Machine Learning Assignment3 (group24)

source: https://github.com/WarClans612/machine_learning/tree/master/hw3

- Working environment

張翔中	劉昱劭	彭敬樺	周才錢
macOS	macOS/Ubuntu 16.04	Ubuntu 16.04	Windows

IDE	Jupyter Notebook

- Our files (click to see readme.md)

ML Assignment3

Problem1

- single-var LR with built-in function
  - The code, graph, accuracy, weight and bias for problem 1

Problem2

- single-var LR with own gradient descent
  - The code, graph, accuracy, weight and bias for problem 2

Problem3

- **multi**-var LR with own gradient descent
  - The code, MSE, R2, and the accuracy for problem 3

Problem3_wj

- **multi**-var LR with own gradient descent
  - Using individual wj to be updated in each iteration, rather than **w** vector

Problem4

- Polynomial Regression with own gradient descent
  - The code, MSE, R2, and the accuracy for problem 4

Bonus

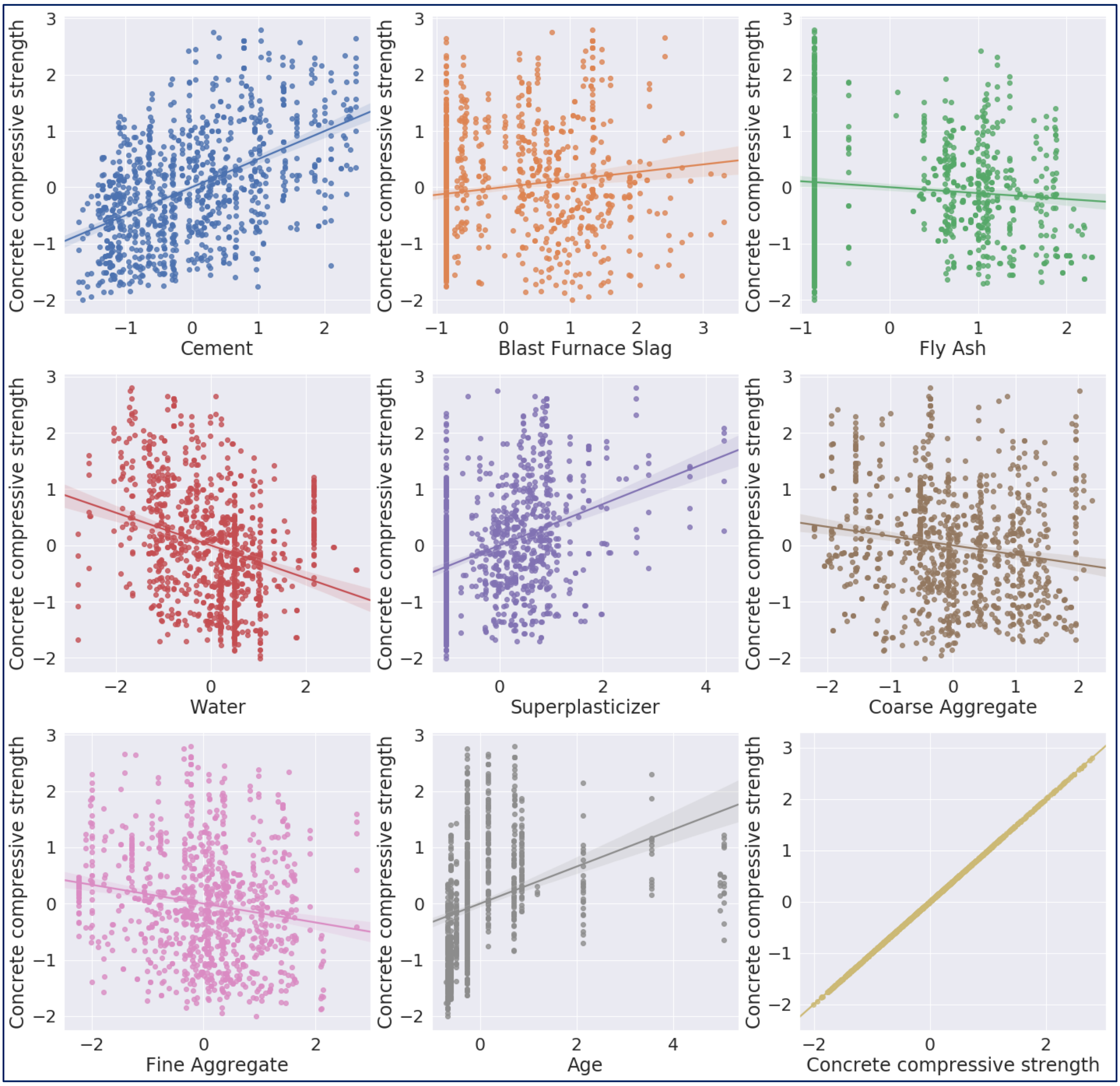
- Making different regression model to make the accuracy > 0.87 Hint: You can also change your loss function

