FinTech HW1

tags: `FinTech`

年級：資工碩二
學號：R08922125
姓名：張皓鈞

Problem 1.(a)

1. Split: 用dataframe裡的reindex配上permutation將原本的data shuffled，接著取shuffled後的前80%資料當training set，後20%資料當testing set。
2. Pop: 用dataframe.pop()取出需要被預測的col 'G3'，因為之後要做標準化所以先將G3取出。
3. Normalization: 用training set的mean及std各自對training set和testing set做標準化。

```
def data_preprocess(data, train_percent, predict_col):
    np.random.seed(7)
    data = data.reindex(np.random.permutation(data.index))
    training = data.iloc[:round(data.shape[0]* train_percent),1:]
    testing = data.iloc[round(data.shape[0]* train_percent):,1:]
    training_y = training.pop(predict_col)
    testing_y = testing.pop(predict_col)
    training_x = normalization(training, training)
    testing_x = normalization(testing, training)
    return training_x, training_y, testing_x, testing_y

def normalization(x, train):
    train_mean = train.iloc[:,:].mean()
    train_std = train.iloc[:,:].std()
    normalize_x = x.copy()
    for col in range(x.shape[1]):
        normalize_x.iloc[:,col] =(x.iloc[:,col] -train_mean[col]) /train_std[col
    return normalize_x
```

Problem 1.(b)

1. 用虛反矩陣求迴歸:利用np.linalg.pinv()找出虛反矩陣，接著將虛反矩陣乘上y即可算出model weight，最後將model weight乘上testing_x即可算出predict_G3。

$$\hat{b} = X^{+}y$$

$$MSE_{test} = \frac{1}{m}\sum_{i}(\hat{\boldsymbol{y}}^{(test)} - \boldsymbol{y}^{(test)})_i^2$$

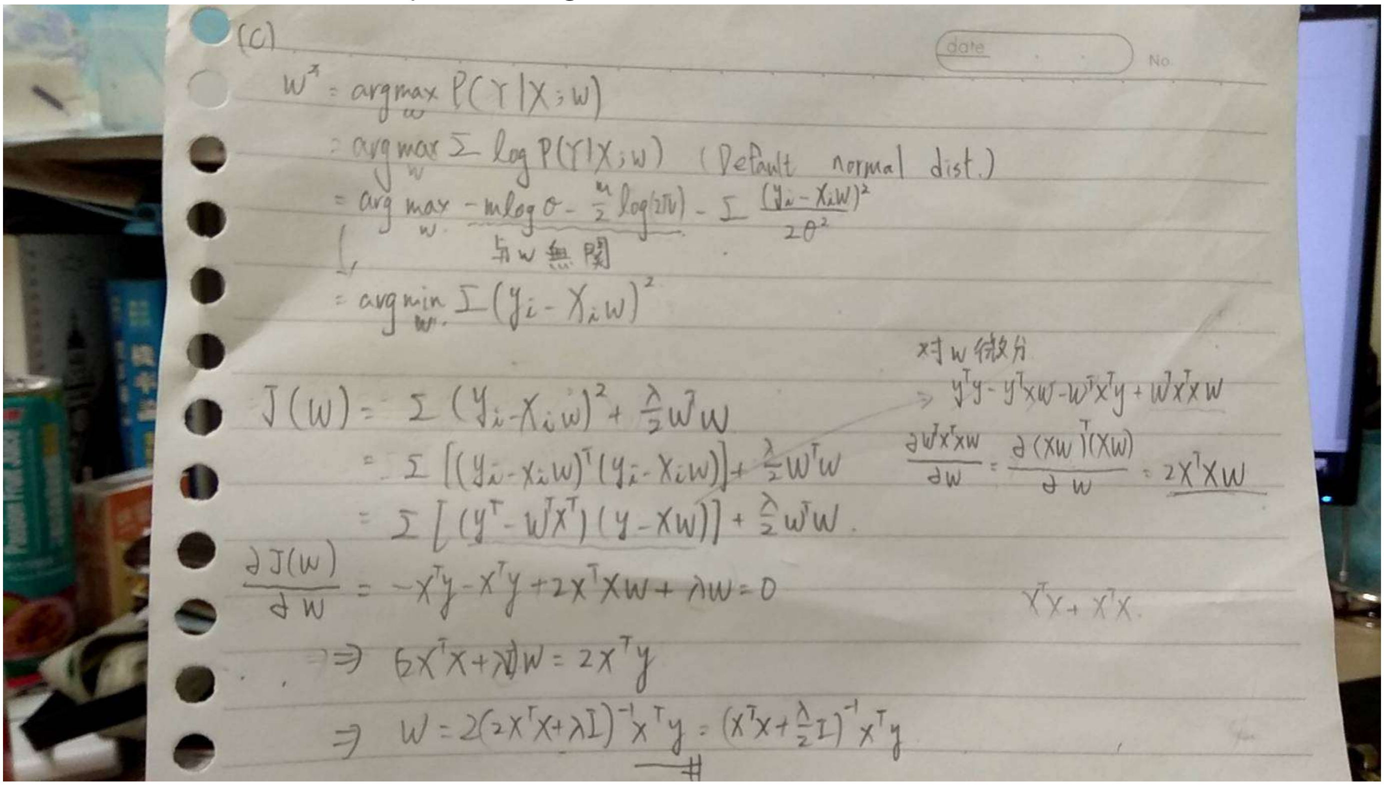
```
def linear(training_x, training_y, testing_x, testing_y):
    training_xpinverse = np.linalg.pinv(training_x)
    weight_linear = np.dot(training_xpinverse, training_y)
    predict_y = np.dot(testing_x, weight_linear)
    return predict_y, RMSE(predict_y,testing_y)

def RMSE(predict_y, testing_y):
    print(np.sqrt(np.sum((predict_y - testing_y)**2) / predict_y.size))
    return np.sqrt(np.sum((predict_y - testing_y)**2) / predict_y.size)
```

