Machine Learning
HW1

r05922018
黃柏智

1. Linear regression function by Gradient Descent.

我用的 loss function 是 $C = \frac{1}{2}\sum_{x,y}(w^T x + b - y)^2 + \lambda \sum (w)^2$

加號前面那一項是總 error 值。首先將當時的 theta 與 training features 做內積，然後減掉預期的答案 y，再將這個值平方，這樣就能得到其中一筆 training data 的 error。把所有 training data 的 error 加起來除 2 即完成。

加號後面那一項是 regularization，算法是 theta 中每一個數值的平方和，再乘上 regularization rate, lamda。

而實際的 gradient_decent code 如下：

```
def gradient_descent(alpha, lamda, contTrainData, max_iter=10000):

    # initial theta
    theta = np.random.uniform(-.01, .01, (18*9+1))

    # Start Iterations
    iter = 0
    oldError = float("inf")
    error = -1
    alphaFlag = 0
    while oldError != error:
        oldError = error

        # count error
        error, loss = countError(contTrainData, theta, lamda)
        print("Iteration %d | Error: %f | alpha: %.15f" % (iter, error, alpha))

        # count gradient
        gradient = np.zeros(18*9+1)
        for i in range(len(theta)):
            gradient[i] += 2 * lamda * theta[i]
        gradient[0] = sum(loss)
        for i in range(len(contTrainData)):
            for k in range(18*9):
                gradient[k+1] += loss[i] * contTrainData[i][0][k]

        # update theta
        if error < oldError:
            if ((oldError - error) / oldError) < 0.01:
                alphaFlag += 1
        if error > oldError:
            alpha = alpha / 2
        if alphaFlag >= 10:
            alpha *= 1.1
            alphaFlag = 0
        for i in range(len(theta)):
            theta[i] -= alpha * gradient[i]

        iter += 1
        if iter >= max_iter:
            print "reach max_iter"
            break
    return theta
```

首先，在 function 一開始，會先隨機 initial 一個 theta 值，接著進入迴圈。每一個迴圈中，會先計算 error，然後算出當下的 gradent 是多少，接著再根據 learning rate 來調整 theta。其中 learning rate 是 dynamic 的，這點在第四點會詳細的講。

2. Describe your method. 因為我們沒限制你該怎麼做，所以請詳述方法 ex: 怎麼取training feature (X,y).

首先講一下全部可用的 features 有哪些，testing data 會給你前九個小時的所有資料，一個小時有 18 個 features，因此可拿來變化的 features 共有 18*9 = 162 個。
接著說明如何把 training data 變成真正能用來 train 的資料。training data 有每個月前 20 天的資料，而我的目的是希望能獲得很多下面這種 set：(前九個小時所有數值, 第十個小時的 PM2.5 數值)。我的做法是把每個月的 20 天資料串接起來，接著會得到一個 18 x 480 的數值矩陣 (24小時 x 20天 = 480)。有了這個矩陣，跨天的資料都串在一起了，就能很輕易的抽出上面提到的那種 set。一個月能夠抽出 471 個這種 set，因此我全部能用的 training data 總共有 471*12 = 5652 個。
再來說說我如何利用這 162 個 features。我試過不少取 training features 的方法，第一種是利用常數項 + 162 個 features 的一次方 + 162 個 features 的二次方。第二種是常數項 + 162 個 features 的一次方。第三種是常數項 + 162 個 features 的一次方 + 第九個小時的 18 個 features 的二次方，因為最後一個小時的數值可能跟第十小時的 PM2.5 更有關聯。
第一種方法，最後的結果不佳，似乎是會 overfitting，結果如下：

圈數	50000	100000	150000	250000
Kaggle 分數	10.96526	9.33548	9.07540	8.66647

可以看到結果並不理想，而且每一個迴圈的計算量十分龐大，頗為耗時。
第二種方法，結果明顯好上許多，結果如下：

圈數	50000	100000	200000	300000
Kaggle 分數	5.76981	5.68799	5.66052	5.65635

第三種方法，結果也不錯：

圈數	5000	15000	25000
Kaggle 分數	5.67377	5.66481	5.65838

(這裡我是以用一次方 train 了100000 圈的 theta 當作起始 theta)

此外，我還有嘗試先用一次方 train 100000 圈產生的 theta 重試一次方法一：

圈數	10000	50000	100000
Kaggle 分數	5.68041	5.65962	5.65145

沒想到結果非常不錯，甚至比跑了同圈數的方法二好 (當然方法二時間上快的多)。或許可以推論：二次方並不一定會 overfitting，如果你一開始 theta 是隨意假定一個起始點，因為二次方函數涉及的參數很多，牽一髮而動全身，所以怎麼跑都沒辦法跑到好結果，看起來就彷彿 overfitting。如果先用比較簡單的算法找到一個相對好的 theta，再用他去跑二次方，反而能讓 gradient descent 的 algorithm “看得更清晰”，進而得到好結果。

3. Discussion on regularization.

下方圖表是我用不同的 regularization rate (lamda) 去試上述的第二種方法，跑 100000 個 iterations 的結果。

lamda	0.11	0.21	0.31	0.51	50	200	500	1000	2000
Kaggle 分數	5.69181	5.69219	5.68799	5.69120	5.68699	5.69611	5.70222	5.70917	5.73077

可以看到 lamda 在一次方函數中影響的能力不大，也許在 feature 更多的時候，會有更加明顯的差異。不過有一點值得說一下，在我測二次方函數的時候，有時 learning rate 取太大，error 會一下變得超大，而加大 lamda 後，error 變得比較不容易爆掉，因此我可以使用更大的 learning rate，這會讓我相同時間內 train 出來的結果更好。

4. Discussion on learning rate.

Learning rate 越大，theta 改變的幅度就越大，通常在剛開始 training 時，都會希望 learning rate 大一點，這樣會更快得到好結果。一開始我會先 try & error 找出一個最大而且不會讓 error 爆掉的 learning rate。但一開始適用的 learning rate，到了後期就不會是最好的 rate，所以我還加了 dynamic 機制。做法是這樣的：如果連續 10 個 iterations 其 error 改善幅度都在 1% 以下，那 learning rate 會 * 1.1。但也不能一直無限上綱，當某一個 iteration 的 error 比上一個大，那 learning rate 就會砍半，其 code 如下：

```
# update theta
if error < oldError:
    if ((oldError - error) / oldError) < 0.01:
        alphaFlag += 1
if error > oldError:
    alpha = alpha / 2
if alphaFlag >= 10:
    alpha *= 1.1
    alphaFlag = 0
for i in range(len(theta)):
    theta[i] -= alpha * gradient[i]
```

dynamic learning rate 在 training 中的實際效益可以在此觀看：
https://www.csie.ntu.edu.tw/~r05922018/ML/hw1/dynamic-is-good

5. Other discussions.

除了 dynamic learning rate 之外，我在 train 一種方法的時候，會將最後得到的 theta 記錄下來。因為經過 training 的 theta，是好 theta，這個 theta 可以當作別的方法的起始 theta，等於是站在巨人的肩膀上，在同樣的圈數中便能得到更好的結果。
一些 training 中的過程我放在：
https://www.csie.ntu.edu.tw/~r05922018/ML/hw1/raw-training-procedure
而一些 train 出來的結果以及其 theta，我放在：
https://www.csie.ntu.edu.tw/~r05922018/ML/hw1/raw-training-result/