I. Problem 1 (click to see .ipynb file)

- Visualization
  - Normalize data first.
  - Plot all attributes with the target
  - Using **scatter plots** with **regression line**

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Create a figure instance, and the two subplots
inputNum = 8

sns.set(font_scale=2)
fig, axes = plt.subplots(3, 3, figsize=(24, 24))


for i in range(0, 3):
    for j in range(0, 3):
        sns.regplot(x=df.columns[i*3+j], y=df.columns[inputNum], data=df, ax=axes[i][j])
```



- # Data Partition

Data Partition

- For each input attribute
  - 80% data for training
  - 20% data for testing

```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

row = ['lm1', 'lm2', 'lm3', 'lm4', 'lm5', 'lm6', 'lm7', 'lm8']
col = ['MSE', 'R2', 'bias', 'weight']
regResult = pd.DataFrame(index=row, columns=col)

inputNum = 8

X, y = df.iloc[:, 0:inputNum], df.iloc[:, inputNum:inputNum+1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

- # Train Model & Predict

  - To find most relative attribute to the target, we do regression for each input attribute.

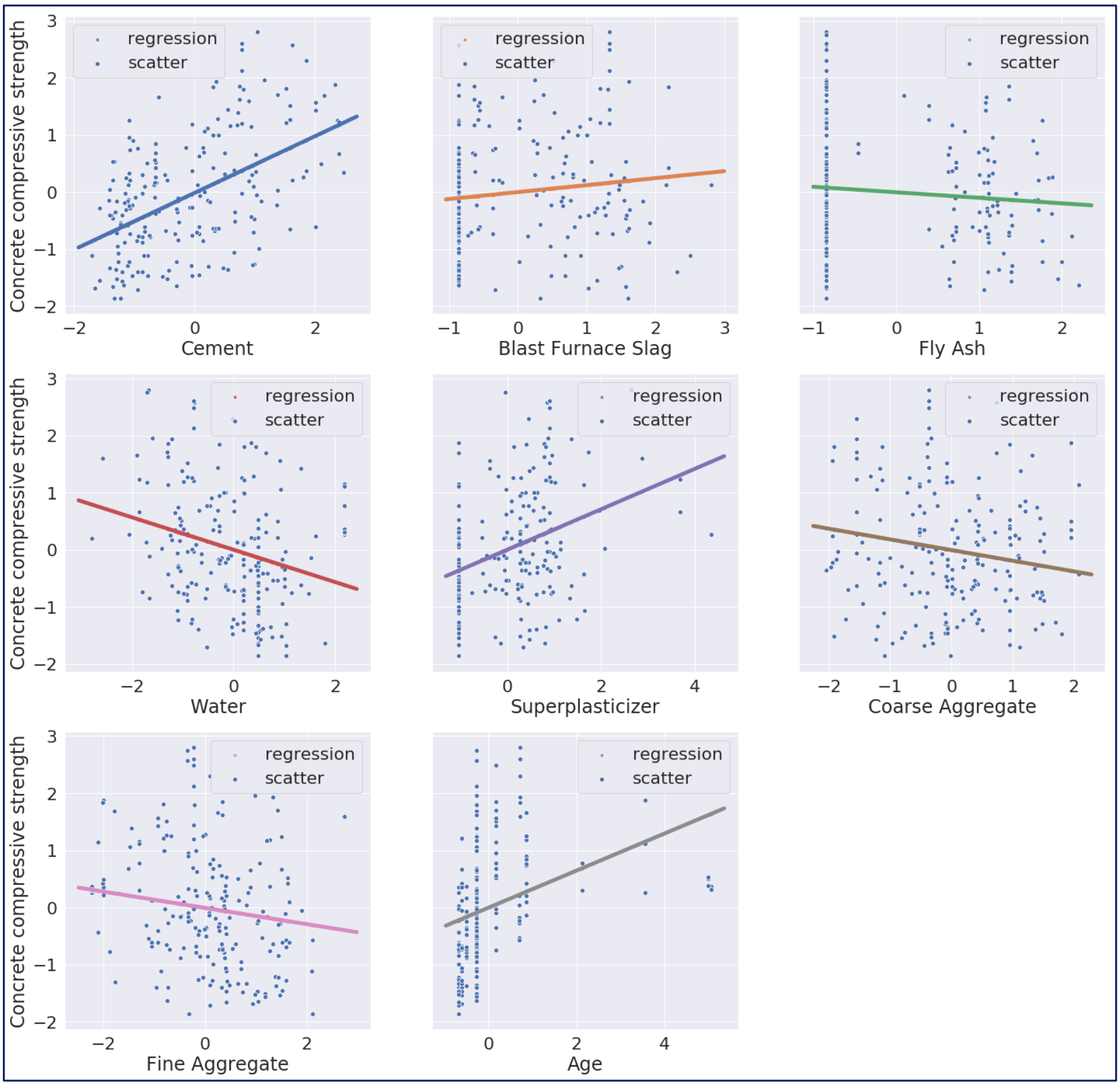
Simple Linear Regression

- iteratively train linear model with each attribute

```
from sklearn.linear_model import LinearRegression

fig, axes = plt.subplots(3, 3, figsize=(24, 24), sharey=True)
