Assignment 1 Part 1 – Regression

Q5. What conclusion if any can be drawn from the weight values? How does gender and BMI affect blood sugar levels? What are the estimated blood sugar levels for the below examples?

AGE	SEX	BMI	BP	S1	S2	S3	S4	S5	S6	y_pred
25	F	18	79	130	64.8	61	2	4.1897	68	43.5294
50	M	28	103	229	162.2	60	4.5	6.107	124	232.2309

Weights –
tensor ([[1.9400, -11.4488, 26.3047, 16.6306, -9.8810, -2.3179, -7.6995, 8.2121, 21.9769, 2.6065, 153.7365]])

Ans
Looking into the weights vector, weight associated with BMI is positive i.e., BMI seems to be positively co-related with blood sugar levels and as for the gender is concerned Females tend to have lower blood sugar than the males. Another noteworthy observation is that weights associated to s1-s6 are higher in comparison to the rest can be seen as major contributing factor for regression output.

Code:

```
x1 = torch.tensor([25, 2, 18, 79, 130, 64.8, 61, 2, 4.1897, 68]).reshape(1, -1)
x2 = torch.tensor([50, 1, 28, 103, 229, 162.2, 60, 4.5, 6.107, 124]).reshape(1, -1)

x1 = norm_set(x1, mu, sigma)
x2 = norm_set(x2, mu, sigma)
x1 = torch.cat([x1, torch.ones(x1.shape[0], 1)], dim=1)
x2 = torch.cat([x2, torch.ones(x2.shape[0], 1)], dim=1)

print(f"Blood sugar level Example 1: {model(x1)[0][0]}")
print(f"Blood sugar level Example 2: {model(x2)[0][0]}")
```

✓ 0.0s

```
Blood sugar level Example 1: 43.52943420410156
Blood sugar level Example 2: 232.23094177246094
```

Q6. Try the code with several learning rates that differ by orders of magnitude and record the error of the training and test sets. What do you observe on the training error? What about the error on the test set?

Ans
Ran experiments with learning rates below –
[0.0125, 0.025, 0.05, 0.125, 0.25, 0.3, 0.45, 0.5, 0.1, 0.01, 0.001, 0.0001, 0.00001, 1, 10, 100] and saw the lowest cost of 2884.9223 for training set at learning rate of 0.05. We also observe NaNs for the learning rate with values of 1, 10, 100 as they overshoot the training error and fails to converge at the optimal cost and finally, we observe a cost of 2885.6189 on the test set, which is close to the training cost indicating a good fit.

```
...  Training cost 3066.773681640625
     learning rate: 0.0125 test cost:3114.945556640625
     Training cost 2898.609619140625
     learning rate: 0.025 test cost:2890.0009765625
     Training cost 2894.800048828125
     learning rate: 0.05 test cost:2884.92236328125
     Training cost 2888.496826171875
     learning rate: 0.125 test cost:2885.8720703125
     Training cost 2907.636962890625
     learning rate: 0.25 test cost:2907.63525390625
     Training cost 10136.5478515625
     learning rate: 0.3 test cost:8.319514378800598e+30
     Training cost 15319.40234375
     learning rate: 0.45 test cost:nan
     Training cost 19150.126953125
     learning rate: 0.5 test cost:nan
     Training cost 2890.406494140625
     learning rate: 0.1 test cost:2885.618896484375
     Training cost 3356.777587890625
     learning rate: 0.01 test cost:3431.069580078125
     Training cost 20040.583984375
     learning rate: 0.001 test cost:18534.30859375
     Training cost 28468.08203125
     learning rate: 0.0001 test cost:25530.2265625
     Training cost 29583.326171875
     ...
     Training cost 29711.322265625
     learning rate: 10 test cost:nan
     Training cost 29711.322265625
     learning rate: 100 test cost:nan
```

