Part 3

In [1]:

```
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


import datetime
import random

from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error


seed = 309
random.seed(seed)
np.random.seed(seed)
```

In [2]:

```
"""
A function used to compute for the loss

"""

import numpy as np


def compute_loss(y, x, theta, metric_type):
    """
    Compute the loss of given data with respect to the ground truth
      y            ground truth
      x            input data (feature matrix)
      theta        model parameters (w and b)
      metric_type  metric type seletor, e.g., "MSE" indicates the Mean Squared E
rror.
    """
    if metric_type.upper() == "MSE":
        return np.mean(np.power(x.dot(theta) - y, 2))
    elif metric_type.upper() == "RMSE":
        return np.sqrt(np.mean(np.power(x.dot(theta) - y, 2)))
    elif metric_type.upper() == "R2":
        return - (1 - np.mean(np.power(x.dot(theta) - y, 2)) / np.mean(np.power(
y - np.mean(y), 2)))
    elif metric_type.upper() == "MAE":
        return np.mean(np.abs(y - x.dot(theta)))
```

In [3]:

```
from utilities.losses import compute_loss


def gradient_descent(y, x, theta, max_iters, alpha, metric_type):
    """
    Batch Gradient Descent
    :param y:                ground truth
    :param x:                input data (feature matrix)
    :param theta:            model parameters (w and b)
    :param max_iters:        max iterations
    :param alpha:            step size
    :param metric_type:      metric type
    :return: thetas          all tracked updated model parameters
             losses          all tracked losses during the learning course
    """
    losses = []
    thetas = []
    num_of_samples = len(x)
    for i in range(max_iters):
        # This is for MSE loss only
        gradient = -2 * x.T.dot(y - x.dot(theta)) / num_of_samples
        theta = theta - alpha * gradient
        loss = compute_loss(y, x, theta, metric_type)

        # Track losses and thetas
        thetas.append(theta)
        losses.append(loss)

        print("BGD({bi}/{ti}): loss={l}, w={w}, b={b}".format(
            bi = i, ti = max_iters - 1, l = loss, w = theta[0], b = theta[1]))
    return thetas, losses


def mini_batch_gradient_descent(y, x, theta, max_iters, alpha, metric_type, mini
_batch_size):
    """
    Mini Batch Gradient Descent
    :param y:                ground truth
    :param x:                input data (feature matrix)
    :param theta:            model parameters (w and b)
    :param max_iters:        max iterations
    :param alpha:            step size
    :param metric_type:      metric type
    :param mini_batch_size: mini batch size
    :return: thetas          all tracked updated model parameters
             losses          all tracked losses during the learning course
    """
    losses = []
    thetas = []
    # Please refer to the function "gradient_descent" to implement the mini-batc
h gradient descent here
    num_of_samples = len(x)
    for i in range(max_iters):
        # This is for MSE loss only
        gradient = -2 * x.T.dot(y - x.dot(theta)) / num_of_samples
        theta = theta - alpha * gradient
        loss = compute_loss(y, x, theta, metric_type)

        # Track losses and thetas
```

```
        thetas.append(theta)
        losses.append(loss)

        print("BGD({bi}/{ti}): loss={l}, w={w}, b={b}".format(
            bi = i, ti = max_iters - 1, l = loss, w = theta[0], b = theta[1]))
    return thetas, losses


def pso(y, x, theta, max_iters, pop_size, metric_type):
    """
    Particle Swarm Optimization
    :param y:                train labels
    :param x:                train data
    :param theta:            model parameters
    :param max_iters:        max iterations
    :param pop_size:         population size
    :param metric_type:      metric type (MSE, RMSE, R2, MAE)
    :return: best_thetas     all tracked best model parameters for each generatio
n
             losses          all tracked losses of the best model in each generat
ion
    """
    # Init settings
    w = 0.729844  # Inertia weight to prevent velocities becoming too large
    c_p = 1.496180  # Scaling co-efficient on the social component
    c_g = 1.496180  # Scaling co-efficient on the cognitive component

    terminate = False
    g_best = theta

    lower_bound = -100
    upper_bound = 100

    velocity = []
    thetas = []
    p_best = []

    # Track results
    best_thetas = []
    losses = []

    # initialization
    for i in range(pop_size):
        theta = np.random.uniform(lower_bound, upper_bound, len(theta))
        thetas.append(theta)
        p_best.append(theta)
        if compute_loss(y, x, theta, metric_type) < compute_loss(y, x, g_best, m
etric_type):
            g_best = theta.copy()
        velocity.append(
            np.random.uniform(-np.abs(upper_bound - lower_bound), np.abs(upper_b
ound - lower_bound), len(theta)))

    # Evolution
    count = 0
    while not terminate:
        for i in range(pop_size):
            rand_p = np.random.uniform(0, 1, size = len(theta))
            rand_g = np.random.uniform(0, 1, size = len(theta))
            velocity[i] = w * velocity[i] + c_p * rand_p * (p_best[i] - thetas[i
]) + c_g * rand_g * (g_best - thetas[i])
```

```
            thetas[i] = thetas[i] + velocity[i]
            if compute_loss(y, x, thetas[i], metric_type) < compute_loss(y, x, p
_best[i], metric_type):
                p_best[i] = thetas[i]
                if compute_loss(y, x, p_best[i], metric_type) < compute_loss(y,
x, g_best, metric_type):
                    g_best = p_best[i]
        best_thetas.append(g_best)
        current_loss = compute_loss(y, x, g_best, metric_type)
        losses.append(current_loss)

        print("PSO({bi}/{ti}): loss={l}, w={w}, b={b}".format(
            bi = count, ti = max_iters - 1, l = current_loss, w = g_best[0], b =
g_best[1]))
        count += 1
        if count >= max_iters:
            terminate = True
    return best_thetas, losses
```