axes[2][2].set_visible(False)

# Put train/test data to DataFrame to draw plot
Train = pd.concat([X_train, y_train], axis=1)
Test = pd.concat([X_test, y_test], axis=1)

for i in range(0, inputNum):

    """ simple linear regression by sklearn function"""

    # Train linear model by training set
    reg1 = LinearRegression().fit(X_train.iloc[:, i:i+1], y_train)

    # Prediction
    y_pred_lm = reg1.predict(X_test.iloc[:, i:i+1])
    Test['y_pred_lm'] = y_pred_lm

    # Plot outputs
    plotRow = i//3
    plotCol = i%3

    sns.set(font_scale=2)
    sns.regplot(x=Test.columns[i], y='y_pred_lm',
                data=Test, label='regression',
                marker='.', line_kws={"linewidth": 5 },
                ax=axes[plotRow][plotCol])

    sns.set(font_scale=2)
    sns.scatterplot(x=Test.columns[i], y='Concrete compressive strength',
                    data=Test, label='scatter',
                    ax=axes[plotRow][plotCol])


    # Record metrics
    regResult.iloc[i, 0] = mean_squared_error(y_test, y_pred_lm)
    regResult.iloc[i, 1] = r2_score(y_test, y_pred_lm)
    regResult.iloc[i, 2] = reg1.intercept_[0]
    regResult.iloc[i, 3] = reg1.coef_[0]
```

- # Result

  - Show scatter plots of target against each attribute.
  - Draw the predicted regression line on each scatter plot.



