기계학습 9주차 과제 보고서

12191656 이채연

P1. 본 실습에서 사용한 데이터를 활용하여 MLE를 활용한 log-likelihood linear regression 코드를 작성

$$\ell(w) = \sum_{i=1}^{N} \log\left[\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}} \exp\left(-\frac{1}{2\sigma^2}(y_i - w^T x_i)^2\right)\right]$$

$$= -\frac{1}{2\sigma^2} RSS(w) - \frac{N}{2}\log(2\pi\sigma^2)$$

w에 대한 MLE를 하기 위해서는 RSS를 minimize 하면 된다.

$$RSS(w) = \|\varepsilon\|_2^2 = \|y - Xw\|^2 = (y-Xw)^T(y-Xw)$$

$$= y^Ty - y^TXw - w^TX^Ty + w^TX^TXw$$

RSS(w)의 gradient가 zero일때 RSS(w)를 minimization 할 수 있다.

$$\frac{\partial RSS(w)}{\partial w} = \frac{\partial}{\partial w}(y^Ty - y^TXw - w^TX^Ty + w^TX^TXw)$$

$$= -2X^Ty + 2X^TXw = 0 \Rightarrow X^Ty = X^TXw$$

$$\boxed{\hat{w} = (X^TX)^{-1}X^Ty}$$

즉, MLE를 활용한 log-likelihood linear regression을 할 때 최종적으로 estimation한 w값이 다음과 같다. 따라서 fit 함수에서 W에 estimation한 W값을 넣어주어 모델을 fitting하고, predict 함수에서 predict할 때 XW 식을 사용하여 predict한다.

구현 코드는 다음과 같다.

```
class LinearRegression:
    def fit(self, X, Y):
        X = np.array(X).reshape(-1, 1)
        Y = np.array(Y).reshape(-1, 1)
        self.parameter_cache = []
        self.W = np.matmul(np.linalg.inv(np.matmul(X.T, X)), np.matmul(X.T, Y))

    def predict(self, X):
        product = np.matmul(np.array(X).reshape(-1,1), self.W)
        return product.reshape(-1)
```

P2. 본 실습에서 사용한 데이터에 MLE와 gradient 기반 linear regression을 각각 적용하여 결과 비교

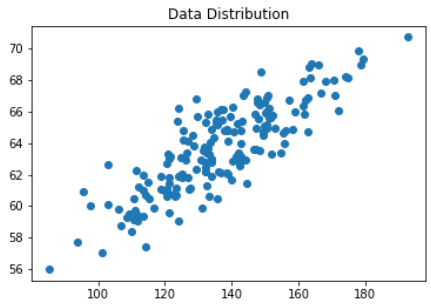
- MLE 기반 linear regression

Raw 데이터를 사용해서 10000개의 raw data중 9820개를 Train dataset으로 사용하고 180개를 Test dataset으로 사용하였다.

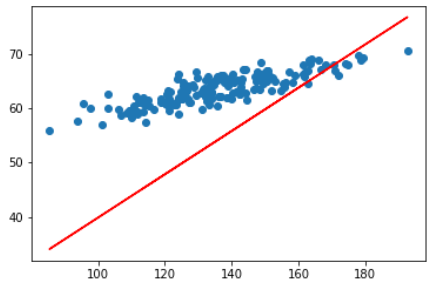
```
reg = LinearRegression()
x = data['Weight']
y = data['Height']
params = reg.fit(x[:-180], y[:-180])
pred = reg.predict(np.array(x[-180:]))
plt.scatter(x[-180:], y[-180:])
plt.plot(x[-180:], pred, 'red')
plt.savefig('plot_fit_mle.png')
```

✓ **시각화 결과**

모델 평가에 사용한 test data distribution은 다음과 같다.



Training한 후 모델이 test 데이터셋에 대해 예측한 결과는 다음과 같다. 빨간 선이 MLE 기반 linear regression 모델의 예측 값이고 파란 점이 실제 값이다.



