[在此鍵入]
資工所 碩一 黃郁庭 R05922038

和上次作業一樣扣掉圖表就在兩頁內~

1. Logistic regression function

作業的目標是要分類 600 個信件是否為 spam，這裡可以用 $P_{w,b}(C_1|x)=f_{w,b}(x)$表示信件(data) 是 spam 的機率，$C_1$ 是 spam 的 label，C_1 在程式裡是以數值 1 表示，而 C_2 是非 spam 的 label，C_2 在程式裡是以數值 0 表示，因此若 $P_{w,b}(C_1|x)$大於 0.5，這裡就會預測該筆資料為 spam，反之則非 spam。在程式中 $P_{w,b}(C_1|x)$會以 $\sigma(wx+b)=1/(1+\exp(-(wx+b)))$表示。在做 gradient descent 時需要計算 weight 與 bias 對 loss 的微分，因此這裡將原本的 loss 轉換成 cross entropy 的加總：$-\ln L(w,b)=\sum_n -[\ \hat{y}^n \ln f_{w,b}(x^n)+(1-\hat{y}^n)\ln(1-f_{w,b}(x^n))]+\lambda w^2$，最後可得：$\frac{\partial(-\ln L(w,b))}{\partial w_i}=\sum_n -(\hat{y}^n-f_{w,b}(x^n))x_i^n+2\lambda w_i$，$\frac{\partial(-\ln L(w,b))}{\partial b}=\sum_n -(\hat{y}^n-f_{w,b}(x^n))$。本作業使用 adagrad，所以 learning rate of b：$\eta b_t=\frac{\eta b_0}{\sqrt{\sum_j^{t-1} gb_j^2}}$（其中 $gb_j=\frac{\partial(-\ln L(w,b))}{\partial b}$），learning rate of w_k：$\eta w_{k_t}=\frac{\eta w_{k_0}}{\sqrt{\sum_j^{t-1} gw_{k_j}^2}}$（其中 $gw_{kj}=\frac{\partial(-\ln L(w,b))}{\partial w_k}$），因此在每一 iteration 更新 bias 的 function 為：$b_{t+1}=b_t-\eta b_t*\frac{\partial L}{\partial b}$，更新 weight 的 function 為：$w_{k_{t+1}}=w_{k_t}-\eta w_{k_t}*\frac{\partial(-\ln L(w,b))}{\partial w_k}$。

```
b_grad=0.0
w_grad=np.array([0.0]*w_len)
for n in xrange(len(testSet)):
    z=np.sum(weight*testSet[n,:w_len])+bias                    wx+b
    f_x=1.0/(1+math.exp(-z))                                   σ(wx+b)
    b_grad=b_grad-(testSet[n,w_len]-f_x)                       ŷⁿ- f_{w,b}(xⁿ))
    w_grad=w_grad-(testSet[n,w_len]-f_x)*testSet[n,:w_len]
w_grad+=2*LAMBDA*weight                                        (ŷⁿ- f_{w,b}(xⁿ))x_i^n + 2 λ w_i

gb+=(b_grad**2)
wb+=(w_grad**2)
                                        adagrad使用的learning rate
bias=bias-alpha*(1./(gb**0.5))*b_grad
weight=weight-alpha*(1./(wb**0.5))*w_grad
```

2. Describe your another method, and which one is best

另一個實作的方法是 ridge regression 的 closed form，即 $w=(X^TX+\lambda I)^{-1}X^TY$，w 為 weight vector，X 為 features，Y 為 label vector，λ 是為了使 weight 盡可能小以減少 noise。

λ	0	0.1	1	10	100	1000
#(ŷ≠y)	79.25	79.75	79.75	86.0	141.5	387.75

但是根據上表，λ 越小越好，最好的 performance 發生在 λ=0 時。上表(即之後的圖表)的 #(ŷ≠y)是根據 4-cross validation(將 train set 隨機切四份)得出之平均分類錯的 data 數。λ=0 時其實就變成 linear regression 的 closed form。

