Python으로 머신러닝 입문

세션장: 구은아, 김은기

단순선형회귀

LinearRegression 클래스

sklearn.linear_model.LinearRegression

class sklearn.linear_model.LinearRegression(*, *fit_intercept=True*, *normalize='deprecated'*, *copy_X=True*, *n_jobs=None*, *positive=False*) [source]

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

LinearRegression 파라미터

파라미터 이름	설명
fil_intercept	절편을 계산할지 말지를 결정 디폴트는 True, 절편을 계산
normalize	회귀 수행 전에 데이터 세트를 정규화할 것인지 결정 디폴트는 False

회귀에서의 주의점

회귀 평가 지표 적용

cross_val_score나 GridSearchCV와 같은 함수에는 평가 지표를 설정하는 scoring 파라미터가 있습니다. scoring에 할당된 평가지표를 최대화하는 방향으로 학습을 합니다.

그런데 회귀의 대표적인 평가지표인 MAE나 MSE는 클수록 모델의 성능이 나빠진다는 의미입니다. 따라서 MAE나 MSE를 사용할 때는 -1을 곱한 값을 평가지표로 사용합니다.

평가 지표	scoring 적용 값
MAE	neg_mean_absolute_error
MSE	neg_mean_squared_error
R^2	r2

경사하강법

경사하강법

```
# w1 과 w0 를 업데이트 할 w1_update, w0_update를 반환.
def get_weight_updates(w1, w0, X, y, learning_rate=0.01):
    N = len(y)
    # 먼저 w1_update, w0_update를 각각 w1, w0의 shape와 동일한 크기를 가
진 0 값으로 초기화
    w1_update = np.zeros_like(w1)
    w0_update = np.zeros_like(w0)
    # 예측 배열 계산하고 예측과 실제 값의 차이 계산
    y_pred = np.dot(X, w1.T) + w0
    diff = y-y_pred

    # w0_update를 dot 행렬 연산으로 구하기 위해 모두 1값을 가진 행렬 생
성
    w0_factors = np.ones((N,1))

    # w1과 w0을 업데이트할 w1_update와 w0_update 계산
    w1_update = -(2/N)*learning_rate*(np.dot(X.T, diff))
    w0_update = -(2/N)*learning_rate*(np.dot(w0_factors.T, diff))

    return w1_update, w0_update
```

$$\frac{\partial R(w)}{\partial w_1} = \frac{2}{N}\sum_{i=1}^{N} -x_t * (y_i - (w_0 + w_1 x_i)) = -\frac{2}{N}\sum_{i=1}^{N} x_i * (\text{실제값}_i - \text{예측값}_i)$$

$$\frac{\partial R(w)}{\partial w_0} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (w_0 + w_1 x_i)) = -\frac{2}{N}\sum_{i=1}^{N} (\text{실제값}_i - \text{예측값}_i)$$

경사하강법

```
# w1 과 w0 를 업데이트 할 w1_update, w0_update를 반환.
def get_weight_updates(w1, w0, X, y, learning_rate=0.01):
    N = len(y)
    # 먼저 w1_update, w0_update를 각각 w1, w0의 shape와 동일한 크기를 가
진 0 값으로 초기화
    w1_update = np.zeros_like(w1)
    w0_update = np.zeros_like(w0)
    # 예측 배열 계산하고 예측과 실제 값의 차이 계산
    y_pred = np.dot(X, w1.T) + w0
    diff = y-y_pred

    # w0_update를 dot 행렬 연산으로 구하기 위해 모두 1값을 가진 행렬 생
성
    w0_factors = np.ones((N,1))

    # w1과 w0을 업데이트할 w1_update와 w0_update 계산
    w1_update = -(2/N)*learning_rate*(np.dot(X.T, diff))
    w0_update = -(2/N)*learning_rate*(np.dot(w0_factors.T, diff))

    return w1_update, w0_update
```

$$\frac{\partial R(w)}{\partial w_1} = \frac{2}{N}\sum_{i=1}^{N} -x_t * (y_i - (w_0 + w_1 x_i)) = -\frac{2}{N}\sum_{i=1}^{N} x_i * (\text{실제값}_i - \text{예측값}_i)$$

$$\frac{\partial R(w)}{\partial w_0} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (w_0 + w_1 x_i)) = -\frac{2}{N}\sum_{i=1}^{N} (\text{실제값}_i - \text{예측값}_i)$$