In [7]:

```
"""
This is an example to perform simple linear regression algorithm on the dataset
 (weight and height),
where x = weight and y = height.
"""
import datetime
import random

from utilities.losses import compute_loss
from utilities.optimizers import gradient_descent, pso, mini_batch_gradient_desc
ent
from sklearn.model_selection import train_test_split

# General settings
from utilities.visualization import visualize_train, visualize_test

seed = 309
# Freeze the random seed
random.seed(seed)
np.random.seed(seed)
train_test_split_test_size = 0.3

# Training settings
alpha = 0.1  # step size
max_iters = 50  # max iterations


def load_data():
    """
    Load Data from CSV
    :return: df    a panda data frame
    """
    df = pd.read_csv("/Users/keirynhart/Documents/Uni/Comp 309/Assignment 4/Part
2.csv")
    return df


def data_preprocess(data):
    """
    Data preprocess:
        1. Split the entire dataset into train and test
        2. Split outputs and inputs
        3. Standardize train and test
        4. Add intercept dummy for computation convenience
    :param data: the given dataset (format: panda DataFrame)
    :return: train_data       train data contains only inputs
             train_labels     train data contains only labels
             test_data        test data contains only inputs
             test_labels      test data contains only labels
             train_data_full       train data (full) contains both inputs and la
bels
             test_data_full       test data (full) contains both inputs and labe
ls
    """
    # Split the data into train and test
    train_data, test_data = train_test_split(data, test_size = train_test_split_
test_size)

    # Pre-process data (both train and test)
```

```
    train_data_full = train_data.copy()
    train_data = train_data.drop(["Height"], axis = 1)
    train_labels = train_data_full["Height"]

    test_data_full = test_data.copy()
    test_data = test_data.drop(["Height"], axis = 1)
    test_labels = test_data_full["Height"]

    # Standardize the inputs
    train_mean = train_data.mean()
    train_std = train_data.std()
    train_data = (train_data - train_mean) / train_std
    test_data = (test_data - train_mean) / train_std

    # Tricks: add dummy intercept to both train and test
    train_data['intercept_dummy'] = pd.Series(1.0, index = train_data.index)
    test_data['intercept_dummy'] = pd.Series(1.0, index = test_data.index)
    return train_data, train_labels, test_data, test_labels, train_data_full, te
st_data_full


def learn(y, x, theta, max_iters, alpha, optimizer_type = "BGD", metric_type =
"MSE"):
    """
    Learn to estimate the regression parameters (i.e., w and b)
    :param y:                   train labels
    :param x:                   train data
    :param theta:               model parameter
    :param max_iters:           max training iterations
    :param alpha:               step size
    :param optimizer_type:      optimizer type (default: Batch Gradient Descien
t): GD, SGD, MiniBGD or PSO
    :param metric_type:         metric type (MSE, RMSE, R2, MAE). NOTE: MAE ca
n't be optimized by GD methods.
    :return: thetas              all updated model parameters tracked during the
learning course
             losses             all losses tracked during the learning course
    """
    thetas = None
    losses = None
    if optimizer_type == "BGD":
        thetas, losses = gradient_descent(y, x, theta, max_iters, alpha, metric_
type)
    elif optimizer_type == "MiniBGD":
        thetas, losses = mini_batch_gradient_descent(y, x, theta, max_iters, alp
ha, metric_type, mini_batch_size = 10)
    elif optimizer_type == "PSO":
        thetas, losses = pso(y, x, theta, max_iters, 100, metric_type)
    else:
        raise ValueError(
            "[ERROR] The optimizer '{ot}' is not defined, please double check an
d re-run your program.".format(
                ot = optimizer_type))

    start_time = datetime.datetime.now()  # Track learning starting time
    thetas, losses = learn(train_labels.values, train_data.values, theta, max_it
ers, alpha, optimizer_type, metric_type)

    end_time = datetime.datetime.now()  # Track learning ending time
    exection_time = (end_time - start_time).total_seconds()  # Track execution t
ime
```

```
    # Step 4: Results presentation
    print("Learn: execution time={t:.3f} seconds".format(t = exection_time))

    # Build baseline model
    print("R2:", -compute_loss(test_labels.values, test_data.values, thetas[-1],
"R2"))  # R2 should be maximize
    print("MSE:", compute_loss(test_labels.values, test_data.values, thetas[-1],
"MSE"))
    print("RMSE:", compute_loss(test_labels.values, test_data.values, thetas[-1
], "RMSE"))
    print("MAE:", compute_loss(test_labels.values, test_data.values, thetas[-1],
"MAE"))

    return thetas, losses


if __name__ == '__main__':
    # Settings
    metric_type = "MSE"  # MSE, RMSE, MAE, R2
    optimizer_type = "BGD"  # PSO, BGD

    # Step 1: Load Data
    data = load_data()

    # Step 2: Preprocess the data
    train_data, train_labels, test_data, test_labels, train_data_full, test_data
_full = data_preprocess(data)

    # Step 3: Learning Start
    theta = np.array([0.0, 0.0])  # Initialize model parameter

    start_time = datetime.datetime.now()  # Track learning starting time
    #thetas, losses = learn(train_labels.values, train_data.values, theta, max_i
ters, alpha, optimizer_type, metric_type)

    end_time = datetime.datetime.now()  # Track learning ending time
    exection_time = (end_time - start_time).total_seconds()  # Track execution t
ime

    # Step 4: Results presentation
    print("Learn: execution time={t:.3f} seconds".format(t = exection_time))

    # Build baseline model
    #print("R2:", -compute_loss(test_labels.values, test_data.values, thetas[-
1], "R2"))  # R2 should be maximize
    #print("MSE:", compute_loss(test_labels.values, test_data.values, thetas[-
1], "MSE"))
    #print("RMSE:", compute_loss(test_labels.values, test_data.values, thetas[-
1], "RMSE"))
    #print("MAE:", compute_loss(test_labels.values, test_data.values, thetas[-
1], "MAE"))
```

```
Learn: execution time=0.000 seconds
```