RMSE = 將上圖的MSE開根號即可，算出值為11.341623468423922

Problem 1.(c)

在原有的regression中加入weight decay，使結果比較不會overfitting。下圖為用maximum likelihood criterion 推導出optimal weight的過程。




```
def linear_regulization(training_x, training_y, testing_x, testing_y, lamda, bias):
    if (bias):
        training_x['bias'] = 1
        testing_x['bias'] = 1
    identiy_size = training_x.shape[1]
    weight_regularized = np.dot(np.dot(np.linalg.inv(np.dot(training_x.T, training_x)

    predict_y_regularized =  np.dot(testing_x,weight_regularized)
    return predict_y_regularized, RMSE(predict_y_regularized,testing_y)
```

RMSE = 11.341570734942113

Problem 1.(d)

為了在training中加入bias，需要在原有的training_x及testing_x裡額外加入一行，且值皆為1。$y = w_0 x_0 + w_1 x_1 + ... w_n x_n$，其中$x_0$皆為1，則即可在training的過程中算出$w_0$，此值即為bias。程式碼在上題中，將function parameter bias設為True。

RMSE = 1.6839557739866113

Problem 1.(e)

原先是以點估計的方式去推導線性回歸。此題採用概率分布的方式，y不被估計為單個值，而是被假定從normal dist.中抽取，最後可以得到下圖的公式:
u_m為posterior mean，將此當作model的optimal weight。