Code:

```
### your code here
learning_rates = [0.0125, 0.025, 0.05, 0.125, 0.25, 0.3, 0.45, 0.5, 0.1, 0.01, 0.001, 0.0001, 0.00001, 1, 10, 100]

def training_function(x_test, y_test, alpha):
    cost_lst = list()
    model = LinearRegression(x_train.shape[1])
    for _ in range(100):
        prediction = model(x_train)
        cost = mean_squared_error(prediction, y_train)
        cost_lst.append(cost)
        gradient_descent_step(model, x_train, y_train, prediction, alpha)

    print(f"Training cost {min(cost_lst)}")
    test_pred= model(x_test)
    cost = mean_squared_error(test_pred, y_test)
    print(f"learning rate: {alpha} test cost:{cost}")
    return cost_lst

min_costs = []
for alpha in learning_rates:
    cost_lst = training_function(x_test, y_test, alpha)
    # print(f"lr: {alpha} min cost: {min(cost_lst)}\n")
    min_costs.append(min(cost_lst))

plt.plot(learning_rates, min_costs)
plt.xlabel("learning rates")
plt.ylabel("cost")
plt.show()
```

Q8. First, find the best value of alpha to use to optimize best. Next, experiment with different values of lambda and see how this affects the shape of the hypothesis.

Ans
We specify a range of values to search for optimal learning rate as below –
alphas = torch.linspace(1, 1.5, 100)

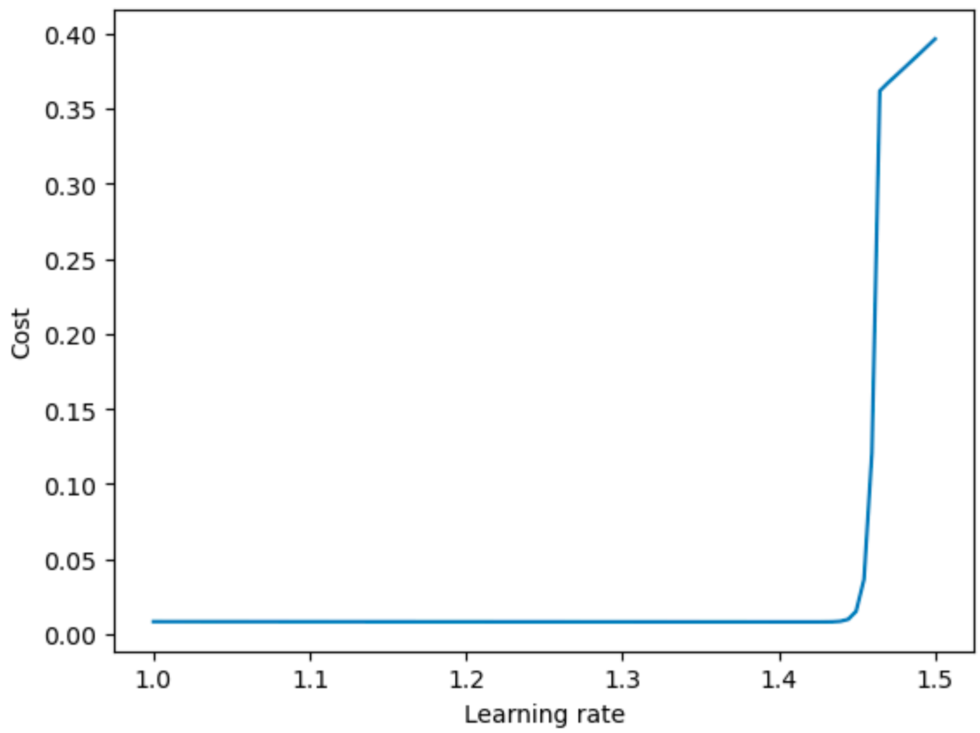
Keeping the regularization parameter constant (0.0001), we can train the model with the learning rate on the above values. The model converges at a cost of 0.0083 for learning rate 1.0.

The next step would be to arrive at a value for the regularization parameter, this is achieved by setting a constant learning rate to a constant value 1.0 and running the model for regularization parameter for the below values –
lambdas = torch.linspace(0, 1, 10000)[1:-1]