- Gradient 기반 linear regression

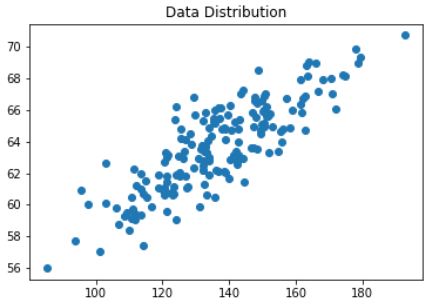
```
class LinearRegression:
    def fit(self, X, Y):
        X = np.array(X).reshape(1, -1)
        Y = np.array(Y).reshape(1, -1)
        x_shape = X.shape
        num_var = x_shape[0]
        self.parameter_cache = []
        self.weight = np.random.normal(0,1,(num_var,1))
        self.bias = np.random.rand(1)
        self.num_iteration = 50

        for t in range(self.num_iteration):
            N = x_shape[1]
            self.delta_W = 2/N*(np.sum(np.multiply(((np.matmul(self.weight, X)+self.bias)-Y), X)))
            self.delta_bias = 2/N*(np.sum(((np.matmul(self.weight, X)+self.bias)-Y)))
            self.weight -= 0.1*self.delta_W
            self.bias -= 0.1*self.delta_bias
            self.parameter_cache.append(np.array((self.weight, self.bias)))

        return self.weight, self.bias, self.parameter_cache

    def predict(self, X, idx):
        product = np.matmul(self.parameter_cache[idx][0], np.array(X).reshape(1,-1))+self.parameter_cache[idx][1]
        return product.reshape(-1)
```

Gradient 기반 linear regression에서는 weight를 정규분포로부터 (num_var, 1) shape의 array형태로 무작위 샘플링하였다. Bias는 0~1의 균일분포 표준정규분포 난수로 지정하였다. 그리고 50번 반복하면서 cost(손실)을 weight와 bias로 각각 편미분하여 gradient를 산출하고 learning rate를 각각 0.1로 해서 weight와 bias를 업데이트하였다.

Predict할때는 WX+b 식을 사용한다. iteration마다 model이 data에 fitting되는 것을 관찰하기 위해 parameter_cache[idx][0]으로 해당 idx번째(0~49) iteration에서의 weight를 불러오고 parameter_cache[idx][1]로 해당 idx번째(0~49) iteration에서의 bias를 불러왔다.

- ✓ **시각화 결과**

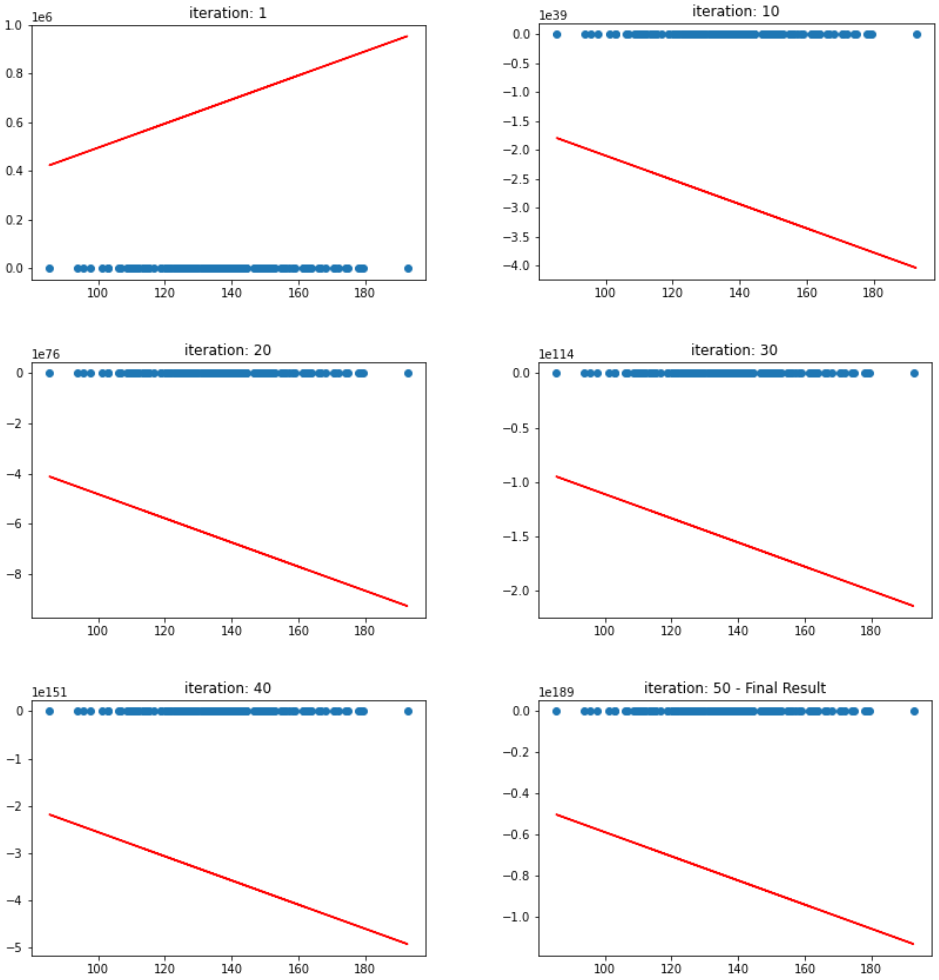
  모델 평가에 사용한 test data distribution은 다음과 같다.