  - Show MSE, R2-score, bias, weight for each model

```
regResult
```

	MSE	R2	bias	weight
lm1	0.874502	0.241111	-0.0143927	[0.49595956271523484]
lm2	1.12196	0.0263646	-0.00161847	[0.12219844837073131]
lm3	1.1344	0.0155765	-0.00553398	[-0.0964613023217141]
lm4	1.05903	0.0809785	0.000667177	[-0.2829982024464298]
lm5	0.982931	0.147017	0.00182119	[0.3533156004214677]
lm6	1.15892	-0.0057036	-0.00510123	[-0.1868353832929971]
lm7	1.09665	0.0483345	-0.00697862	[-0.1421942502700733]
lm8	1.01867	0.116005	-0.00362499	[0.3248046251648580 3]

Find the largest R2-score.

The first attribute “**Cement**” is most relative to target.

II. Problem 2

- Data Partition

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

inputNum = 8

X, y = df.iloc[:, 0:inputNum], df.iloc[:, inputNum:inputNum+1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

- Gradient Descent Function

```python
wName = ['w0', 'w1', 'w2', 'w3', 'w4', 'w5', 'w6', 'w7', 'w8']
trainMSE = []
trainR2 = []

def descent3(X, y, dimension, w_current, learning_rate):
    w_gradient = np.zeros(dimension)
    N = float(X.shape[0])
    for i in range(0, X.shape[0]):
        x_data = X.iloc[i]
        y_data = y.iloc[i]
        #Calculating predicted value of model
        predicted = w_current[0]
        for j in range(1, X.shape[1]+1):
            predicted += w_current[j]*x_data[j-1]
        #Error value by model
        error = y_data[0] - predicted

        #Accumulating gradient from each
        w_gradient[0] += -(2/N) * error
        for j in range(1, X.shape[1]+1):
            w_gradient[j] += -(2/N) * error * x_data[j-1]
    step_size = w_gradient * learning_rate
    new_w = w_current - step_size
    return new_w, step_size


def gd3(X, y, dimension, learning_rate=0.01, epochs=3000, stopThreshold = 0.000001):
    w_cur = np.random.uniform(-0.5,0.5,dimension)

    for i in range(epochs):
        w_cur, stepsize = descent3(X, y, dimension, w_cur, learning_rate)
        #y_pred_temp = y.copy.drop(index)

        Xcopy = X.copy()
        for j in range(dimension-1):
            Xcopy.iloc[:, j] *= w_cur[j+1]
        y_pred_temp = Xcopy.sum(axis=1) + w_cur[0]

        mse = mean_squared_error(y, y_pred_temp)
        r2 = r2_score(y, y_pred_temp)

        trainMSE.append(mse)
        trainR2.append(r2)
        #print('w_cur:\n', w_cur, '\n----------\n')
        #print('step:\n', step, '\n----------\n')
        if all(abs(stepsize) < stopThreshold):
            print('prev_grad:\n', stepsize, '\n----------\n')
            print("epoch: ", i)
            break
    return w_cur
```

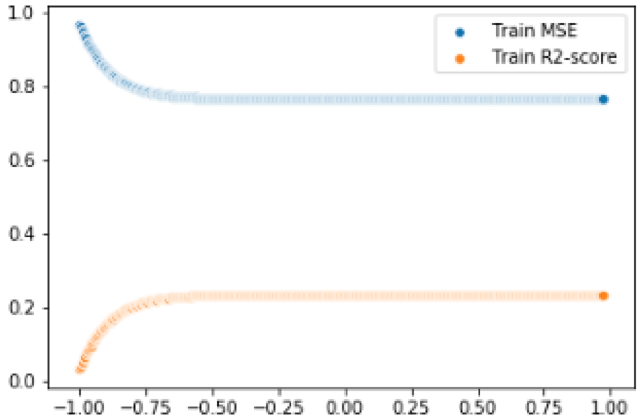
- Training

```
w_finish = gd3(X_train.iloc[:, 0:1], y_train, 2)
```

- MSE and R2-Score for Training Data

```
T = np.linspace(-1, max(trainMSE), len(trainMSE))
sns.scatterplot(x=T, y=trainMSE, label='Train MSE')
sns.scatterplot(x=T, y=trainR2, label='Train R2-score')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f362c0f2780>
```




```
print("Init MSE: %f \nFinal MSE: %f" % (trainMSE[0], trainMSE[-1]))
print()
print("Init R2: %f \nFinal R2: %f" % (trainR2[0], trainR2[-1]))
```

```
Init MSE: 0.969309
Final MSE: 0.765628

Init R2: 0.031006
Final R2: 0.234620
```

