Machine Learning HW2 report

姓名：顏修溫

學號：R05922034

系級：資工所 碩一

---

1. Logistic regression function. (藍色為程式碼　灰色為註解)

```
def  sigmoid(z):
    return 1. / (1. + np.exp(-z))    #由於丟進來的參數 z=b+wx 可能是很大範圍的數值，此 sigmoid 方法就是把這個大範圍的數值壓縮在 0 到 1 這個區間之內並輸出。
def  logistic(train_data, setting):
    learning_rate = 0.1  # 嘗試過各種可能後覺得 learning rate 設為 0.1 能得到較佳結果
    epoch_no = 5000      # epoch 次數
    feature_size = 57    # train data 中每一封信件裡的 feature 數量
    train_mail_no = len(train_data)  # train data 中信件的數量
    b = rd.uniform( -2, 2 )          # 隨機產生一個 bias
    w = np.array( [rd.uniform( -2, 2 )] * feature_size )
                                     # 每封信件有 x 個 feature，則隨機產生 x 個 weight
    ada_b = 0.  # 使用 adagrad 需要累計往後每個 epoch 的 gradient 平方值，用此變數累積
    ada_w = np.array( [0.] * feature_size )    # 同上，用來累積 gradient 平方值
    for epoch in range(epoch_no):              # train 的過程為 5000 個 epoch
        grad_b = 0.          # 儲存每個 epoch 算出來的 gradient 矩陣中 ∂L/∂b 的部分
        grad_w = np.array( [0.] * feature_size )  # 儲存 gradient 矩陣中 ∂L/∂w 的部分
        for n in range(train_mail_no):
            # 已知 ∂L/∂w = Σ_n -(ŷ^n - f_{w,b}(x^n)) x_i^n ，為了計算 ∂L/∂w 得要用此 loop 對所有信件進行加總
            y_hat = train_data[n][-1]  # 取得第 n 封信件的 y_hat
            x = np.array(train_data[n][1:-1]) # 取出第 n 封信件的 57 個 feature 作為 x
            f = sigmoid(b + np.sum(w * x))  # 計算 f=sigmoid(b+wx)
            grad_b += (y_hat - f) * (-1)    # 計算 gradient 矩陣中 ∂L/∂b 的部分
            grad_w += (y_hat - f) * (-x)    # 計算 gradient 矩陣中 ∂L/∂w 的部分
        ada_b += grad_b**2                  # 累計 adagrad 的數值
        ada_w += grad_w**2                  # 累計 adagrad 的數值
        b = b - grad_b * learning_rate * (1./(ada_b**0.5))     # 更新 bias
        w = w - grad_w * learning_rate * (1./(ada_w**0.5))     # 更新 weight
```

```
return (b, w)
```

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

我所使用的另外一個方法是 Linear regression，從課程中覺得 Linear regression 和 Logistic regression 概念上其實有非常多雷同之處。在實作上最主要的差異，就是 Logistic regression 多了一個 sigmoid function，原本 y = b + wx 的數值得要通過這個 sigmoid function 後輸出的值就會被壓縮在 0~1 之間。雖然兩者相像但一般來說都還是使用 Logistic regression 來進行二分類。所以這就引起我的興趣，兩者既然這麼像，那如果我把 Linear regression 也拿來分類的話，效果究竟會是如何呢？

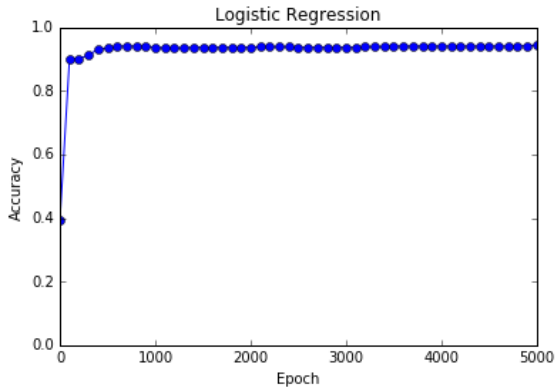

Fig.1 Logistic regression
(learning rate=0.1, no regularization)

