
2. 데이터 $\{D_1, D_2, \cdots, D_{K-1}\}$를 학습용 데이터로 사용하여 회귀분석 모형을 만들고 데이터 $\{D_K\}$로 교차검증을 한다.
3. 데이터 $\{D_1, D_2, \cdots, D_{K-2}, D_K\}$를 학습용 데이터로 사용하여 회귀분석 모형을 만들고 데이터 $\{D_{K-1}\}$로 교차검증을 한다.

   ⋮

4. 데이터 $\{D_2, \cdots, D_K\}$를 학습용 데이터로 사용하여 회귀분석 모형을 만들고 데이터 $\{D_1\}$로 교차검증을 한다.

이렇게 하면 총 K개의 모형과 K개의 교차검증 성능이 나온다. 이 K개의 교차검증 성능을 평균하여 최종 교차검증 성능을 계산한다.



학습데이터의 R^2, 검증 데이터의 R^2

(result 객체 활용) (직접계산)

K-Fold 교차검증

- 데이터 수가 적을 때, 데이터를 나눠 여러번 testing 진행

cross_val_score()

- 11page

벤치마크 검증 데이터

- 11page

In [8]:

```
from sklearn.base import BaseEstimator, RegressorMixin
import statsmodels.formula.api as smf
import statsmodels.api as sm

class StatsmodelsOLS(BaseEstimator, RegressorMixin):
    def __init__(self, formula):
        self.formula = formula
        self.model = None
        self.data = None
        self.result = None

    def fit(self, dfX, dfy):
        self.data = pd.concat([dfX, dfy], axis=1)
        self.model = smf.ols(self.formula, data=self.data)
        self.result = self.model.fit()

    def predict(self, new_data):
        return self.result.predict(new_data)
```

Statsmodels OLS 클래스 생성해 sklearn 에서도 statsmodels 패키지 모형 객체 사용가능!

이 래퍼 클래스와 `cross_val_score` 명령을 사용하면 교차검증 성능 값을 다음처럼 간단하게 계산할 수 있다.

In [9]:

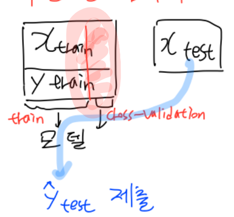
```
from sklearn.model_selection import cross_val_score

model = StatsmodelsOLS("MEDV ~ " + "+".join(boston.feature_names))
cv = KFold(5, shuffle=True, random_state=0)
cross_val_score(model, dfX, dfy, scoring="r2", cv=cv)
```

↳ R^2 값으로 성능평가 해라!

Out[9]:

```
array([0.58922238, 0.77799144, 0.66791979, 0.6680163 , 0.83953317])
```

캐글 등 컴피티션 에서, 정답인 test data의 y값은 주지 않는다.



06.04 다중공선성과 변수 선택

1) overfitting 주요원인 2가지

1) 모수 갯수가 너무 많아서

2) 다중공선성 (1page 하단)
 *x1과 x2가 거의 같은 데이터라면, 모형이 어떻게든 이를 구분하려 overfitting하게 됨

다중공선성 → 조건수↑
→ overfitting.
→ 독립변수 공분산행렬이 fullrank 여야 한다는 가정 침해.

3) 다중공선성에 따른 overfitting 방지법 : 독립변수 제거

- VIF 활용해 의존적인 변수 삭제 (VIF, Variance Inflation Factor)
- PCA를 활용한 의존적인 변수 삭제
- 정규화(regularized) 방법 사용

2) VIF

다중 공선성을 없애는 가장 기본적인 방법은 다른 독립변수에 의존하는 변수를 없애는 것이다. 가장 의존적인 독립변수를 선택하는 방법으로는 VIF(Variance Inflation Factor)를 사용할 수 있다. VIF는 독립변수를 다른 독립변수로 선형회귀한 성능을 나타낸 것이다. i번째 변수의 VIF는 다음과 같이 계산한다.

클수록 i는 쓸모없는 feature x_1, x_2, x_3, x_4

$$\text{VIF}_i = \frac{\sigma^2}{(n-1)\text{Var}[X_i]} \cdot \boxed{\frac{1}{1-R_i^2}}$$

feature 종류

R_i^2 ⇒ 클수록 쓸모없음!

i번수를 다른 변수로 예측시 성능

여기에서 R_i^2는 다른 변수로 i번째 변수를 선형회귀한 성능(결정 계수)이다. 다른 변수에 의존적일 수록 VIF가 커진다.

StatsModels에서는 `variance_inflation_factor` 명령으로 VIF를 계산한다.

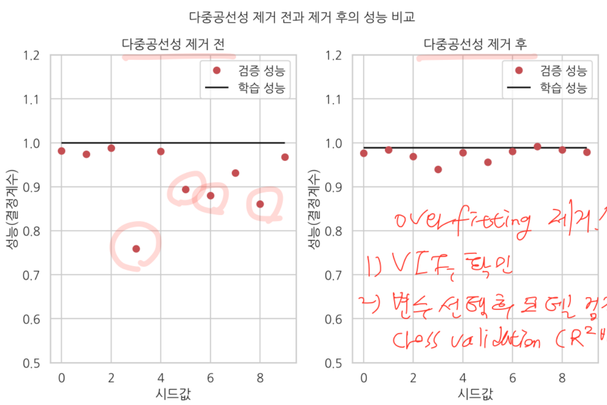
```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(
    dfX.values, i) for i in range(dfX.shape[1])]
vif["features"] = dfX.columns
vif
```

Out[6]:

	VIF Factor	features
0	12425.514335	GNPDEFL
1	10290.435437	GNP
2	136.224354	UNEMP
3	39.983386	ARMED
4	101193.161993	POP
5	84709.950443	YEAR

이 3개만 사용해도 됨.

상관계수와 VIF를 사용하여 독립 변수를 선택하면 GNP, ARMED, UNEMP 세가지 변수만으로도 비슷한 수준의 성능이 나온다는 것을 알 수 있다.



overfitting 제거!
1) VIF 확인
2) 변수 선택후 모델 검증
cross validation (R^2비교)