In [8]:

```
"""
Visualization functions
"""

import matplotlib.pyplot as plt
import numpy as np

# Visualize the training course
from utilities.losses import compute_loss


def compute_z_loss(y, x, thetas):
    """
    Compute z-axis values
    :param y:           train labels
    :param x:           train data
    :param thetas:      model parameters
    :return: z_losses   value (loss) for z-axis
    """
    thetas = np.array(thetas)
    w = thetas[:, 0].reshape(thetas[:, 0].shape[0], )
    b = thetas[:, 1].reshape(thetas[:, 1].shape[0], )
    z_losses = np.zeros((len(w), len(b)))
    for ind_row, row in enumerate(w):
        for ind_col, col in enumerate(b):
            theta = np.array([row, col])
            z_losses[ind_row, ind_col] = compute_loss(y, x, theta, "MSE")
    return z_losses


def predict(x, thetas):
    """
    Predict function
    :param x:               test data
    :param thetas:          trained model parameters
    :return:                prediced labels
    """
    return x.dot(thetas)


def visualize_train(train_data_full, train_labels, train_data, thetas, losses, n
iter):
    """
    Visualize Function for Training Results
    :param train_data_full:   the train data set (full) with labels and data
    :param thetas:            model parameters
    :param losses:            all tracked losses
    :param niter:             completed training iterations
    :return: fig1              the figure for line fitting on training data
             fig2              learning curve in terms of error
             fig3              gradient variation visualization
    """
    fig1, ax1 = plt.subplots()
    ax1.scatter(train_data_full["Weight"], train_data_full["Height"], color = 'b
lue')

    # De-standarize
    train_mean = train_data_full["Weight"].mean()
    train_std = train_data_full["Weight"].std()
```

```
    train_data_for_plot = train_mean + train_data["Weight"] * train_std

    ax1.plot(train_data_for_plot, predict(train_data, thetas[niter - 1]), color
= 'red', linewidth = 2)
    ax1.set_xlabel("Height")
    ax1.set_ylabel("Weight")

    fig2, ax2 = plt.subplots()
    ax2.plot(range(len(losses)), losses, color = 'blue', linewidth = 2)
    ax2.set_xlabel("Iteration")
    ax2.set_ylabel("MSE")

    fig3, ax3 = plt.subplots()
    np_gradient_ws = np.array(thetas)

    w = np.linspace(min(np_gradient_ws[:, 0]), max(np_gradient_ws[:, 0]), len(np
_gradient_ws[:, 0]))
    b = np.linspace(min(np_gradient_ws[:, 1]), max(np_gradient_ws[:, 1]), len(np
_gradient_ws[:, 1]))
    x, y = np.meshgrid(w, b)
    z = compute_z_loss(train_labels, train_data, np.stack((w,b)).T)
    cp = ax3.contourf(x, y, z, cmap = plt.cm.jet)
    fig3.colorbar(cp, ax = ax3)
    ax3.plot(3.54794951, 66.63949115837143, color = 'red', marker = '*', markers
ize = 20)
    if niter > 0:
        thetas_to_plot = np_gradient_ws[:niter]
    ax3.plot(thetas_to_plot[:, 0], thetas_to_plot[:, 1], marker = 'o', color =
'w', markersize = 10)
    ax3.set_xlabel(r'$w$')
    ax3.set_ylabel(r'$b$')
    return fig1, fig2, fig3


def visualize_test(test_data_full, test_data, thetas):
    """
    Visualize Test for Testing Results
    :param test_data_full:          the test data set (full) with labels and dat
a
    :param thetas:                  model parameters
    :return: fig
    """
    fig, ax = plt.subplots()
    ax.scatter(test_data_full["Weight"], test_data_full["Height"], color='blue')
    ax.plot(test_data_full["Weight"], predict(test_data, thetas[-1]), color='re
d', linewidth=2)
    return fig
```