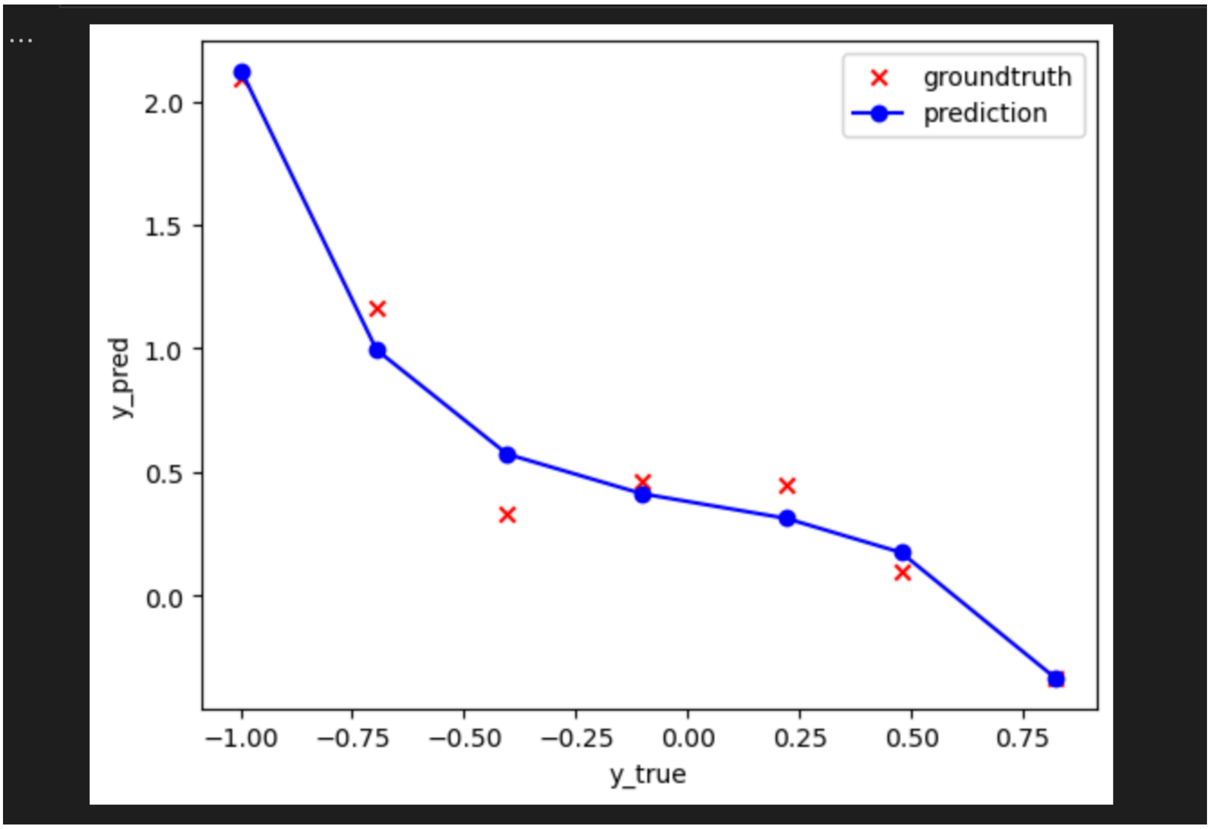
We notice that, keeping the learning rate constant 1.0, we can train the model with the regularization parameter on the above values. The model converges at a cost of 0.0083 for learning rate 0.0001.

Thus, we can conclude this model has the low cost at 0.0083 by training with the hyperparameters, learning rate set at 1.0 and the regularization param set at 0.0001.

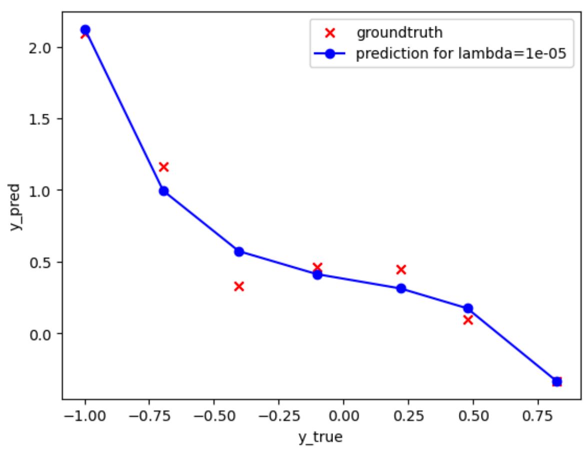
The below graph indicates that learning rates above 1.45 tends to diverge from the hypothesis function.