Training한 후 모델이 test 데이터셋에 대해 예측한 값을 iteration 1, 10, 20, 30, 40, 50번째마다 출력하기 위해 idx를 0, 9, 19, 29, 39, 49로 지정하여 그때의 weight값과 bias값을 parameter_cache에서 가져와서 예측을 하도록 하였다.

```
reg = LinearRegression()

x = data['Weight']
y = data['Height']

params = reg.fit(x[:-180], y[:-180])
pred0 = reg.predict(np.array(x[-180:]), 0)
pred1 = reg.predict(np.array(x[-180:]), 9)
pred2 = reg.predict(np.array(x[-180:]), 19)
pred3 = reg.predict(np.array(x[-180:]), 29)
pred4 = reg.predict(np.array(x[-180:]), 39)
pred5 = reg.predict(np.array(x[-180:]), 49)
```

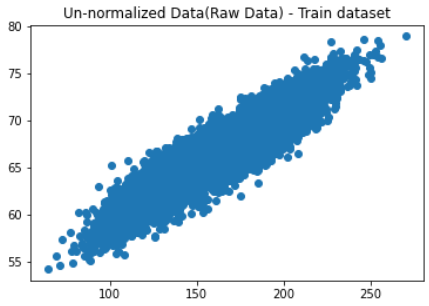
Training한 model이 test 데이터셋에 대해 예측한 결과를 시각화 하면 다음과 같다. 빨간 선이 Gradient기반 linear regression 모델의 예측 값이고 파란 점이 실제 값이다. Iteration 1, 10, 20, 30, 40, 50마다 모델 fitting 결과를 보면 다음과 같다.

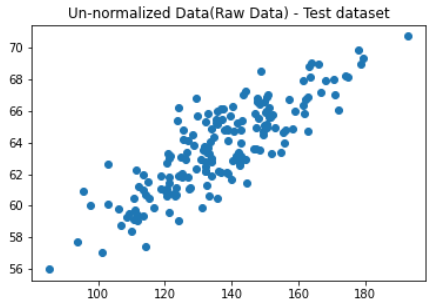


P3. 본 실습에서 사용한 raw 데이터를 min-max normalization, Z-score normalization 각각 적용하고 비교

- Raw Data Distribution

정규화를 하지 않은 raw data Distribution은 다음과 같다.



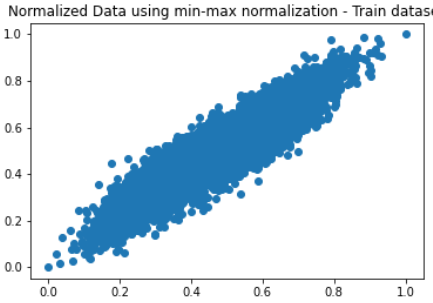


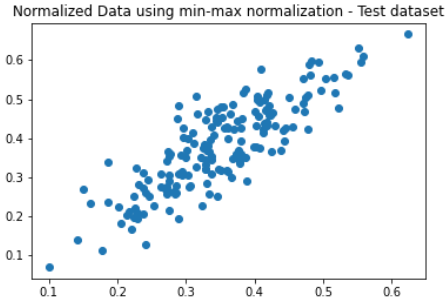
- Min-max normalization을 적용했을 때 Data Distribution

min()과 max()함수를 이용해서 min-max normalization을 구현했다.

```
x = (data['Weight']-data['Weight'].min()) / (data['Weight'].max() - data['Weight'].min())
y = (data['Height']-data['Height'].min()) / (data['Height'].max() - data['Height'].min())

plt.scatter(x[-180:], y[-180:])
plt.title("Normalized Data using min-max normalization")
plt.savefig('min_max.png')
```

Min-max normalization을 적용한 data distribution은 다음과 같다.