In [9]:

```
theta = np.array([0.0,0.0])
```

In [10]:

```
df = load_data()
```

In [11]:

```
df = data_preprocess(df)
```

In [12]:

```
#BGD_MSE = learn(train_labels, train_data, theta, 50, 0.1, "BGD", "MSE")
start_time = datetime.datetime.now()
thetas1, losses1 = gradient_descent(train_labels, train_data, theta, 50, 0.1, "M
SE")

end_time = datetime.datetime.now()
exection_time = (end_time - start_time).total_seconds()
print("Learn: execution time={t:.3f} seconds".format(t = exection_time))

print("R2:", -compute_loss(test_labels, test_data, thetas1[-1], "R2"))  # R2 sho
uld be maximize
print("MSE:", compute_loss(test_labels, test_data, thetas1[-1], "MSE"))
print("RMSE:", compute_loss(test_labels, test_data, thetas1[-1], "RMSE"))
print("MAE:", compute_loss(test_labels, test_data, thetas1[-1], "MAE"))


#visualize_train(train_data_full, train_labels, train_data, thetas, losses, 50)
```

```
BGD(0/49): loss=2852.039134851559, w=0.7075625032674936, b=13.327898
231674286
BGD(1/49): loss=1825.9850450301685, w=1.2740168273119257, b=23.99021
6817013714
BGD(2/49): loss=1169.3077854571823, w=1.7275039747326382, b=32.52007
168528526
BGD(3/49): loss=749.0326459781148, w=2.090552828182019, b=39.3439555
7990249
BGD(4/49): loss=480.055471417244, w=2.3811993674292085, b=44.8030626
9559628
BGD(5/49): loss=307.9093841173318, w=2.613882682563672, b=49.1703483
8815131
BGD(6/49): loss=197.73544243739008, w=2.800162296565605, b=52.664176
94219533
BGD(7/49): loss=127.22383403736698, w=2.949292433260866, b=55.459239
785430555
BGD(8/49): loss=82.09622153611429, w=3.0686817598380434, b=57.695290
06001873
BGD(9/49): loss=53.21443216767159, w=3.164261443572115, b=59.4841302
7968927
BGD(10/49): loss=34.73001174923218, w=3.2407798075215055, b=60.91520
24554257
BGD(11/49): loss=22.89993447014434, w=3.3020382234604178, b=62.06006
0196014845
BGD(12/49): loss=15.328654112210042, w=3.3510799610206554, b=62.9759
46388486165
BGD(13/49): loss=10.483014879307657, w=3.3903413749188798, b=63.7086
5534246322
BGD(14/49): loss=7.3817930776896485, w=3.421772941131115, b=64.29482
250564486
BGD(15/49): loss=5.397002989806634, w=3.446936154995882, b=64.763756
23619018
BGD(16/49): loss=4.126732119818887, w=3.4670811050670465, b=65.13890
322062643
BGD(17/49): loss=3.3137554214629206, w=3.483208576524019, b=65.43902
080817543
BGD(18/49): loss=2.793448192858004, w=3.496119769387572, b=65.679114
87821463
BGD(19/49): loss=2.4604501939315475, w=3.5064561015029083, b=65.8711
9013424599
BGD(20/49): loss=2.247330594886836, w=3.5147310736706716, b=66.02485
033907108
BGD(21/49): loss=2.1109334876652746, w=3.5213557799604067, b=66.1477
7850293115
BGD(22/49): loss=2.0236389776746937, w=3.5266593305386458, b=66.2461
210340192
BGD(23/49): loss=1.9677702596741875, w=3.5309052016015676, b=66.3247
9505888965
BGD(24/49): loss=1.9320141317138604, w=3.534304324663941, b=66.38773
4278786
BGD(25/49): loss=1.9091301146818787, w=3.5370255654698757, b=66.4380
8565470309
BGD(26/49): loss=1.8944842828064745, w=3.539204113109369, b=66.47836
675543675
BGD(27/49): loss=1.885110911326492, w=3.5409481961053295, b=66.51059
163602369
BGD(28/49): loss=1.8791119285325415, w=3.5423444591209527, b=66.5363
7154049323
BGD(29/49): loss=1.8752725634915703, w=3.5434622673980316, b=66.5569
9546406888
BGD(30/49): loss=1.872815359576852, w=3.5443571527672817, b=66.57349
```

```
460292939
BGD(31/49): loss=1.8712427424773803, w=3.54507357242575, b=66.586693
9140178
BGD(32/49): loss=1.8702362633074971, w=3.5456471175351862, b=66.5972
5336288852
BGD(33/49): loss=1.8695921139301248, w=3.546106281362798, b=66.60570
09219851
BGD(34/49): loss=1.8691798565925957, w=3.5464738748042173, b=66.6124
5896926236
BGD(35/49): loss=1.8689160107839402, w=3.546768159610748, b=66.61786
540708418
BGD(36/49): loss=1.868747148753297, w=3.547003755618719, b=66.622190
55734163
BGD(37/49): loss=1.868639076596648, w=3.547192367051386, b=66.625650
67754758
BGD(38/49): loss=1.8685699101234654, w=3.5473433639754814, b=66.6284
1877371235
BGD(39/49): loss=1.868525643392892, w=3.547464247798714, b=66.630633
25064417
BGD(40/49): loss=1.8684973125649997, w=3.547561023933771, b=66.63240
483218962
BGD(41/49): loss=1.868479180758035, w=3.547638500142465, b=66.633822
09742598
BGD(42/49): loss=1.8684675763521488, w=3.5477005253815395, b=66.6349
5590961507
BGD(43/49): loss=1.868460149500707, w=3.5477501810157928, b=66.63586
295936635
BGD(44/49): loss=1.8684553962954777, w=3.5477899338978434, b=66.6365
8859916737
BGD(45/49): loss=1.8684523542311213, w=3.5478217589194165, b=66.6371
6911100818
BGD(46/49): loss=1.8684504073015893, w=3.5478472371224017, b=66.6376
3352048083
BGD(47/49): loss=1.8684491612613474, w=3.547867634243763, b=66.63800
504805896
BGD(48/49): loss=1.8684483637921663, w=3.54788396359635, b=66.638302
27012146
BGD(49/49): loss=1.8684478534096924, w=3.5478970364094784, b=66.6385
4004777146
Learn: execution time=0.198 seconds
R2: 0.8362099758134548
MSE: 2.4156981721089568
RMSE: 1.5542516437530174
MAE: 1.2801806378951264
```

