Chaojie Feng

CM146, Fall 2018
Problem Set 2: Perceptron and Regression

Oct.25, 2018

1 Problem 1

Solution: Please see CCLE for solution

2 Problem 2

Solution: Please see CCLE for solution

3 Problem 3

(a) Problem 3a

Solution: Assume there is an optimal solution δ for linear program in equation (2), δ must be 0 in order for at least one inequalities in (2) to satisfy equality conditions, and leaves the other inequality as before. As a result:

$$y_i(w^T x + \theta) \geq 1$$

Obviously, this equation can be reformatted as:

$$\begin{cases} \text{y = 1 if } w^T x + \theta \geq 1 \\ \text{y = -1 if } w^T x + \theta \leq -1 \end{cases}$$

Its constraint is more relaxed compared to the condition of separability in equation (1). So we can say the dataset is linear separable

(b) Problem 3b

Solution: If $0 \leq \delta < 1$ then D is separable. If $\delta = 1$ then the minimal separating margin will be 0. If $\delta > 1$ then D is not separable. As a result, there may be many cases for $\delta > 0$

(c) Problem 3c

Solution: The optimal solution is $\delta = 0$, which is equivalent as formulation (2). However, this formulation doesn't leave a margin. To be specific, it will give trivial solutions if a point lies on the separating hyperplane (i.e. the margin between dataset γ is 0) because $y_i(w^T x_i + \theta) \geq 0$ holds for both positive and negative cases

(d) Problem 3d

Solution: The optimal solution will be $\delta = 0$. Also with $y_i(w^T x_i + \theta) \geq 1 - \delta$:

$$1([w_1 \; w_2 \; w_3][1 \; 1 \; 1]^T + \theta) \geq 1$$

$$-1([w_1 \; w_2 \; w_3][-1 \; -1 \; -1]^T + \theta) \leq -1$$

$$w_1 + w_2 + w_3 \geq 1 + |\theta|$$

4 Problem 4

(a) Problem 4a

Solution:

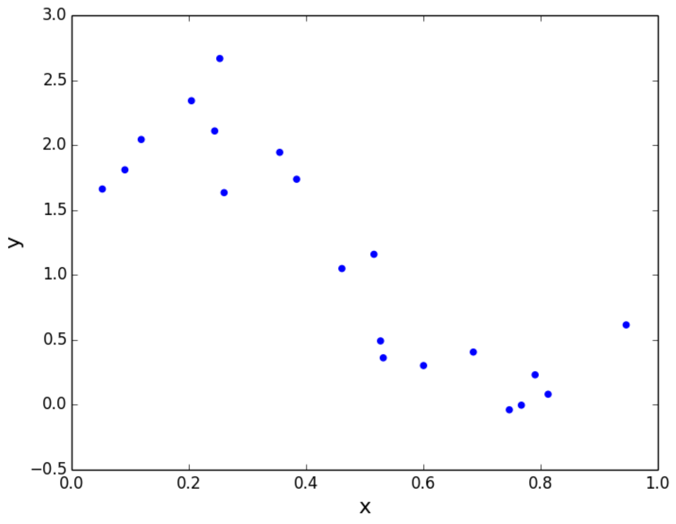
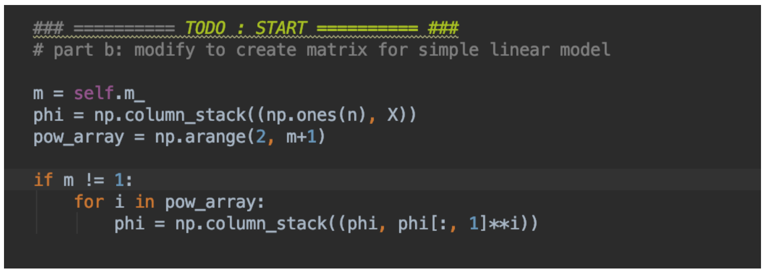


Figure 1: Visualization of training data

From the visualization of training data, a linear regression model will not behave well because the trend of data is non-linear. Underfitting will occur if a linear model is chosen

(b) Problem 4b

Solution:


```
### ========== TODO : START ========== ###
# part b: modify to create matrix for simple linear model

m = self.m_
phi = np.column_stack((np.ones(n), X))
pow_array = np.arange(2, m+1)

if m != 1:
    for i in pow_array:
        phi = np.column_stack((phi, phi[:, 1]**i))
```

Figure 2: Modified for polynomial feature

(c) Problem 4c

Solution:

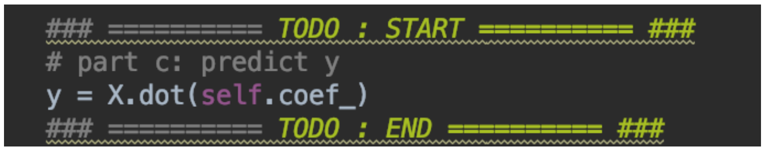

```
### ========== TODO : START ========== ###
# part c: predict y
y = X.dot(self.coef_)
### ========== TODO : END ========== ###
```

Figure 3: Prediction function

(d) Problem 4d

Solution:

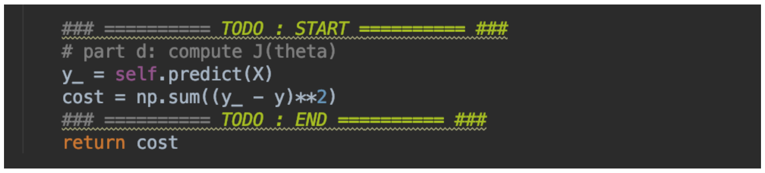
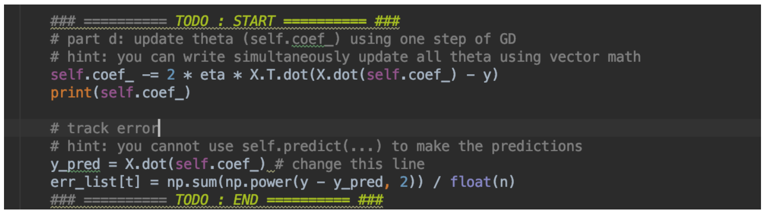

```
### ========== TODO : START ========== ###
# part d: compute J(theta)
y_ = self.predict(X)
cost = np.sum((y_ - y)**2)
### ========== TODO : END ========== ###
return cost
```

Figure 4: Calculation of cost

Figure 4, figure 5 and table 1 represent the algorithm for gradient descent as well as their performance. As we can see that if we tune our



```
### ========== TODO : START ========== ###
# part d: update theta (self.coef_) using one step of GD
# hint: you can write simultaneously update all theta using vector math
self.coef_ -= 2 * eta * X.T.dot(X.dot(self.coef_) - y)
print(self.coef_)

# track error
# hint: you cannot use self.predict(...) to make the predictions
y_pred = X.dot(self.coef_) # change this line
err_list[t] = np.sum(np.power(y - y_pred, 2)) / float(n)
### ========== TODO : END ========== ###
```

Figure 5: Gradient descent

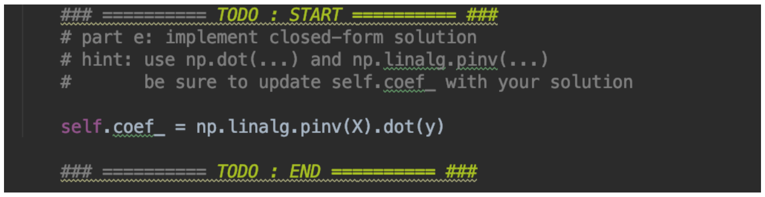
Table 1: Parameters for gradient descent

η	iterations	J*	θ
10^{-4}	10000	4.0863	$[2.2704, -2.4606]$
10^{-3}	7076	3.9126	$[2.4464, -2.8163]$
10^{-2}	765	3.9126	$[2.4464, -2.8163]$
0.047	1000	$2.711e39$	$[-9.4e18, -4.65e18]$

learning rate η to be very small, the GD algorithm will not converge to its global optimal within maximum iteration. If we tune our learning rate to be larger, we can achieve our global optimal, but number of iterations may vary greatly based on different η. If we tune our learning rate to be too large, we might not be able to converge our GD algorithm to global optimal.

(e) Problem 4e

Solution:



```
### ========== TODO : START ========== ###
# part e: implement closed-form solution
# hint: use np.dot(...) and np.linalg.pinv(...)
#       be sure to update self.coef_ with your solution

self.coef_ = np.linalg.pinv(X).dot(y)

### ========== TODO : END ========== ###
```

Figure 6: Closed form solution

The closed form solution is actually the least square solution for this model, and since our cost is defined as squared loss, the cost will be minimized when ω achieves optimal value, which will be the same as using gradient descent method. The cost using closed-form solution is 3.9126 and the coefficients θ is $[2.4464, -2.8163]$. In this case, closed form solution runs faster because the number of features are not large. As number of features increases, the computational cost for closed-form solution will be more expensive than gradient descent.

(f) Problem 4f

Solution: If we set η to be a function of k, It takes 2113 iterations for the algorithm to converge the cost to the same value as closed form solution.

(g) Problem 4g

Solution: Please refer to Problem 4b

(h) Problem 4h

Solution: From the equation, RMSE includes sample size into calculation. As a result, it is a more comparable and universal metric for the model with different data size. It also scales the error so that it matches with predictions.

(i) Problem 4i

Solution:

From the figure, we can see that if m ¡ 3, the model is underfitting because the training error is high, and when m ¿ 8 the model is overfitting because although training error is nearly zero but testing error is high. As a result, from the figure, the best m is 5

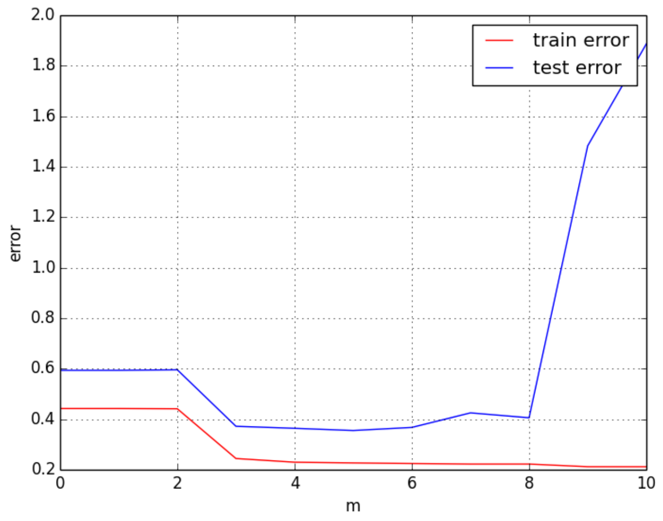


Figure 7: Comparison between training and testing error for m = 1 to 10