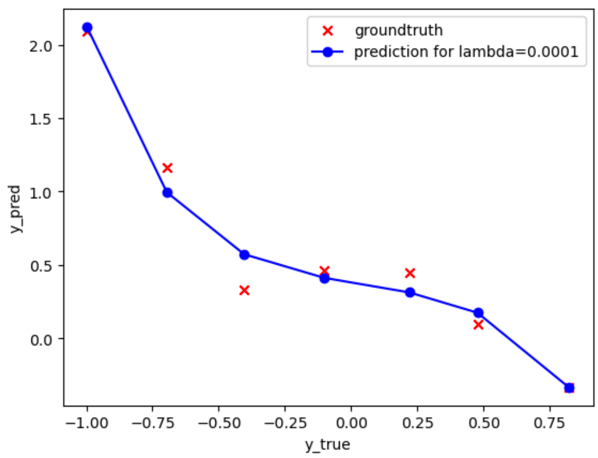


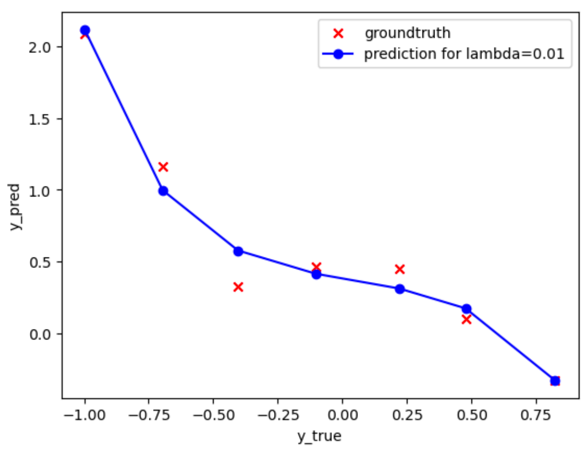
The visualization below confirms that the above model creates a good fit for the data (albeit limited data) with minimal errors with the ground truth for learning rate 1.0 and regularisation param 0.0001.

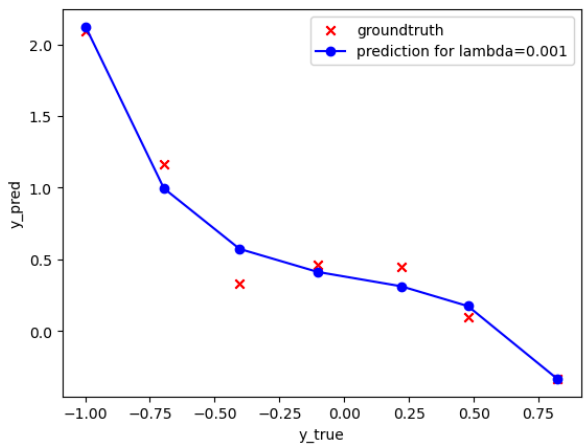


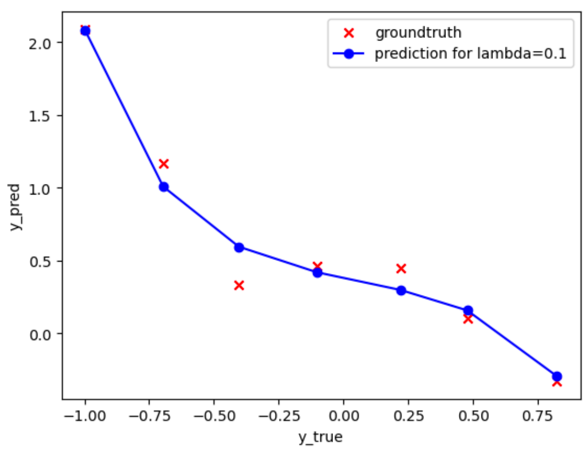
For the below values of regularisation param-
Lambdas = [0.00001, 0.0001, 0.001, 0.01, 0.1, 0, 1, 5, 10, 100], the model underfits starting from 2 as the regularization parameter now has a high value it tends to penalizes the cost function much more causing it to underfit.