In [14]:

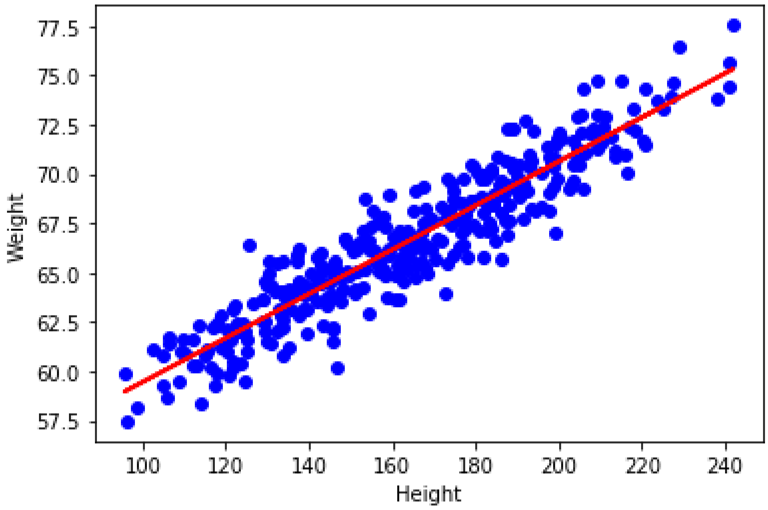
```
pred = predict(test_data, thetas1[-1])
```

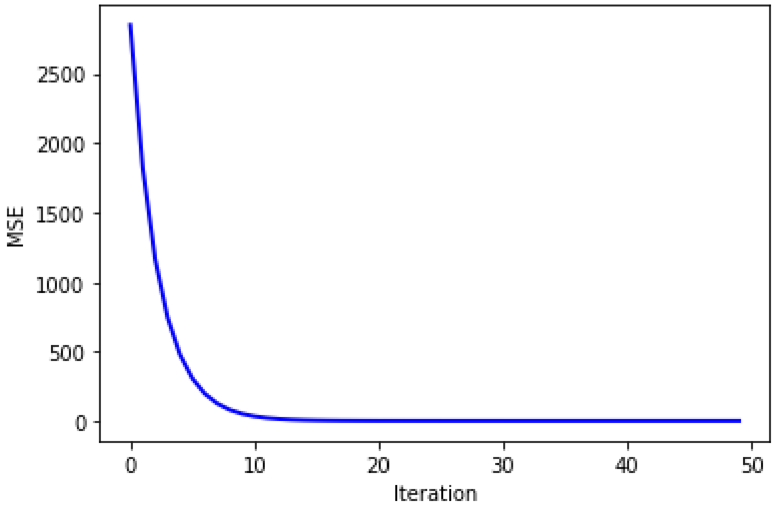
In [15]:

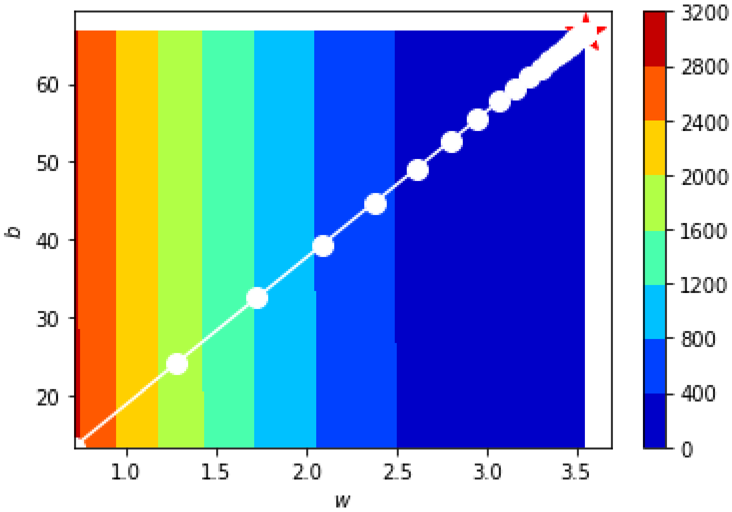
```
visualize_train(train_data_full, train_labels, train_data, thetas1, losses1, 50)
```

Out[15]:

```
(<Figure size 432x288 with 1 Axes>,
 <Figure size 432x288 with 1 Axes>,
 <Figure size 432x288 with 2 Axes>)
```







In [18]:

```
start_time = datetime.datetime.now()

thetas2, losses2 = pso(train_labels, train_data, theta, 50, 100, "MSE")

end_time = datetime.datetime.now()
exection_time = (end_time - start_time).total_seconds()
print("Learn: execution time={t:.3f} seconds".format(t = exection_time))

print("R2:", -compute_loss(test_labels, test_data, thetas2[-1], "R2"))  # R2 sho
uld be maximize
print("MSE:", compute_loss(test_labels, test_data, thetas2[-1], "MSE"))
print("RMSE:", compute_loss(test_labels, test_data, thetas2[-1], "RMSE"))
print("MAE:", compute_loss(test_labels, test_data, thetas2[-1], "MAE"))
```

