ML hw1 Report

b02902080 資工四 郭傳駿

I. linear regression function

- aga_grad:

```
def ada_grad(X, Y, L, e):
    global _weight, _b, _G_w, _G_b

    err = np.dot(X.T,_weight) + _b - Y.T
    sq_err = np.dot(err.T , err) + L * (_weight**2).sum()

    partial_w = 2 * (np.dot(X,err) + _weight * L )
    partial_b = 2 * (err.sum())

    _G_w += partial_w ** 2
    _G_b += partial_b ** 2

    _weight = _weight - (e * partial_w) /(_G_w ** 0.5)
    _b = _b - (e * partial_b) /(_G_b**0.5)
```

$$G_{i,t} = \sum_{n=0}^{t} g_{i,n}^2$$

$$x_{i,t+1} \leftarrow x_{i,t} - \frac{\eta}{\sqrt{G_{i,t}}} g_{i,t}$$

參數說明:
X : 9-hr data input of selected features, dim = (9*#features) to size_of _data
Y : Target of prediction, dim = 1 to size_of_data
L : lambda (regularization term)
e : eta (learning rate)
err : distance between target and actual output

- ada_delta:

```
def ada_delta(X, Y, L):
    global _rho, _eps, _weight, _b, _delta_w, _delta_b ,_G_w, _G_b
    err = np.dot(X.T,_weight) + _b - Y.T
    sq_err = np.dot(err.T, err) + L * (_weight**2).sum()

    partial_w = 2 * (np.dot(X,err) + _weight * L)
    partial_b = 2 * (err.sum())

    _G_w = _rho*_G_w + (1 - _rho)*partial_w ** 2
    _G_b = _rho*_G_b + (1 - _rho)*partial_b ** 2

    delta_w = (-1)*(((_delta_w + _eps)**0.5/(_G_w + _eps)**0.5)*partial_w)
    delta_b = (-1)*(((_delta_b + _eps)**0.5/(_G_b + _eps)**0.5)*partial_b)

    _weight = _weight + delta_w
    _b = _b + delta_b

    _delta_w = (_rho * _delta_w) + (1 - _rho) * (delta_w ** 2)
    _delta_b = (_rho * _delta_b) + (1 - _rho) * (delta_b ** 2)
```

$$\mathbf{G}_t = \rho \mathbf{G}_{t-1} + (1-\rho)\mathbf{g}_t^2$$

$$\Delta \mathbf{x}_t = -\frac{\sqrt{\mathbf{D}_{t-1} + \epsilon}}{\sqrt{\mathbf{G}_t + \epsilon}} \mathbf{g}_t$$

$$\mathbf{D}_t = \rho \mathbf{D}_{t-1} + (1-\rho)\Delta \mathbf{x}_t^2$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \Delta x_t$$

參數說明：

_rho : 衰退係數, 限制梯度平方累積的影響時間
_eps : 常數,防止分母為零

II. method description

- feature extraction:
  在開始訓練之前,我嘗試先手動抓出在18種feature中數值變化規律與目標pm2.5比較相關的feature,我比較每種feature與pm2.5的相關係數的來推測相關性,一來是想事先排除掉沒有幫助的feature(例如rainfall這一項幾乎都是NR),再來是在減少feature的數量同時也減少了weight的數量,總體的維度降低了也讓訓練速度加快

- feature normalisation:
  由於有各種feature的分佈情形差異很大,我有考慮把data做normalize,但是這使得error下降在數值上表現得不明顯,經過試驗發覺normalize過後預測出的結果數字會比較集中,performance也維持在一個區間內,使用的normalize公式: (Xi - Xmin)/(Xmax - Xmin)

- data structure:
  我把csv檔讀進來的資料重組成一個三維的矩陣(12個月*18個feature*每個月480個小時)的形式,方便之後運算時使用numpy的數學函式,實際訓練時再依照提取的feature數量與切割出的training/validation set組成(9*#feature)*(training_data_size)大小的X,同時也取出Y(1*train_data_size)

- training method:
  每個月共480小時以每九小時的區間滑動,總共可以做出5千多筆data,為了能配合先前做出來的structure,我從12個月中random出一至二個月作為validation set,其餘的用來train,在每個iteration的training完接著做一次validation test,觀察兩者error的變化來猜測參數的調整

- linear regression:
  在程式的核心linear-regression我實作了兩種梯度下降的方法ada_grad 及 ada_delta, 理論上ada_delta可以改善ada_grad的問題,不過實際使用發現他過度修正gradient累加的問題,導致error一直往上加,最後不得不放棄使用,推測是Gt=ρ*Gt–1+(1–ρ)gt^2這個式子讓Gt衰減得太快。

III. regularization

by ada_grad
#training params:
weight by random in range(0,1)
bias by random in range(0,1)
iteration 1,000,000

training set size 471*11
validation set size 471

• training error

	λ = 0	λ = 10	λ = 100	λ = 1000	λ = 10000
η = 0.1	33.7690619	33.754472	33.785021	34.083360	36.659434
η = 1.0	33.66262201	33.666713	33.703000	34.034136	36.635682

• validation error

	λ = 0	λ = 10	λ = 100	λ = 1000	λ = 10000
η = 0.1	49.02975994	48.673979	48.682386	48.756499	49.253441
η = 1.0	48.61998703	48.619862	48.621216	48.644456	48.964757

加入regularization項後, training set上的變化不明顯, 而validation set上的error有降低,與理論中避免overfitting的作用相符,不過由於這次作業是實作linear regression,不應該會出現嚴重的overfitting,使得加上regularization對error的改善程度不及調整其他參數(如eta)來得有效。

IV. learning rate

by ada_grad
#training params:
weight by random in range(0,1)
bias by random in range(0,1)
iteration 1,000,000

training set size 471*11
validation set size 471

- training error

	η = 0.1	**η = 1**	**η = 10**	**η = 50**
λ=10	33.80491063	33.66671666	33.66679819	33.66682649

- validation error

	η = 0.1	**η = 1**	**η = 10**	**η = 50**
λ=10	48.89675461	48.6205928	48.62934939	48.63123134

ada_grad會讓下降的步伐一直縮小,因此學習率如果設得太小,會很慢才收斂,而學習率很大雖然不會讓學習失敗,但也會拖慢收斂的時間

V. discussion

1.矩陣的結構無法搭配SGD,因此我嘗試過把datasize切成數等份,用類似一個batch update一次weight的方式模擬stochastic的效果,結果發現在後期err會不斷上下抖動,收斂不下去。

2.在所有上傳kaggle的submission中,表現最好的是只拿前九小時pm2.5的data,增加feature的使用都會始預測結果有很大的偏差(全部變負數或是一兩百以上),不過只使用單一feature也讓參數調整的空間變的有限也因此我陷入了某種兩難,或許是取參數的range不夠大的原因,之後的作業可以考慮更多。