機器學習 作業二

Machine Learning HW2

R05943040 電子一 林家禾

1. (1%) Logistic regression function.

```
rate = 8e-3
iter_num = 2650
lamb = 0
power_num = 1
w_num = feature_num*power_num
b = 0
b_table = np.zeros(iter_num)
w = np.ones(w_num)
entry = np.ones( w_num )
ada_b = 0
ada_g = np.zeros(w_num)
sample_num = train.shape[0]
grad_w = np.zeros(w_num)
batch = 5

w = np.random.randn(train.shape[1]*power_num) / train.shape[1] / train.shape[0] /
power_num
w = np.resize(w,(train.shape[1]*power_num,1))
batch_counter = 0

# interation 外層迴圈
for ite in range(iter_num):       # iter_num：iteration times
    idx = []
    random = np.random.permutation(sample_num)

     # 累加 N 次求出
     # ∂L/∂w 、∂L/∂b

    for n in random:     # shuffle training set
        idx.append(n)
        batch_counter += 1
```

```
        if(batch_counter==batch):       # minimum batch
            train_this = train[idx]
            ans_this = ans[idx]
            grad_b = 0
            grad_w = np.zeros(w_num)

            #用現有w、b求出預測值y = b + ∑_{i=0} w_i × x_i^n
            y = np.zeros([len(idx),1])
            y += b
            for power in range(1,power_num+1):
                w_this = w[0+ feature_num *(power-1): feature_num + feature_num
*(power-1)]
                entry_this = entry[0+ feature_num *(power-1): feature_num +
feature_num *(power-1)]
                prod = (train_this**power).dot(w_this)

                y += np.sum( prod,axis=1 ).reshape( len(idx) ,1)

            #丟入sigmoid function
            sigmoid = 1. / (1+np.exp(-y))
            temp = ( sigmoid-ans_this )

            # gradient 計算
            grad_b = np.sum(temp)
            for power in range(1,power_num+1):
                w_this = w[0+ feature_num *(power-1): feature_num + feature_num
*(power-1)]
                entry_this = entry[0+ feature_num *(power-1): feature_num +
feature_num *(power-1)]

                norm = (train_this**power).T.dot(temp)

                grad_w[0+ feature_num *(power-1): feature_num + feature_num
*(power-1)] = norm.reshape(feature_num,)

            # Adagrad parameters
            ada_b += grad_b**2
            for i in range(w_num):
```

```
            if(entry[i]==0):
                ada_g[i] = 0
            else:
                ada_g[i] += grad_w[i]**2

        grad_w[ada_g==0] = 0
        if(ada_b==0):
            grad_b = 0

    # gradient descent
        if(ada_b!=0):
            b = b - rate * grad_b / np.sqrt(ada_b)
        for i in range(w_num):
            if(ada_g[i]!=0):
                w[i] = w[i] - rate * grad_w[i] / np.sqrt(ada_g[i])

        dummy = 0
        batch_counter = 0
        idx = []

# predict result & compute accuracy
product = train.dot(w) + b
temp = 1+np.exp(-product)
sig = 1. / temp
pred = np.copy(sig)
pred[sig >= 0.5] = 1
pred[sig < 0.5] = 0
err = np.sum( abs(pred-ans) ) /np.shape(pred)[0]
acc_train = 1-err
```

2. (1%) Describe your another method, and which one is best.

Decision Tree

概念上是在高維空間中選擇分界線，以遞迴方式建立出更多界線來。每次選擇分界線時，窮舉各種分界線走向（窮舉每個維度），窮舉各種分界線位置（窮舉每筆數據），從中找到分類效果最好者。

評斷分界好壞以 Gini impurity 亂度來作評估，數值低代表效果好

每遇到一筆資料就判斷屬於哪一測，每建一題分界線就是在決策數中長出一個節點

效能比較

Accuracy	Logistic Regression	Decision Tree
Training set	0.929768	**0.9985003749**
Public set	**0.92667**	0.90667
Private set	**0.92333**	0.91000

決策樹對於已知資料的分類準確度極高，但對未知的資料分類笑我就沒有那麼好，類似 over-fitting 的感覺。相對 logistic regression 在 train 與 test 上的效能較為平均

3.(1%) TA depend on your other discussion and detail.

(1) feature 挑選

每次只對一個 feature 進行 training 觀察 accuracy 變化試圖找出哪一個 feature 對於分類較有幫助，但測試結果是每一項單獨 train 出來的分數都差不多。Training accuracy 也是以 57 個全選時為最高

(2) 正規化

經實測後正規化對於 error 並無實質上的幫助，在同樣條件下正規化系數越大 error 會越大，因此最後的版本並不使用正規化的技巧

探其原因，這可能是因為正確的模型遠比我們想像中複雜，還有很多因素沒有考慮到，因此 training data 對現在的模型來說過於不規律，才導致硬是平滑造成反效果

(3) public & private set

觀察排行榜可發現兩者分數的趨勢大致上成相反趨勢，最後 private 分高者通常 public 與 training score 較低。總和上次經驗這次將 iteration 與 learning rate 調小在 private set 上取得較好的成績