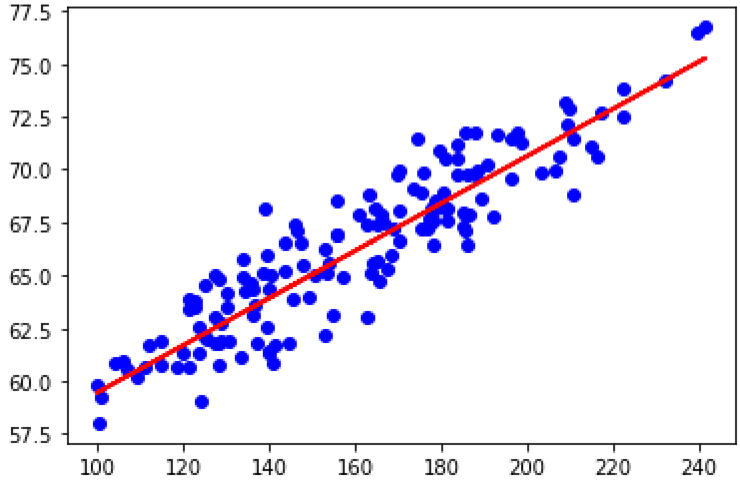
```
PSO(0/49): loss=19.369943517048956, w=2.481511642780859, b=62.593819
862400856
PSO(1/49): loss=19.369943517048956, w=2.481511642780859, b=62.593819
862400856
PSO(2/49): loss=19.369943517048956, w=2.481511642780859, b=62.593819
862400856
PSO(3/49): loss=8.946694744240066, w=6.136432670709667, b=66.0092957
0629049
PSO(4/49): loss=8.946694744240066, w=6.136432670709667, b=66.0092957
0629049
PSO(5/49): loss=4.372222219853642, w=2.676832965515933, b=65.3177122
9754142
PSO(6/49): loss=4.372222219853642, w=2.676832965515933, b=65.3177122
9754142
PSO(7/49): loss=2.0757232433537185, w=3.0954486564949235, b=66.58377
499846797
PSO(8/49): loss=2.0757232433537185, w=3.0954486564949235, b=66.58377
499846797
PSO(9/49): loss=2.0757232433537185, w=3.0954486564949235, b=66.58377
499846797
PSO(10/49): loss=2.0757232433537185, w=3.0954486564949235, b=66.5837
7499846797
PSO(11/49): loss=2.0757232433537185, w=3.0954486564949235, b=66.5837
7499846797
PSO(12/49): loss=2.0757232433537185, w=3.0954486564949235, b=66.5837
7499846797
PSO(13/49): loss=2.0757232433537185, w=3.0954486564949235, b=66.5837
7499846797
PSO(14/49): loss=1.9160046865464149, w=3.5830601320080593, b=66.4242
5057850204
PSO(15/49): loss=1.9160046865464149, w=3.5830601320080593, b=66.4242
5057850204
PSO(16/49): loss=1.9160046865464149, w=3.5830601320080593, b=66.4242
5057850204
PSO(17/49): loss=1.8919268933663267, w=3.414328695253861, b=66.56414
902685388
PSO(18/49): loss=1.8919268933663267, w=3.414328695253861, b=66.56414
902685388
PSO(19/49): loss=1.8919268933663267, w=3.414328695253861, b=66.56414
902685388
PSO(20/49): loss=1.8919268933663267, w=3.414328695253861, b=66.56414
902685388
PSO(21/49): loss=1.8705541669159615, w=3.5287875314570014, b=66.5977
6480767258
PSO(22/49): loss=1.8705541669159615, w=3.5287875314570014, b=66.5977
6480767258
PSO(23/49): loss=1.8705541669159615, w=3.5287875314570014, b=66.5977
6480767258
PSO(24/49): loss=1.8705541669159615, w=3.5287875314570014, b=66.5977
6480767258
PSO(25/49): loss=1.8705541669159615, w=3.5287875314570014, b=66.5977
6480767258
PSO(26/49): loss=1.8687123528439757, w=3.534793105813745, b=66.62985
735141953
PSO(27/49): loss=1.8687123528439757, w=3.534793105813745, b=66.62985
735141953
PSO(28/49): loss=1.8687123528439757, w=3.534793105813745, b=66.62985
735141953
PSO(29/49): loss=1.8687123528439757, w=3.534793105813745, b=66.62985
735141953
PSO(30/49): loss=1.8687123528439757, w=3.534793105813745, b=66.62985
```