Feature (0~56)	Logistic regression		ridge regression	Kaggle
	-lnL(w,b)	#(ŷ≠y)	(#(ŷ≠y))	(LR)
a. 54,56	216.727738641	68.5	79.25	0.93667
b. 54,56,18	214.44125799	67.75	79.25	0.94333
c. 54,56,49	214.680338999	69.5	75.25	0.94000
d. 54,56,37	215.939247675	67.75	78.75	0.93667

[在此鍵入]
資工所 碩一 黃郁庭 R05922038

e. 54,56,37,49	215.080481218	69.25	76.0	0.94000
f. 54,56,18,37	214.139131001	68.25	82.5	0.94333
g. 54,56,18,49	212.508670466	69.0	75.5	0.94333
i. 54,56,18,1+14	215.176719047	70.5	94.0	-

上表為 Logistic regression 與 ridge regression 之比較，feature 共有八種類型：a 為將第 54 與 56 個 feature 取 log，b 為將第 54、56 與 18 個 feature 取 log，以此類推，i 為除了將第 54、56 與 18 個 feature 取 log，另將第 1 與第 14 個 feature 相加然後整組 feature 只剩 56 種 feature，會將這些 feature 取 log 是因為用 decision stump 算出這些 feature 區分類別的效果較差，所以試著取 log，而將 feature 相加只是想看若兩個 feature 有相關性(例如第 1 與第 14 個 feature 分別代表 address 與 addresses)，是否能藉由相加來降維。由上表可知 Logistic regression 的效果較 ridge regression 好(#(ŷ≠y)都較小)，且#(ŷ≠y)變化大致一致。

若由-lnL(w,b) (這裡的-lnL(w,b)是根據 4-cross validation 將 train set 隨機切四份得出之平均 loss)作為衡量效果的依據，當將第 54、56、18、49 個 feature 取 log 時 Logistic regression 效果會最好，但若根據#(ŷ≠y) 作為衡量效能的依據，則 b 或 d 的效果會最好，而 ridge regression 在 c 時效果會最好。

normalize		no	$\frac{x - X_{min}}{X_{max} - X_{min}}$	$\frac{x - X_{mean}}{X_{SD}}$
Logistic regression	-lnL(w,b)	213.885106961	234.006805428	221.117417407
	#(ŷ≠y)	66.25	79.0	71.75
ridge regression		131.0	82.75	131.0

上表為探討將 data normalize 之效果以及用何種 normalize 方法會較好。由上表可知 Logistic regression 不論是以-lnL(w,b)或#(ŷ≠y)計算效果結果都是不加 normalize 效果較好，而 ridge regression 則是做過(x-X_min)/(X_max-X_min)這種使所有 feature 值都介於 0~1 間的方法較好。

3. other discussion and detail

為了測試 Regularization 對 Logistic regression 的影響，這裡調整了 λ 以下表表示，可以發現在 λ=1 時-lnL(w,b)最小，而#(ŷ≠y)在 λ=0.1 最小，可能是因為 weight 都很小，所以這裡適合的 λ 也很小。

λ	0	0.01	0.1	1	10
-lnL(w,b)	215.7720	215.1572	212.0054	209.8470	230.4199
#(ŷ≠y)	68.75	68.5	67.75	70.25	77.25

這裡也調整了初始的 learning rate η ，如下表所示，當 η 在 0.1 時-lnL(w,b)與#(ŷ≠y)皆最小，因為這裡使用 adagrad 所以 learning rate η 在碟代過程中會隨著距離目標的遠近調整每次前進的步伐。

η	0.2	0.1	0.01	0.001	0.0001
-lnL(w,b)	217.4883	214.4412	230.7167	400.4490	606.4747
#(ŷ≠y)	68.75	67.75	77.0	118.0	153.0