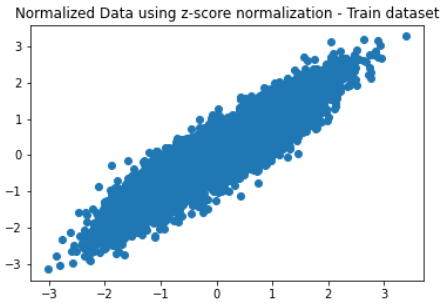
특징량이 범위가 [0,1]인 실수 값인 것을 확인할 수 있다.

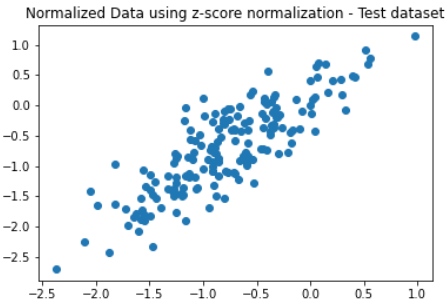
- Z-score normalization을 적용했을 때 Data Distribution

mean()과 std()함수를 이용해서 Z-score normalization을 구현했다.

```
x = (data['Weight']-data['Weight'].mean())/data['Weight'].std()
y = (data['Height']-data['Height'].mean())/data['Height'].std()

plt.scatter(x[-180:], y[-180:])
plt.title("Normalized Data using z-score normalization")
plt.savefig('z_score.png')
```

Z-score normalization을 적용한 data distribution은 다음과 같다.





표준 편차상에 데이터가 어떤 위치를 차지하는지 알 수 있다.

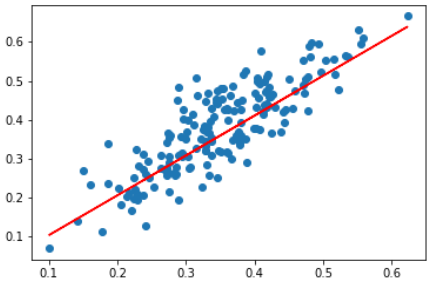
- Min-max normalization 기법을 적용한 MLE 기반 linear regression

MLE 기반 linear regression 클래스는 P1과 동일하고, fit함수에 x와 y 데이터를 넣고 훈련하기 전에 x와 y데이터에 min-max normalization 기법을 적용하였다.

```
reg = LinearRegression()
x = (data['Weight']-data['Weight'].min()) / (data['Weight'].max() - data['Weight'].min())
y = (data['Height']-data['Height'].min()) / (data['Height'].max() - data['Height'].min())
params = reg.fit(x[:-180], y[:-180])
pred = reg.predict(np.array(x[-180:]))

plt.scatter(x[-180:], y[-180:])
plt.plot(x[-180:], pred, 'red')
plt.savefig('plot_fit_mle_minmax.png')
```

- ✓ **시각화 결과**

  Training한 후 모델이 test 데이터셋에 대해 예측한 결과는 다음과 같다. 빨간 선이 min-max normalization 기법을 적용한 MLE기반 linear regression 모델의 예측값이고 파란 점이 실제값이다.



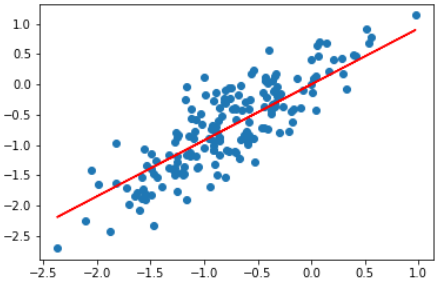
- Z-score normalization 기법을 적용한 MLE 기반 linear regression

MLE 기반 linear regression 클래스는 P1과 동일하고, fit함수에 x와 y 데이터를 넣고 훈련하기 전에 x와 y데이터에 Z-score normalization 기법을 적용하였다.

```
reg = LinearRegression()
x = (data['Weight']-data['Weight'].mean())/data['Weight'].std()
y = (data['Height']-data['Height'].mean())/data['Height'].std()
params = reg.fit(x[:-180], y[:-180])
pred = reg.predict(np.array(x[-180:]))

plt.scatter(x[-180:], y[-180:])
plt.plot(x[-180:], pred, 'red')
plt.savefig('plot_fit_mle_zscore.png')
```

✓ **시각화 결과**

Training한 후 모델이 test 데이터셋에 대해 예측한 결과는 다음과 같다. 빨간 선이 Z-score normalization 기법을 적용한 MLE기반 linear regression 모델의 예측 값이고 파란 점이 실제 값이다.