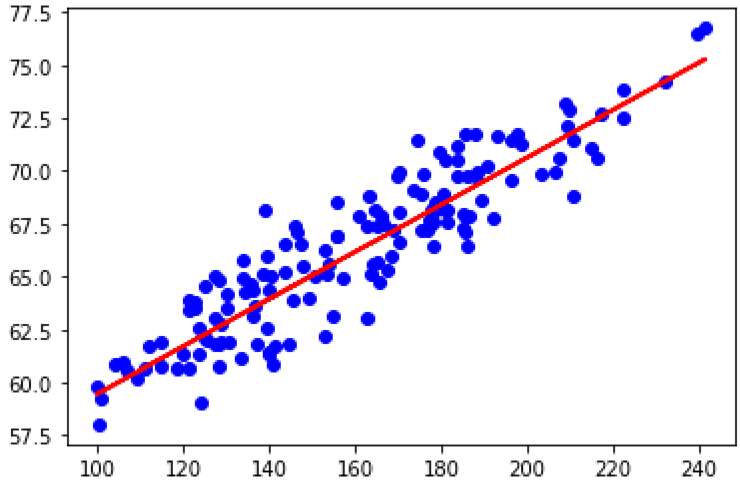
```
735141953
PSO(31/49): loss=1.8687123528439757, w=3.534793105813745, b=66.62985
735141953
PSO(32/49): loss=1.8687123528439757, w=3.534793105813745, b=66.62985
735141953
PSO(33/49): loss=1.8687123528439757, w=3.534793105813745, b=66.62985
735141953
PSO(34/49): loss=1.8687123528439757, w=3.534793105813745, b=66.62985
735141953
PSO(35/49): loss=1.8687123528439757, w=3.534793105813745, b=66.62985
735141953
PSO(36/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(37/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(38/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(39/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(40/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(41/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(42/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(43/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(44/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(45/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(46/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(47/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(48/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
PSO(49/49): loss=1.8685139188899675, w=3.5399628622564734, b=66.6413
2649594033
Learn: execution time=5.011 seconds
R2: 0.8363357683241667
MSE: 2.413842889775977
RMSE: 1.5536546880745339
MAE: 1.2799235885852764
```

In [19]:

```
pred2 = predict(test_data, thetas2[-1])
visualize_test(test_data_full, test_data, thetas2)
```

Out[19]:





In [20]:

```
start_time = datetime.datetime.now()

thetas3, losses3 = pso(train_labels, train_data, theta, 50, 100, "MAE")

end_time = datetime.datetime.now()
exection_time = (end_time - start_time).total_seconds()
print("Learn: execution time={t:.3f} seconds".format(t = exection_time))

print("R2:", -compute_loss(test_labels, test_data, thetas3[-1], "R2"))  # R2 sho
uld be maximize
print("MSE:", compute_loss(test_labels, test_data, thetas3[-1], "MSE"))
print("RMSE:", compute_loss(test_labels, test_data, thetas3[-1], "RMSE"))
print("MAE:", compute_loss(test_labels, test_data, thetas3[-1], "MAE"))
```

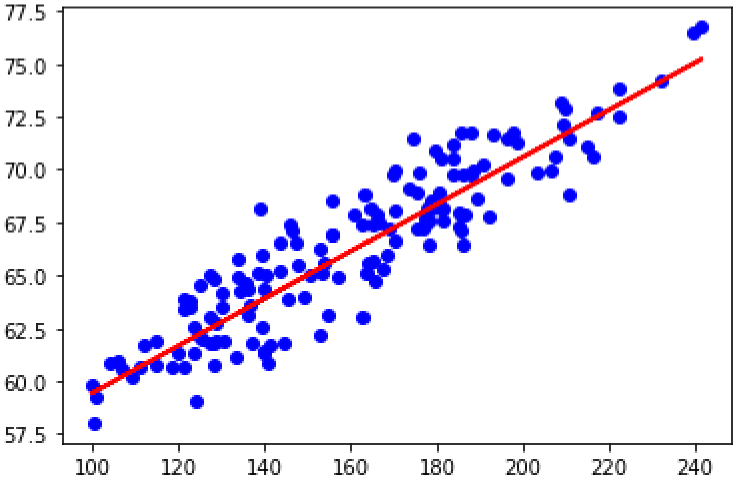
```
PSO(0/49): loss=2.968543043747424, w=6.042035561811645, b=64.3387885
8377645
PSO(1/49): loss=2.968543043747424, w=6.042035561811645, b=64.3387885
8377645
PSO(2/49): loss=2.968543043747424, w=6.042035561811645, b=64.3387885
8377645
PSO(3/49): loss=2.968543043747424, w=6.042035561811645, b=64.3387885
8377645
PSO(4/49): loss=2.968543043747424, w=6.042035561811645, b=64.3387885
8377645
PSO(5/49): loss=2.22568584231119, w=1.786601696348967, b=68.20814917
623436
PSO(6/49): loss=1.9915926111325302, w=5.527624426507129, b=66.580513
80019702
PSO(7/49): loss=1.9915926111325302, w=5.527624426507129, b=66.580513
80019702
PSO(8/49): loss=1.9915926111325302, w=5.527624426507129, b=66.580513
80019702
PSO(9/49): loss=1.9915926111325302, w=5.527624426507129, b=66.580513
80019702
PSO(10/49): loss=1.9915926111325302, w=5.527624426507129, b=66.58051
380019702
PSO(11/49): loss=1.9915926111325302, w=5.527624426507129, b=66.58051
380019702
PSO(12/49): loss=1.9915926111325302, w=5.527624426507129, b=66.58051
380019702
PSO(13/49): loss=1.9915926111325302, w=5.527624426507129, b=66.58051
380019702
PSO(14/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(15/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(16/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(17/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(18/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(19/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(20/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(21/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(22/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(23/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(24/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(25/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(26/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(27/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(28/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(29/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(30/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
```