- Prediction

Prediction ¶

```
X_test2=X_test.iloc[:, 0:1].copy()
```

```
# fit test data to our gd model

# w1X1, w2X2, ... , w8X8
for i in range(1):
    X_test2.iloc[:, i] *= w_finish.iloc[0, i+1]

# (w1X1 + w2X2 + ... + w8X8) + w0
y_pred_gd2 = X_test2.sum(axis=1) + w_finish.iloc[0, 0]
```

```
#row = ['lm1', 'lm2', 'lm3', 'lm4', 'lm5', 'lm6', 'lm7', 'lm8']
col = ['MSE', 'R2', 'bias', 'weight']
regResult = pd.DataFrame(columns=col)
regResult.loc['gd2'] = mean_squared_error(y_test, y_pred_gd2), r2_score(y_test, y_pred_gd2),
                       w_finish.iloc[0, 0],  np.array(w_finish.iloc[0, 1:2])
```

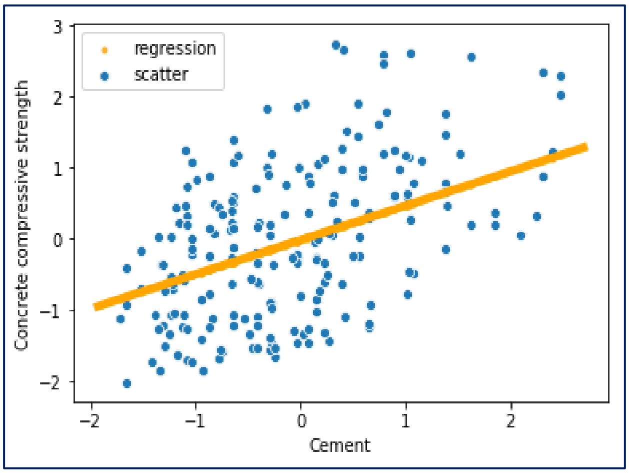
- Weight and Bias

	w0	w1
0	-0.012413	0.482582

- MSE and R2-Score for Testing Data

	MSE	R2	bias	weight
gd2	0.899731	0.240419	-0.012413	[0.482581993569137]

- Graph



- Explanation of initial weight and bias

```
w_cur = np.random.uniform(-0.5,0.5,dimension)
```

  - Initial weight and bias are chosen on random value within [-0.5, 0.5]. This range is chosen so that the initial value is not too near to 1, which made the multiplied value change only a little for each iteration. Negative and positive range is covered so that we hope that the value can be balanced.

- Problem 1 and 2 Comparison

our GD

sklearn SGD

	MSE	R2	bias	weight
gd2	0.899731	0.240419	-0.0124127	[0.482581993569137]
sgd	0.899983	0.240206	[-0.008156156752071948]	[0.4777353225361553]

  - compare our GD function with **SGDRegressor** in **sklearn**
  - MSE, R2-score, bias & weight are all close to build-in function.
  - Normalization of the data has made significant improvement in training time and training result.
  - Result for Problem 1 and 2 are both really low, this is due to the choice of only one parameter involved in the regression process which made even the highest relevant parameter pale in comparison.

III. Problem 3

- Gradient Descent Function

(the same as Problem 2)

- Training

```
w_finish = gd3(X_train.iloc[:, :], y_train, 9)
```

- Weight and Bias Result

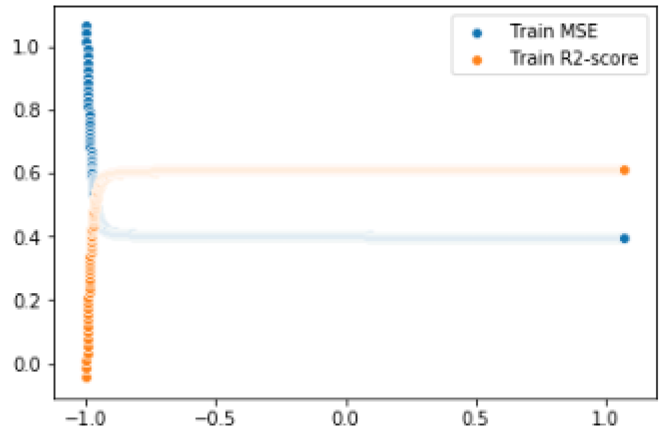
```
w_finish = pd.DataFrame(w_finish.reshape((1,9)), columns=wName)
w_finish
```