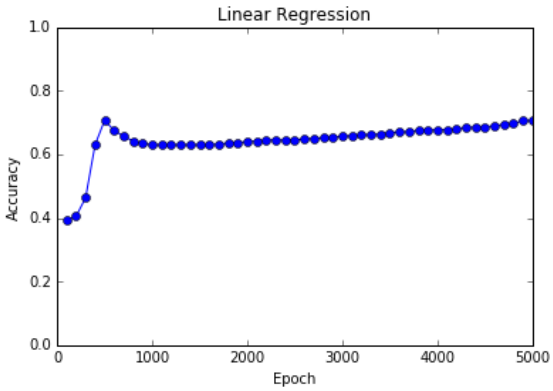

Fig.2 Linear regression
(learning rate=0.1, no regularization)

由上面兩張圖可以看出比較，Logistic regression 無論是收斂時間或是精確度都明顯優於 Linear regression。Logistic regression 大概在第 500 個 epoch 就收斂到 0.93 的精確度；而 Linear regression 的精確度則是極緩慢地上升，即使到第 5000 個 epoch，都還只有約 0.7 的精確度。而從 Fig.2 尚看不出 Linear regression 收斂後的效能會是如何，所以我試著讓 Linear regression 跑個 10 萬次 epoch，發現 Linear regression 最後也只收斂到 0.91 的精確度(見 Fig.3)。所以結論是如果用 linear regression 來分類的話，不僅收斂的效率極差、精確度也不及 Logistic。

我在 Kaggle 上得到最好的分數是使用 Logistic 方法；而 Linear 方法收斂後大概可以得到 0.89 的精確度。

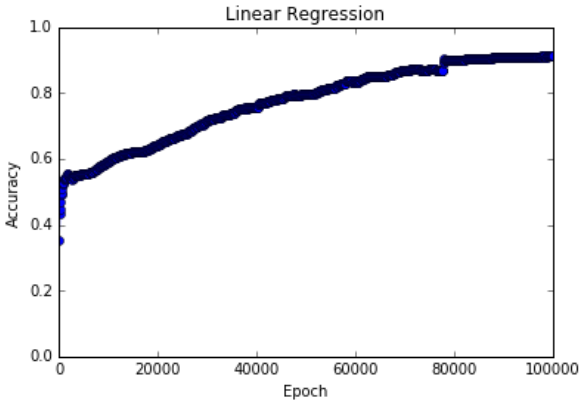

Fig.3

3. TA depend on your other discussion and detail.

這次作業的主題是關於分類，可以應用在非常多的生活情境上。為了對分類法有更廣的了解，所以我找了一些除了 Logistic regression 之外的分類方法，在這些資料中我覺得 SVM(support vector machine)算是比較大宗而且具有穩固理論基礎的方法，經過了解這個方法的理論與實作後，我對 Logistic regression 和 SVM 之間的差異有一些理解，想要在這裡分享討論一下：

A. Logistic regression 透過 sigmoid 映射方式，大大減少了離 hyperplane 較遠的資料點的權重，相對來說就是提升與 hyperplane 較相關的點的權重；SVM 則是只考慮 support vector，也就是只參考和分類最相關的那些少數點而已，接著拿這些點去歸納出 hyperplane。兩者的做法都是增加對分類影響較大的資料的權重，但 SVM 只考慮和分類較相關的點，而 Logistic 則是每個樣本點都會有貢獻。

B. SVM 有多種 kernel 選擇，可以用於各種線性分類與非線性分類，用途較廣；而 Logistic regression 只用於線性分類，training 的效率較高。

C. 把一個新的樣本丟入 Logistic regression 可以得到此樣本是屬於哪一類別的概率解釋，但 SVM 則無法。