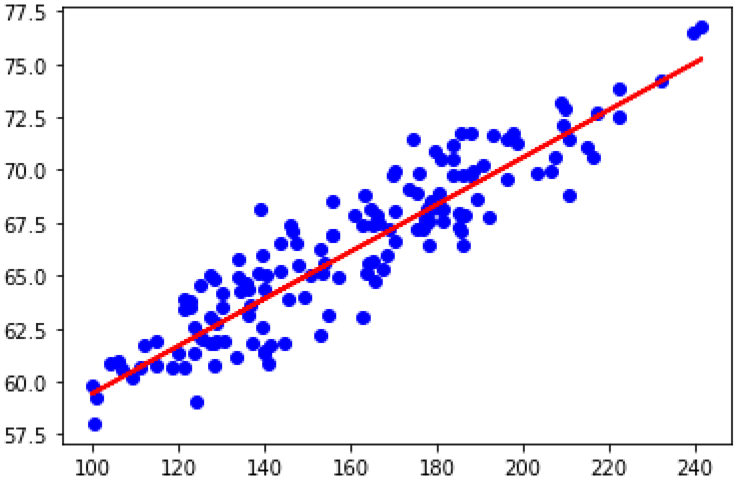
```
8216484461
PSO(31/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(32/49): loss=1.0860794665957685, w=3.4671936342826513, b=66.5595
8216484461
PSO(33/49): loss=1.0841863970862267, w=3.48521855082269, b=66.589079
68466055
PSO(34/49): loss=1.0837053233391143, w=3.5417488275094717, b=66.5984
2498299058
PSO(35/49): loss=1.0836990052122268, w=3.529313425037122, b=66.58670
611487817
PSO(36/49): loss=1.0836990052122268, w=3.529313425037122, b=66.58670
611487817
PSO(37/49): loss=1.0836990052122268, w=3.529313425037122, b=66.58670
611487817
PSO(38/49): loss=1.0836990052122268, w=3.529313425037122, b=66.58670
611487817
PSO(39/49): loss=1.0836990052122268, w=3.529313425037122, b=66.58670
611487817
PSO(40/49): loss=1.0836990052122268, w=3.529313425037122, b=66.58670
611487817
PSO(41/49): loss=1.083670367199235, w=3.533395029974717, b=66.603520
72607142
PSO(42/49): loss=1.083670367199235, w=3.533395029974717, b=66.603520
72607142
PSO(43/49): loss=1.083670367199235, w=3.533395029974717, b=66.603520
72607142
PSO(44/49): loss=1.083670367199235, w=3.533395029974717, b=66.603520
72607142
PSO(45/49): loss=1.083670367199235, w=3.533395029974717, b=66.603520
72607142
PSO(46/49): loss=1.083670367199235, w=3.533395029974717, b=66.603520
72607142
PSO(47/49): loss=1.083670367199235, w=3.533395029974717, b=66.603520
72607142
PSO(48/49): loss=1.083670367199235, w=3.533395029974717, b=66.603520
72607142
PSO(49/49): loss=1.083670367199235, w=3.533395029974717, b=66.603520
72607142
Learn: execution time=4.694 seconds
R2: 0.835518431664266
MSE: 2.4258975841026045
RMSE: 1.557529320463215
MAE: 1.283042336373881
```

In [21]:

```
visualize_test(test_data_full, test_data, thetas3)
```

Out[21]:





In [22]:

```
table = {'Name':  ['kNN', 'naive Bayes', 'SVM', 'decision tree', 'random forest'
,  'AdaBoost', 'gradient Boosting', 'linear discriminant analysis', 'multi-layer
perceptron', 'logistic regression'],
         'Accuracy': [0.94,0.86,0.98,0.87,0.97,0.92,0.95,0.89,0.99,0.90],
         'Precision': [0.93,0.81,0.98,0.82,0.97,0.89,0.94,0.86,0.99,0.87],
         'Recall': [0.91,0.84,0.96,0.80,0.94,0.88,0.92,0.83,0.99,0.96],
         'F1':[0.92,0.82,0.97,0.81,0.95,0.89,0.93,0.85,0.99,0.86]}




table = pd.DataFrame (table, columns = ['Name','Accuracy','Precision', 'Recall',
'F1'])

table
```

Out[22]:

	Name	Accuracy	Precision	Recall	F1
0	kNN	0.94	0.93	0.91	0.92
1	naive Bayes	0.86	0.81	0.84	0.82
2	SVM	0.98	0.98	0.96	0.97
3	decision tree	0.87	0.82	0.80	0.81
4	random forest	0.97	0.97	0.94	0.95
5	AdaBoost	0.92	0.89	0.88	0.89
6	gradient Boosting	0.95	0.94	0.92	0.93
7	linear discriminant analysis	0.89	0.86	0.83	0.85
8	multi-layer perceptron	0.99	0.99	0.99	0.99
9	logistic regression	0.90	0.87	0.96	0.86

In [23]:

```
table = {'Name': ['BGD', 'Mini batch', 'PSO+MSE', 'PSO+MAE'],
        'R2':[0.84, 0,0.84,0.83],
        'MSE':[2.42,0 , 2.41, 2.43],
        'RMSE':[1.55, 0, 1.55, 1.56],
        'MAE':[1.28,0 , 1.28,1.28]}
table = pd.DataFrame (table, columns = ['Name', 'R2', 'MSE', 'RMSE', 'MAE'])

table
```

Out[23]:

	Name	R2	MSE	RMSE	MAE
0	BGD	0.84	2.42	1.55	1.28
1	Mini batch	0.00	0.00	0.00	0.00
2	PSO+MSE	0.84	2.41	1.55	1.28
3	PSO+MAE	0.83	2.43	1.56	1.28

In [ ]: