ML HW1 Report R05944013 網媒一 高滿馨

Linear regression function by Gradient Descent

使用batch gradient descent, 每次都會處理全部的data, 使用的cost function如下:

$$J(\beta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\beta(x^{(i)}) - y^{(i)})^2$$

```
def gradientDescent( it, yHead, iteration, alpha, theta ):

    m = len(yHead)
    J_History = zeros( shape = ( iteration, 1 ) )
    accumulate = 0

    for i in range( 0, iteration ):

        #compute prediction
        prediction = it.dot(theta)

        for x in range( len(theta) ):
            tmp = it[ :, x ]
            tmp.shape = ( m, 1 )

            #compute gradient
            derivative =  ( (( prediction - yHead )*tmp ).sum() )/m
            accumulate = accumulate + derivative*derivative
            learningRate = alpha/( delta+sqrt(accumulate) )

            #update theta
            theta[x][0] = theta[x][0] - learningRate*derivative

        #compute cost
        J_History[i][0] = computeCost( it, yHead, theta )

    return theta, J_History
```

Method

取feature:先把所有的data都讀進來, 放進同一個list裡, 然後取連續的10個數字當作一筆data做training([1, 10], [2, 11], ….)。

挑選data:看了一下data後發現有些data不太對, 比如說PM2.5最小值應該是0, 但在training set裡面有很多是-1的值, 或是SO2, NO等的檢測結果有很多也有一些負值。因此會把這些含有invalid value的那筆data刪掉。

training:使用adaptive learning rate(使用adagrad algorithm)。另外，為了避免不同資料間的數值差距過大, 所以有對全部的feature做normalization。

Discussion on regularization

使用教授上課時講的Regularization function來實作, 希望藉由第二個term來使train出的weight更接近0, 得到一個更平滑的曲線，這樣可以避免testing error的noise，增加testing data的準確度。

$$L = \sum_n \left(\hat{y}^n - \left(b + \sum w_i x_i \right) \right)^2 \boxed{+\lambda \sum (w_i)^2}$$

實際去測試lamda=0, 10, 100, 1000, 10000，用結果如下表，選擇最佳的lamda的值10，然而上傳的結果發現準確率比沒有使用regularization的準確率稍低。

lamda	training error	testing error
0	15.2826643692	19501184.2221
10	16.5873632266	19426288.6268
100	46.1528265021	23514828.7584
1000	617.887054355	30987387.7442
10000	6984.85034782	32524690.1427

推測準確率沒有比較好的原因如下:

(1)Data的數量不夠多, Variance不夠大, 所以沒有發揮regularization的效果

(2)使用的regularization function不適合這個case, 因為這個function會把每個weight都乘上相同的lamda，但其實不同weight對於最終預測值的影響力不一樣，比如說第九小時的data理論上應該會比第一個小時的影響力大，不適合同時處理，所以應該要另外找比較適合的regularization的function。

Discussion on learning rate

下表是使用不同的learning rate, 跑同樣的圈數所得到的結果, 可以發現learning rate比較大, 在跑同樣次數的情況下, training error會比較低。不過其實learning rate比較低的case都還沒完全收斂, 還有在緩慢下降, 所以若跑的圈數多一點的話, 還是可以達到比較好的準確率。但learning rate也不是越大越好, 在測試時, 發現若使用固定的比較大的learning rate, 會出現震盪的現象，代表已經跨過了一個波谷，若learning rate一直持續保持很大的話，可能無法找到真正的最低點。

learning rate	training error
0.0001	17.60113427(收斂)
0.00001	17.60233675
0.000001	17.83655476
0.0000001	19.04295499

Other Discussion

我發現test data裡面有invalid data的狀況有點麻煩，因為train data的部分可以直接丟掉，但test data無法丟掉。搜尋相關資料後找到兩種解決方法，一種是不要管這些invalid data，因為train data也會有這些invalid data，所以就讓model自己去學習。後來是決定用neighbor的值來模擬計算test data裡面的invalid data，藉由invalid data前後的兩個數字，透過內差來計算最終的值，準確度有好一些些。

另外，Adaptive learning rate的algorithm我有試過之前聽過的另外一個叫RMSprop的演算法。因為adagrad的更新方式是會一直累加gradient，所以後面的learning rate可能會受到前方極端gradient值的影響，而且到比較後面的圈數，因為累加值越來越大，所以learning rate會趨近於0。RMSProp則是增加了一個decay rate，讓先前的gradient值會隨著圈數增加，影響力會慢慢減弱。不過我實作後發現，RMSProp確實跑得比較快，但調節機制看起來沒有很好，因為很快就會開始震盪，而且error會飆得很高，最後train出來的效果也沒有比adaGrad好。