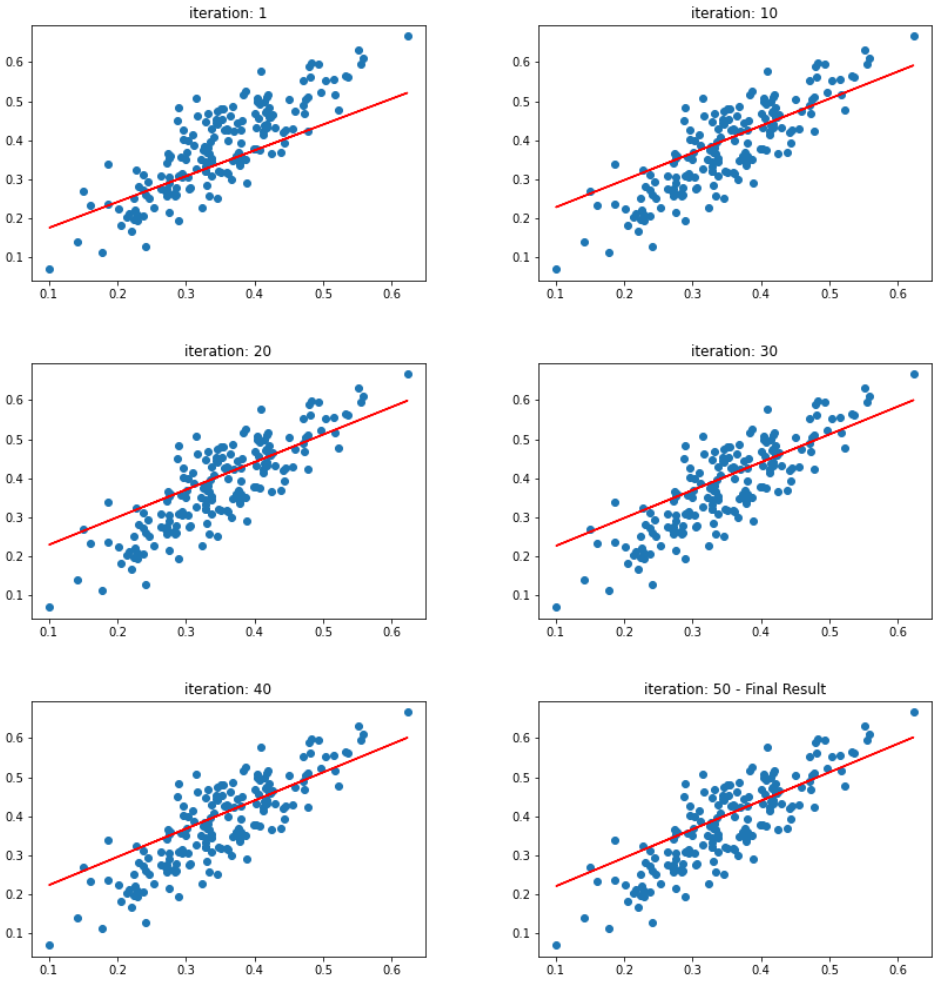
- Min-max normalization 기법을 적용한 gradient 기반 linear regression

Gradient 기반 linear regression 클래스는 P2에서 구현한 gradient 기반 linear regression 클래스와 동일하고, fit함수에 x와 y 데이터를 넣고 훈련하기 전에 x와 y데이터에 min-max normalization 기법을 적용하였다.

```
reg = LinearRegression()

x = (data['Weight']-data['Weight'].min()) / (data['Weight'].max() - data['Weight'].min())
y = (data['Height']-data['Height'].min()) / (data['Height'].max() - data['Height'].min())

params = reg.fit(x[:-180], y[:-180])
pred0 = reg.predict(np.array(x[-180:]), 0)
pred1 = reg.predict(np.array(x[-180:]), 9)
pred2 = reg.predict(np.array(x[-180:]), 19)
pred3 = reg.predict(np.array(x[-180:]), 29)
pred4 = reg.predict(np.array(x[-180:]), 39)
pred5 = reg.predict(np.array(x[-180:]), 49)
```

✓ **시각화 결과**

Training한 model이 test 데이터셋에 대해 예측한 결과를 시각화 하면 다음과 같다. 빨간 선이 min-max normalization 기법을 적용한 gradient기반 linear regression 모델의 예측 값이고 파란 점이 실제 값이다. Iteration 1, 10, 20, 30, 40, 50마다 모델 fitting 결과를 보면 다음과 같다.