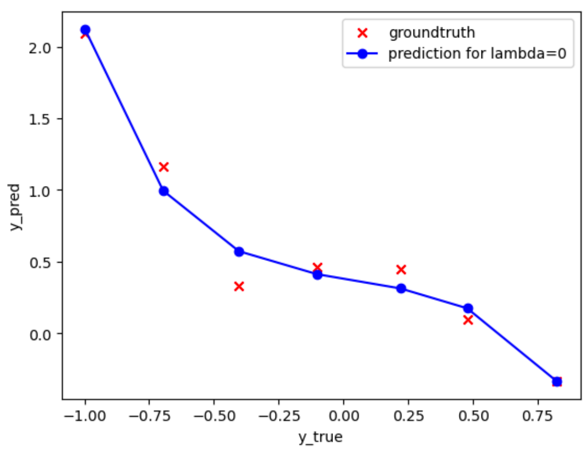


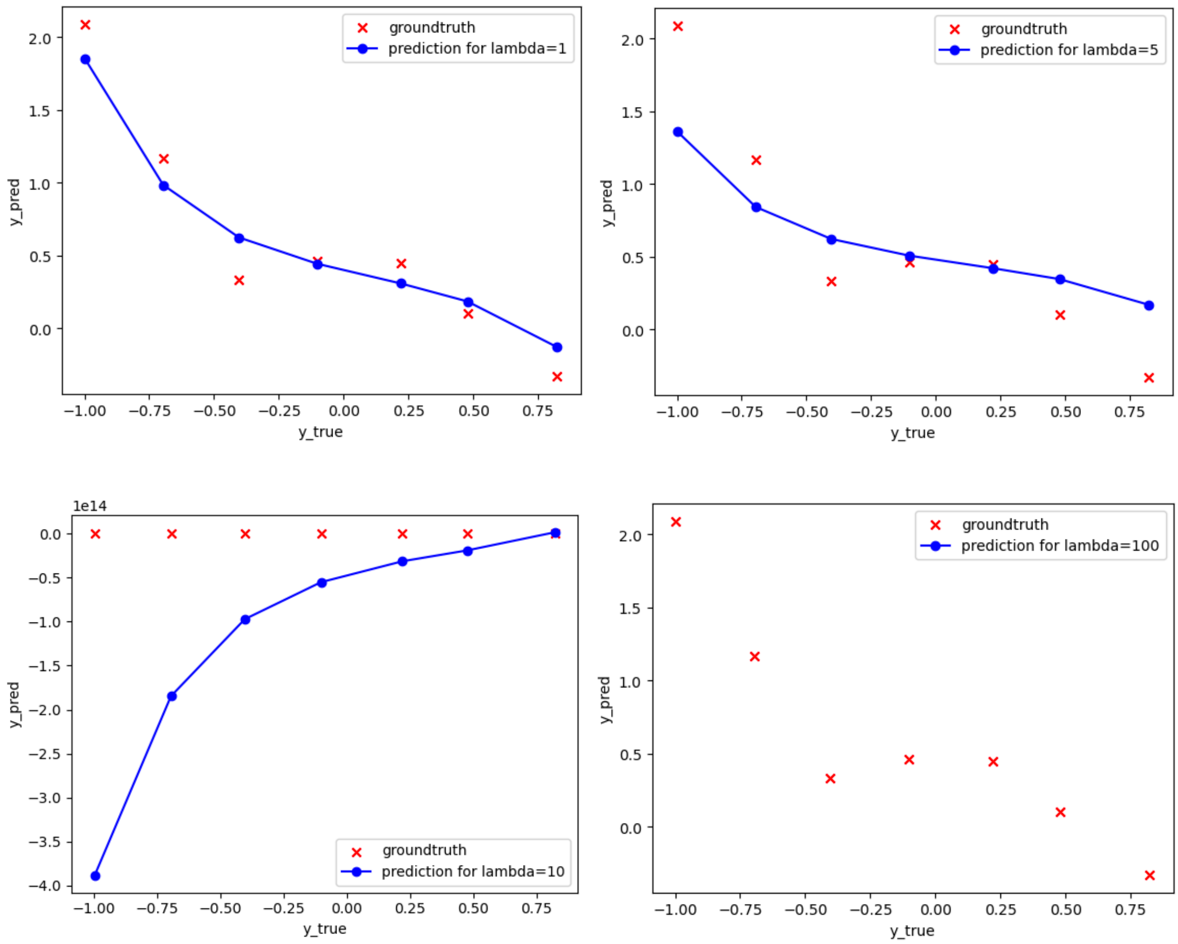












groundtruth
prediction for lambda=1
y_pred
y_true
groundtruth
prediction for lambda=5
y_pred
y_true
1e14
groundtruth
prediction for lambda=10
y_pred
y_true
groundtruth
prediction for lambda=100
y_pred
y_true