	w0	w1	w2	w3	w4	w5	w6	w7	w8
0	-0.008716	0.691551	0.47938	0.286302	-0.252036	0.082099	0.044139	0.049922	0.433864

- MSE and R2-Score for Training Data

```
T = np.linspace(-1, max(trainMSE), len(trainMSE))
sns.scatterplot(x=T, y=trainMSE, label='Train MSE')
sns.scatterplot(x=T, y=trainR2, label='Train R2-score')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9afc19dd30>
```




```
print("Init MSE: %f \nFinal MSE: %f" % (trainMSE[0], trainMSE[-1]))
print()
print("Init R2: %f \nFinal R2: %f" % (trainR2[0], trainR2[-1]))
```

```
Init MSE: 1.068170
Final MSE: 0.397721

Init R2: -0.042614
Final R2: 0.611794
```

- MSE and R2-Score for Testing Data

Prediction

```
X_test2=X_test.iloc[:, 0:8].copy()
```

```
# w1X1, w2X2, ... , w8X8
for i in range(8):
    X_test2.iloc[:, i] *= w_finish.iloc[0, i+1]
```

```
# fit test data to our gd model

# (w1X1 + w2X2 + ... + w8X8) + w0
y_pred_gd3 = X_test2.sum(axis=1) + w_finish.iloc[0, 0]


#row = ['lm1', 'lm2', 'lm3', 'lm4', 'lm5', 'lm6', 'lm7', 'lm8']
col = ['MSE', 'Cor', 'R2', 'bias', 'weight']
regResult = pd.DataFrame(columns=col)
regResult.loc['gd3'] = mean_squared_error(y_test, y_pred_gd3), 0, r2_score(y_test, y_pred_gd3), \
                        w_finish.iloc[0, 0],  np.array(w_finish.iloc[0, 1:9])
```

```
regResult
```

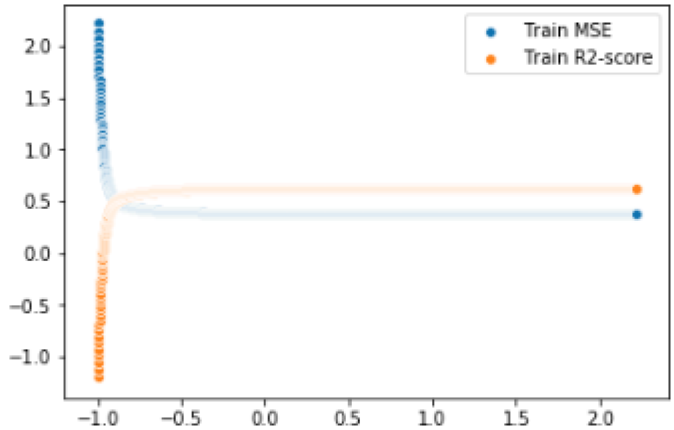
	MSE	Cor	R2	bias	weight
gd3	0.336063	0	0.627384	-0.008716	[0.6915508856761443, 0.4793802520826204, 0.286...

- Comparison each iteration only update w_j and each iteration updates **w** vector

Training

```
T = np.linspace(-1, max(trainMSE), len(trainMSE))
sns.scatterplot(x=T, y=trainMSE, label='Train MSE')
sns.scatterplot(x=T, y=trainR2, label='Train R2-score')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa832d0aef0>
```




```
print("Init MSE: %f \nFinal MSE: %f" % (trainMSE[0], trainMSE[-1]))
print()
print("Init R2: %f \nFinal R2: %f" % (trainR2[0], trainR2[-1]))
```

```
Init MSE: 2.218775
Final MSE: 0.379261

Init R2: -1.194602
Final R2: 0.624871
```

Testing

Prediction

```
X_test2=X_test[:, 0:inputNum].copy()
```

```
# w1X1, w2X2, ... , w8X8
for i in range(inputNum):
    X_test2[:, i] *= w_finish.iloc[0, i+1]
```

```
# fit test data to our gd model

# (w1X1 + w2X2 + ... + w8X8) + w0
y_pred_gd3 = X_test2.sum(axis=1) + w_finish.iloc[0, 0]


#row = ['lm1', 'lm2', 'lm3', 'lm4', 'lm5', 'lm6', 'lm7', 'lm8']
col = ['MSE', 'Cor', 'R2', 'bias', 'weight']
regResult = pd.DataFrame(columns=col)
regResult.loc['gd3'] = mean_squared_error(y_test, y_pred_gd3), 0, r2_score(y_test, y_pred_gd3), \
                        w_finish.iloc[0, 0],  np.array(w_finish.iloc[0, 1:inputNum+1])
```

```
regResult
```

	MSE	Cor	R2	bias	weight
gd3	0.409053	0	0.565881	-0.006248	[0.6921876134540351, 0.4928534606872832, 0.299...

IV. Problem 4

- ## Gradient Descent Function

(the same as Problem 2)

- ## Polynomial Aspect

Adding consideration for the second power and third power of the value of "Cement" and "Superplasticizer". This two column is chosen due to its high relevance to the wanted data to be predicted. We hope that this could further improve accuracy.

Cement	Cement	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Superplasticizer	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Concrete compressive strength

Repeated column names indicated increasing power value as it is more to the right.

- ## Training (input number of **12**)

```
w_finish = gd3(X_train[:, :], y_train, inputNum+1)
```

- ## Weight and Bias Result

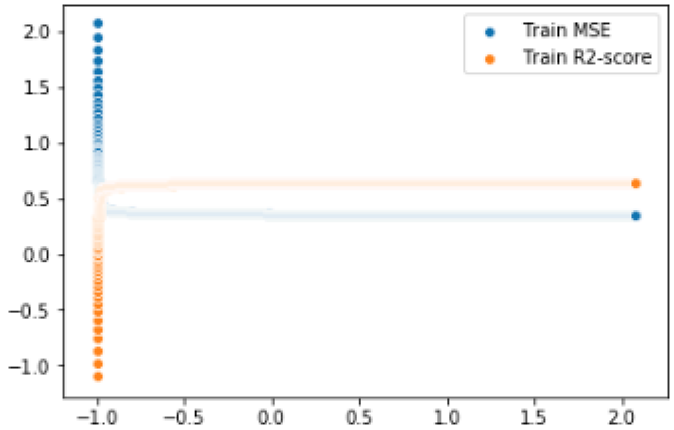
```
w_finish = pd.DataFrame(w_finish.reshape((1,inputNum+1)), columns=wName[:inputNum+1])
w_finish
```

	w0	w1	w2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12
0	0.010151	0.838794	-0.047345	-0.122054	0.450951	0.152827	-0.232281	0.840311	-1.235047	0.588117	0.031404	0.031946	0.420258

- ## MSE and R2-Score for Training Data

```python
T = np.linspace(-1, max(trainMSE), len(trainMSE))
sns.scatterplot(x=T, y=trainMSE, label='Train MSE')
sns.scatterplot(x=T, y=trainR2, label='Train R2-score')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4266303710>
```




```python
print("Init MSE: %f \nFinal MSE: %f" % (trainMSE[0], trainMSE[-1]))
print()
print("Init R2: %f \nFinal R2: %f" % (trainR2[0], trainR2[-1]))
```

```
Init MSE: 2.076882
Final MSE: 0.351959

Init R2: -1.097180
Final R2: 0.644601
```