- **Z-score normalization 기법을 적용한 gradient 기반 linear regression**

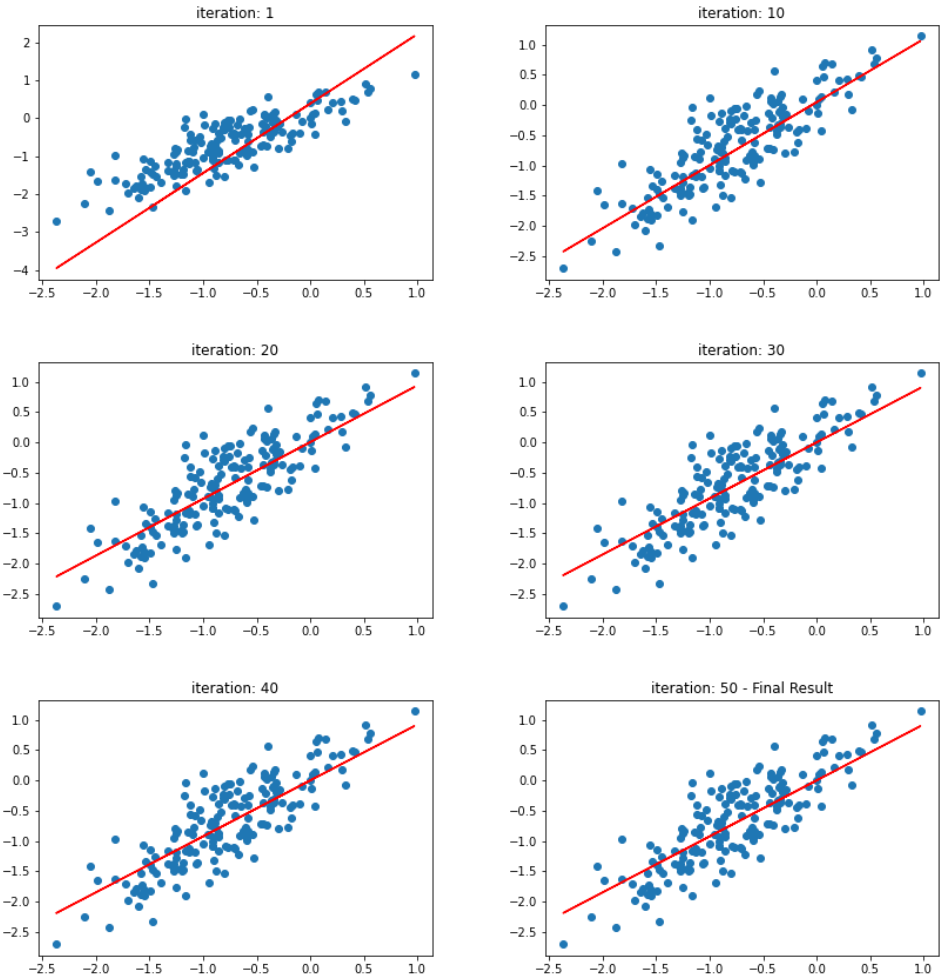
Gradient 기반 linear regression 클래스는 P2에서 구현한 gradient 기반 linear regression 클래스와 동일하고, fit함수에 x와 y 데이터를 넣고 훈련하기 전에 x와 y데이터에 Z-score normalization 기법을 적용하였다.

```
reg = LinearRegression()

x = (data['Weight']-data['Weight'].mean())/data['Weight'].std()
y = (data['Height']-data['Height'].mean())/data['Height'].std()

params = reg.fit(x[:-180], y[:-180])
pred0 = reg.predict(np.array(x[-180:]), 0)
pred1 = reg.predict(np.array(x[-180:]), 9)
pred2 = reg.predict(np.array(x[-180:]), 19)
pred3 = reg.predict(np.array(x[-180:]), 29)
pred4 = reg.predict(np.array(x[-180:]), 39)
pred5 = reg.predict(np.array(x[-180:]), 49)
```

✓ **시각화 결과**

Training한 model이 test 데이터셋에 대해 예측한 결과를 시각화 하면 다음과 같다. 빨간 선이 Z-score normalization 기법을 적용한 gradient기반 linear regression 모델의 예측 값이고 파란 점이 실제 값이다. Iteration 1, 10, 20, 30, 40, 50마다 모델 fitting 결과를 보면

다음과 같다.


iteration: 1
iteration: 10
iteration: 20
iteration: 30
iteration: 40
iteration: 50 - Final Result