경사하강법

```
# w1 과 w0 를 업데이트 할 w1_update, w0_update를 반환.
def get_weight_updates(w1, w0, X, y, learning_rate=0.01):
    N = len(y)
    # 먼저 w1_update, w0_update를 각각 w1, w0의 shape와 동일한 크기를 가
진 0 값으로 초기화
    w1_update = np.zeros_like(w1)
    w0_update = np.zeros_like(w0)
    # 예측 배열 계산하고 예측과 실제 값의 차이 계산
    y_pred = np.dot(X, w1.T) + w0
    diff = y-y_pred

    # w0_update를 dot 행렬 연산으로 구하기 위해 모두 1값을 가진 행렬 생
성
    w0_factors = np.ones((N,1))

    # w1과 w0을 업데이트할 w1_update와 w0_update 계산
    w1_update = -(2/N)*learning_rate*(np.dot(X.T, diff))
    w0_update = -(2/N)*learning_rate*(np.dot(w0_factors.T, diff))

    return w1_update, w0_update
```

$$\frac{\partial R(w)}{\partial w_1} = \frac{2}{N}\sum_{i=1}^{N} -x_t * (y_i - (w_0 + w_1 x_i)) = -\frac{2}{N}\sum_{i=1}^{N} x_i * (\text{실제값}_i - \text{예측값}_i)$$

$$\frac{\partial R(w)}{\partial w_0} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (w_0 + w_1 x_i) = -\frac{2}{N}\sum_{i=1}^{N} (\text{실제값}_i - \text{예측값}_i)$$

경사하강법

```
# w1 과 w0 를 업데이트 할 w1_update, w0_update를 반환.
def get_weight_updates(w1, w0, X, y, learning_rate=0.01):
    N = len(y)
    # 먼저 w1_update, w0_update를 각각 w1, w0의 shape와 동일한 크기를 가
진 0 값으로 초기화
    w1_update = np.zeros_like(w1)
    w0_update = np.zeros_like(w0)
    # 예측 배열 계산하고 예측과 실제 값의 차이 계산
    y_pred = np.dot(X, w1.T) + w0
    diff = y-y_pred

    # w0_update를 dot 행렬 연산으로 구하기 위해 모두 1값을 가진 행렬 생
성
    w0_factors = np.ones((N,1))

    # w1과 w0을 업데이트할 w1_update와 w0_update 계산
    w1_update = -(2/N)*learning_rate*(np.dot(X.T, diff))
    w0_update = -(2/N)*learning_rate*(np.dot(w0_factors.T, diff))

    return w1_update, w0_update
```

$$\frac{\partial R(w)}{\partial w_1} = \frac{2}{N}\sum_{i=1}^{N} -x_t * (y_i - (w_0 + w_1 x_i) = -\frac{2}{N}\sum_{i=1}^{N} x_i * (\text{실제값}_i - \text{예측값}_i)$$

$$\frac{\partial R(w)}{\partial w_0} = \frac{2}{N}\sum_{i=1}^{N} -(y_i - (w_0 + w_1 x_i)) = -\frac{2}{N}\sum_{i=1}^{N} 1 * (\text{실제값}_i - \text{예측값}_i)$$

경사하강법

```
# 입력 인자 iters로 주어진 횟수만큼 반복적으로 w1과 w0를 업데이트 적용
함.
def gradient_descent_steps(X, y, iters=10000):
    # w0와 w1을 모두 0으로 초기화.
    w0 = np.zeros((1,1))
    w1 = np.zeros((1,1))

    # 인자로 주어진 iters 만큼 반복적으로 get_weight_updates() 호출하여
w1, w0 업데이트 수행.
    for ind in range(iters):
        w1_update, w0_update = get_weight_updates(w1, w0, X, y, learning
_rate=0.01)
        w1 = w1 - w1_update
        w0 = w0 - w0_update

    return w1, w0
```

$$w_1 + \eta \frac{2}{N} \sum_{i=1}^{N} x_i * (\text{실제값}_i - \text{예측값}_i)$$

$$w_0 + \eta \frac{2}{N} \sum_{i=1}^{N} (\text{실제값}_i - \text{예측값}_i)$$

확률적 경사하강법

```
def stochastic_gradient_descent_steps(X, y, batch_size=10  iters=1000):
    w0 = np.zeros((1,1))
    w1 = np.zeros((1,1))
    prev_cost = 100000
    iter_index =0

    for ind in range(iters):
        np.random.seed(ind)
        # 전체 X, y 데이터에서 랜덤하게 batch_size만큼 데이터 추출하여 s
ample_X, sample_y로 저장
        stochastic_random_index = np.random.permutation(X.shape[0])
        sample_X = X[stochastic_random_index[0:batch_size]]
        sample_y = y[stochastic_random_index[0:batch_size]]
        # 랜덤하게 batch_size만큼 추출된 데이터 기반으로 w1_update, w0_u
pdate 계산 후 업데이트
        w1_update, w0_update = get_weight_updates(w1, w0, sample_X, samp
le_y, learning_rate=0.01)
        w1 = w1 - w1_update
        w0 = w0 - w0_update

    return w1, w0
```