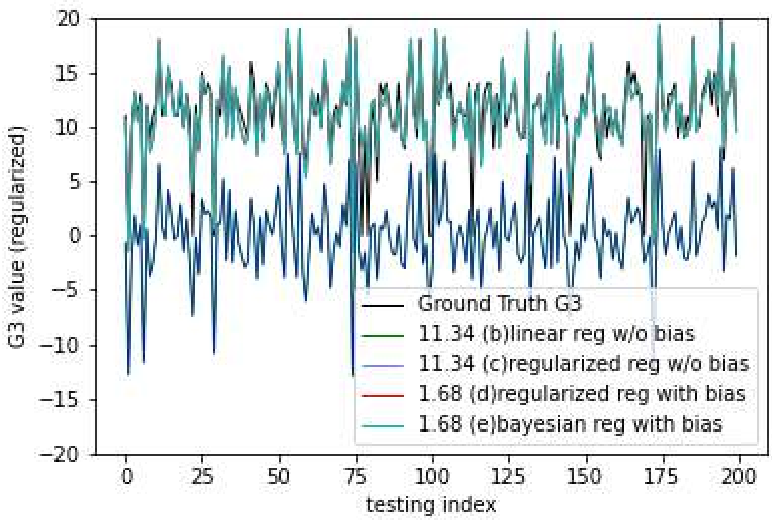
$$\text{If we set } \boldsymbol{\mu}_0 = \mathbf{0}, \Lambda_0 = \frac{1}{\alpha}\mathrm{I}$$
$$\Rightarrow \Lambda_m = (X^\top X + \alpha I)^{-1}$$
$$\boldsymbol{\mu}_m = (X^\top X + \alpha I)^{-1} X^\top \boldsymbol{y}$$

(same as frequentist linear regression with weight decay penalty of $\alpha \boldsymbol{w}^\top \boldsymbol{w}$ if we use $\boldsymbol{\mu}_m$ as the estimate of $\boldsymbol{w}$)

```
def linear_bayesian(training_x, training_y, testing_x, testing_y, alpha, prior_m
    training_x['bias'] = 1
    testing_x['bias'] = 1
    identiy_size = training_x.shape[1]
    cov_matrix = (1/alpha)*np.identity(identiy_size)
    posterior_mean = np.dot(np.dot(np.linalg.inv(np.dot(training_x.T,training_x)
    weight_bayesian = posterior_mean
    predict_y_bayesian = np.dot(testing_x,weight_bayesian)
    return weight_bayesian, predict_y_bayesian, RMSE(predict_y_bayesian, testing
```

RMSE = 1.6834058950921422

Problem 1.(f)



	RMSE
linear	11.341623468423922
regularized	11.341570734942113
regularized_bias	1.6839557739866113
bayesian_bias	1.6834058950921422

圖畫的沒有很清楚，其中bc兩條線幾乎重疊，而de兩條線也幾乎重疊，但從RMSE可以看出經過weight decay後的值比較小一點。外可以看到de兩條線距離ground truth比較近，這是因為de加入的bias，使得原本的回歸線不用強迫通過原點，有更好的彈性去做上下平移(y軸截距不需為零)，所以會使結果更好。

bc兩條線幾乎重疊應該是因為λ很小，幾乎沒有weight decay，於是在公式$(x^T x + \frac{\lambda I}{2})^{-1}$時因為$x^T x$的值很大，所以有沒有加上$\frac{\lambda I}{2}$並沒有太大影響。

de很接近的主要原因我想是因為α值與λ值的調整，由於α及λ都為1，且$x^T x$的值很大，則在過程 $x^T x + \frac{\lambda I}{2}$(或$\alpha I$)時，區別並不大。

Problem 1.(g)

由於原先我是拿全部的feature下去做training，但在此data內col屬性有些許不同，於是我找出此data與原本data的共同屬性，用原本data再重train bayesian reg.的weight，最後將此weight乘上此data的testing_x來預測G3。(這邊沒有tune α，預設為1)

結果請看.txt檔

Problem 2

此data set為人口資料，用以預測該人口年收入是否超過50K
這是我的預測結果，可以看到沒加上bias時有約79%的正確率，而加上bias後正確率高達84%。

```
error rate from linear:  0.21053439803439802
error rate from linear regulization:  0.21007371007371006
error rate from linear regulization with bias:  0.1613943488943489
error rate from linear bayesian with bias:  0.1613943488943489
```

```
RMSE from linear:  0.41413553221803867
RMSE from linear regulization:  0.4141372195793796
RMSE from linear regulization with bias:  0.34079382567361216
RMSE from linear bayesian with bias:  0.34079357109965014
```

從此data set中也可以看出，regulization加上weight decay後，RMSE和error rate都有些許下降，如果再加上bias後效果更是顯著。(這邊沒有tune α，預設為1)