在嘗試將各個 feature 畫出圖來時有發現部分 feature 在 feature 值超過 threshold 時都會是同種 class，所以挑了其中幾個 feature 找出臨界值，如下表所示，當做完 test(或 validation)後可

[在此鍵入]
資工所 碩一 黃郁庭 R05922038

以直接用這種 threshold 再直接調整 class，目前 threshold 的命中率都很高，但整個 test set(或 validation set)中拿達到這些臨界值的 data 數通常是個位數，因此效果有限。

Feature id	threshold	description
27	0.5	當 data 的 x_{27}>0.5 時，預測 data 的 class=0 (在 train set 中 x_{27}>0.5 之 data 共有 555 筆且 class 全為 0)
26	0.5	當 data 的 x_{26}>0.5 時，預測 data 的 class=0 (在 train set 中 x_{26}>0.5 之 data 共有 555 筆且 class 全為 0)
41	0.4	當 data 的 x_{41}>0.4 時，預測 data 的 class=0 (在 train set 中 x_{41}>0.4 之 data 共有 227 筆且 class 全為 0)
28	0.5	當 data 的 x_{28}>0.5 時，預測 data 的 class=0 (在 train set 中 x_{28}>0.5 之 data 共有 555 筆且 class 全為 0)
31	0	當 data 的 x_{31}>0 時，預測 data 的 class=0 (在 train set 中 x_{31}>0 之 data 共有 186 筆且 class 全為 0)
40	0.1	當 data 的 x_{40}>0.1 時，預測 data 的 class=0 (在 train set 中 x_{40}>0.1 之 data 共有 119 筆且 class 全為 0)

另外其實也有實作結合 Aggregation 方法(其中的 blending)的 logistic regression，但因為後來聽說這不能算是第二種實作方法，因此沒有將程式碼放上來，但是他的 model 在 kaggle private set 的排名可以在前 20%(可是我沒選他當我的那兩筆…)，所以還是記錄下來。如下表所示，$g_t(x)$ 表從原本的 teain set G(x)中隨機取出部分的 data 去用 logistic regression 訓練模型，已經有證明顯示 $g_t(x)$中允許重複 data 的 performance 其實很好，如下圖的 h 和 i 所示，當取同樣的 data 數(1000)、一樣的跌代數(7000)、分同樣數量的組(15)去投票時，不允許 $g_t(x)$出現重複 data 的 i 的 performance 較 h 差，不過只做了一組而已還需要多加驗證。Aggregation 的投票概念其實有點像是將 feature 做 transformation(每組從不同角度去投票)，因此原本是 linear 的 logistic regression 可能因為經過類似 feature transformation 的過程後，其 performance 略好於原來的 logistic regression，原來的 logistic regression 訓練好模型後拿去預估整個 train set 目前發現最低的#(ŷ≠y)都超過 240，而 Aggregation 可以讓 logistic regression 訓練出的模型的#(ŷ≠y)低於 240，且 kaggle 上的排名確實也普片較好。由下表可知，根據 e,f,g 所示#$g_t(x)$增加不保證#(ŷ≠y)減少；根據 d,e 所示 iteration 增加也許可以降低#(ŷ≠y)；根據 a,e 與 b,d 所示#data in $g_t(x)$不一定能降低#(ŷ≠y)，可能與 iteration 大小有關。

	a	b	c	d	e	f	g	h	i	k	l	m	n
#(ŷ≠y)	240	241	239	244	230	237	231	231	238	237	237	244	248
iteration	5000	3000	2500	3000	5000	5000	5000	7000	7000	8000	6000	5000	10000
#$g_t(x)$	21	21	31	21	21	31	41	15	15	15	11	11	11
#data in $g_t(x)$	700	800	800	1000	1000	1000	1000	1000	1000	1200	1500	1800	2000
允許重複 data	o	o	o	o	o	o	o	o	x	o	o	o	o