다항회귀

PolynomialFeatures 클래스

sklearn.preprocessing.PolynomialFeatures

class sklearn.preprocessing.PolynomialFeatures(*degree=2, *, interaction_only=False, include_bias=True, order='C'*) [source]

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

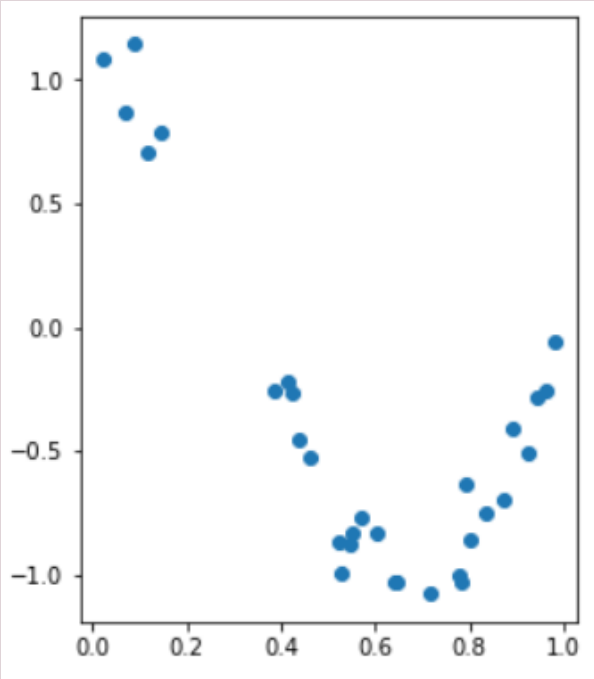
다항회귀 실습

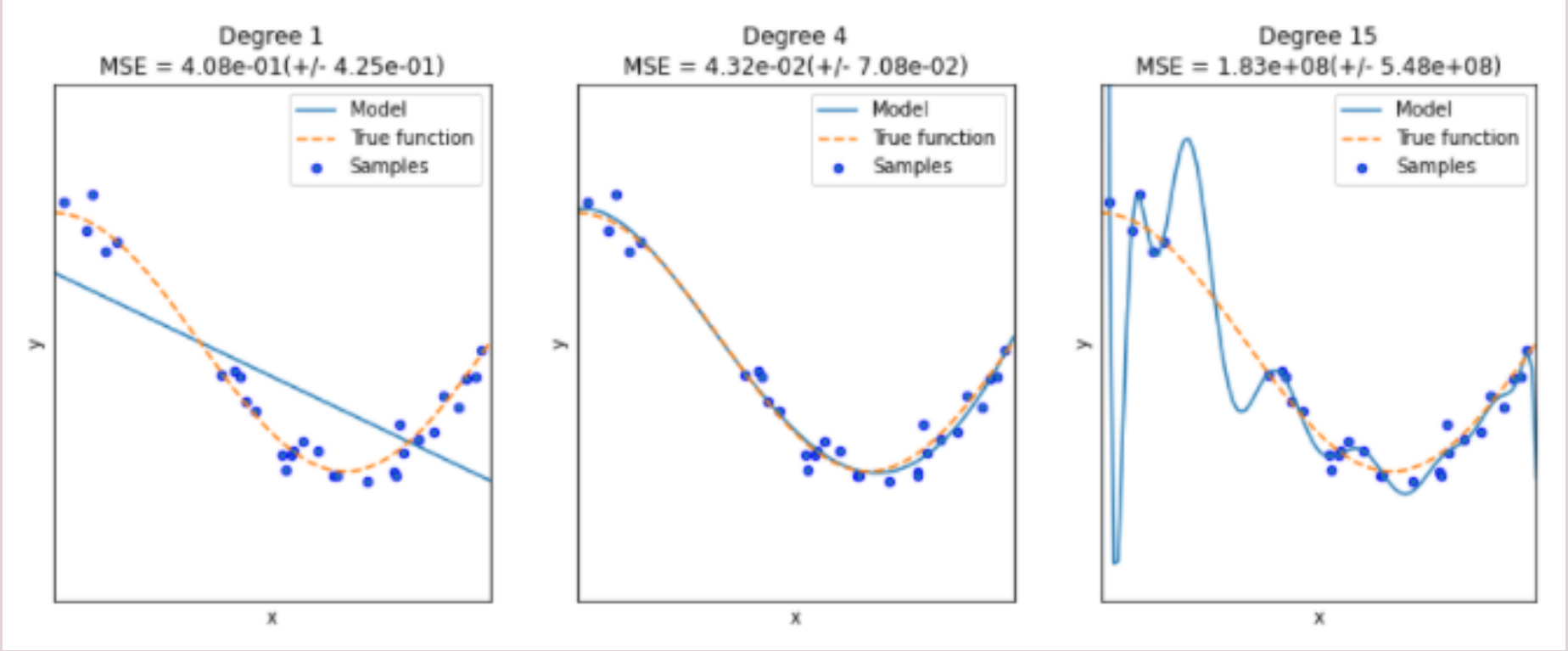
X_1	X_2
0	1
2	3

PolynomialFeatures(degree=2)

1	X_1	X_2	${X_1}^2$	$X_1 * X_2$	${X_2}^2$
1	0	1	0	0	1
1	2	3	4	6	9

과대적합

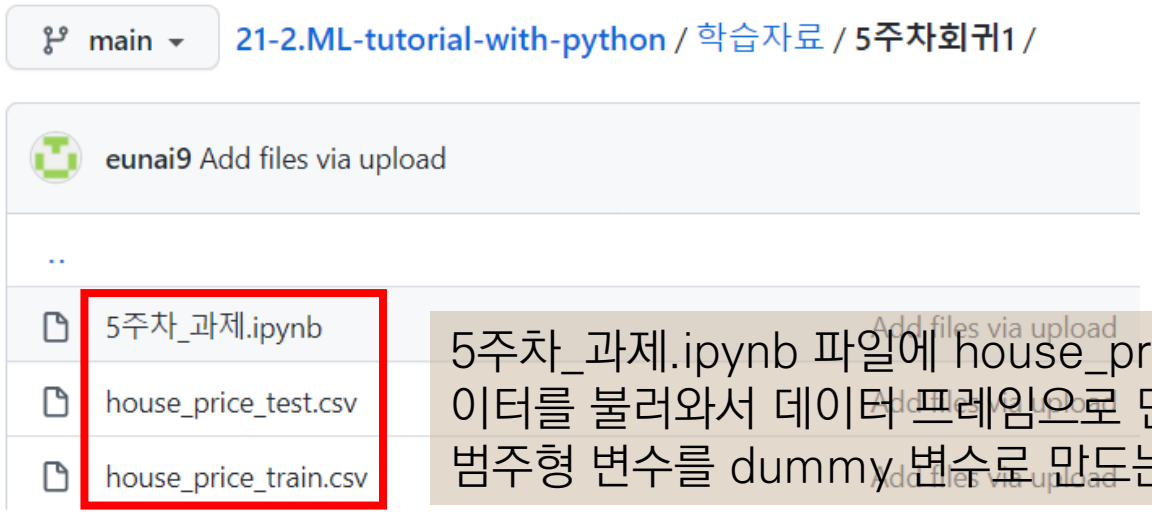




보스턴 주택 가격 예측 실습

과제 안내

과제: 주택 가격 예측



5주차_과제.ipynb 파일에 house_price_train, house_price_test 데이터를 불러와서 데이터 프레임으로 만들어 두었습니다.
범주형 변수를 dummy 변수로 만드는 전처리만 되어 있습니다.

1) EDA: 결측치 확인, 데이터 간 상관관계 확인, 데이터 분포 확인 등은 필수로 시행하고 이외의 EDA 방법으로 데이터를 파악해봅시다. 문제가 있을 경우, 이를 해결해봅시다.
2) LinearRegression 클래스를 사용하여 회귀 모델을 만들고, test 데이터의 주택 가격을 예측해봅시다.
3) train 데이터를 train과 validation set로 나누어 train으로는 학습을 하고, validation으로는 성능을 평가해봅시다. 성능 평가지표는 rmse를 이용하세요. 특정 피처를 삭제하거나 변형하면서 rmse를 줄여봅시다.

수고하셨습니다!
과제 열심히 하시고 다음 주에 뵈어요~