- MSE and R2-Score for Testing Data

```python
X_test2=X_test[:, 0:inputNum].copy()
```

```python
# w1X1, w2X2, ... , w8X8
for i in range(inputNum):
    X_test2[:, i] *= w_finish.iloc[0, i+1]
```

```python
# fit test data to our gd model

# (w1X1 + w2X2 + ... + w8X8) + w0
y_pred_gd3 = X_test2.sum(axis=1) + w_finish.iloc[0, 0]


#row = ['lm1', 'lm2', 'lm3', 'lm4', 'lm5', 'lm6', 'lm7', 'lm8']
col = ['MSE', 'Cor', 'R2', 'bias', 'weight']
regResult = pd.DataFrame(columns=col)
regResult.loc['gd3'] = mean_squared_error(y_test, y_pred_gd3), 0, r2_score(y_test, y_pred_gd3), \
                       w_finish.iloc[0, 0],  np.array(w_finish.iloc[0, 1:inputNum+1])
```

```python
regResult
```

	MSE	Cor	R2	bias	weight
gd3	0.364037	0	0.648472	0.010151	[0.8387943930995451, -0.04734501288695203, -0....

V. Question

1. What is overfitting?

- Use too many parameters to train model, which causes the model fits to training data.
- If a model is overfitting, the loss function (e.g. MSE) will get very low, accuracy will get very high against training data.
- An overfitting model may predict terrible for testing data, especially the distribution of testing data varies from distribution of training data.

2. Stochastic gradient descent is also a kind of gradient descent, what is the benefit of using SGD?

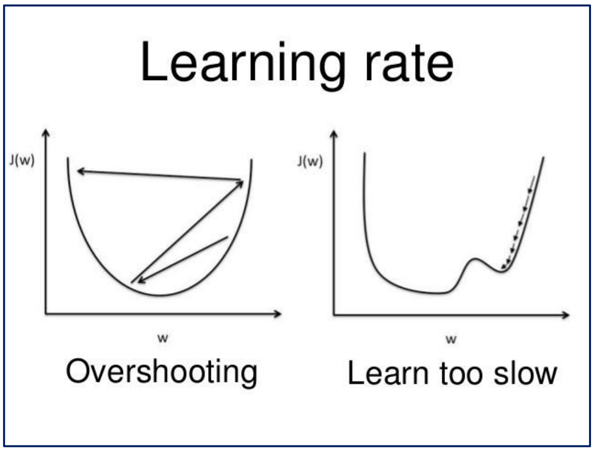
- SGD is faster due to using less data to update gradient.

3. Why the different initial value to GD model may cause different result?

- GD will converge at local minimum of parameters, there may be several local min of each parameter, so different initial value may cause different result.

4. What is the bad learning rate? What problem will happen if we use it?

- The learning rate is bad, if it is too large or too small.
- If too large, it may cause the parameter diverge. → Wrong model
- If too small, it may cause the parameter slow to converge. → Time consuming



Ref.

5. After finishing this homework, what have you learned, what problems you encountered, and how the problems were solved?

- pandas.DataFrame is slow, use data structure in numpy instead.
- Training time decreased from **about 1 hour** to **less than 5 minutes**.

VI. Bonus

- Use **Polynomial Features** & **Ridge regression** in sklearn
  - Show ridge regression with power from 1 to 5
  - The accuracy is related to the dispersion of data distribution.

```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

for count, degree in enumerate([1, 2, 3, 4, 5]):
    model = make_pipeline(PolynomialFeatures(degree), Ridge())
    model.fit(X_train, y_train)
    y_polyRidge = model.predict(X_test.iloc[:, 0:8])
    regResult.iloc[count] = 1-mean_squared_error(y_test, y_polyRidge), r2_score(y_test, y_polyRidge)
```

- Result
  - Show (1-MSE), R2-score for each model
  - The accuracy is higher when power is 3 or 4

```
regResult
```

	1-MSE	R2
ridge1	0.628474	0.640151
ridge2	0.777878	0.784859
ridge3	0.896956	0.900195
ridge4	0.882823	0.886506
ridge5	0.834225	0.839435