B02901024 電機四 鄭佳維

Machine Learning Report~

Logistic regression

1. Logistic Regression function:

```
//initialization of logistic parameters
double bparameter=0;
double wparameter[57]={0};
double iter_diff[4001]={0}; // for future storage of y-f(x)

//One iteration of logistic regression
for(int itercounter=0;itercounter<MAX_ITERATION;itercounter++){
 if (itercounter%250==0){ //display use
  cout<<setw(7)<<itercounter<<' ';
  cout<<setw(15)<<wparameter[13]<<setw(15)<<wparameter[47]<<endl;
 }
//one iteration
//calculate all iterated difference (y-f)
for(int dat=0;dat<=4000;dat++){
    double iteration_ans = bparameter;
    for(int i=0;i<57;i++)
      iteration_ans += wparameter[i] * traindata[dat][i];
    // sigmoid
    iteration_ans *= -1;
    iteration_ans = exp(iteration_ans);
    iteration_ans = 1/(1+iteration_ans);
    // end sigmoid
    iter_diff[dat] = traindata[dat][57] - iteration_ans;
}
//calculate new b
double bcount=0;
for(int dat=0;dat<=4000;dat++)
    bcount += iter_diff[dat]; //summation of errors
bparameter += ( N_VALUE * bcount); //N_VALUE is learning rate
//calculate new ws
for(int i=0;i<57;i++){
  double wcount=0;
  double tempans;
  for(int dat=0;dat<=4000;dat++){
     tempans = iter_diff[dat] * traindata[dat][i];
     wcount += tempans;
  }
  wparameter[i] += (N_VALUE * wcount);
}
}
```

*** Description on Logistic Regression**

我的 Logistic Regression 所使用的 Model 為：

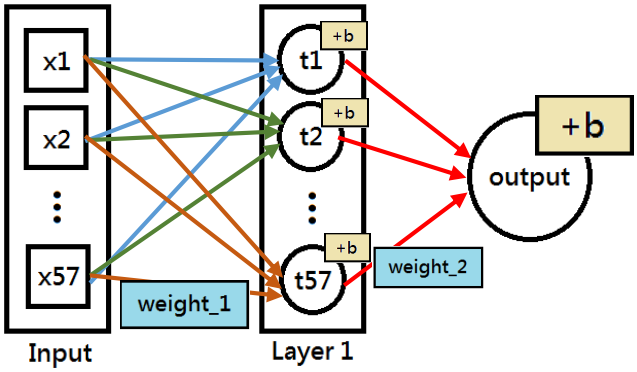
$$f_{w,b}(x) = \sigma(z), \text{where } z = b + \sum_{i=1}^{57} w_i \times x_i$$

而我選取的 feature 為全部 57 個 feature，每個 iteration 都會利用 cross entropy 的概念去調整 b 和所有 w 的值，使 Model 更加貼近 Training data。在 Logistic Regression 當中，我所有參數的初始值皆為 0。

2. Other Method Description

我使用的另一個方法，使用到了 Deep Leaning 的技巧。

我在這邊使用了兩層的 Logistic Regression，我以下圖來解釋要怎麼由原先的 57 個 feature 得到最後的解答(或是說預測)。



上面大致上就是我的方法了，其中 Input 到 Layer 1 會有一個 57*57 的矩陣(可以想像成 weight_1)，而 Layer 1 到 Output 則會有一個 1*57 的矩陣(可以表示成 weight_2，與原先 logistic 的 w 矩陣類似)。另外，每次做矩陣相乘，結果都要加上一個 bias。

其中，我在從 Input 到 Layer 1 和從 Layer 1 到 Output 都有做 logistic regression，也就是做完矩陣乘法以後結果會經過 sigmoid function。而 weight 和 bias 的更新無論是哪一層，也都是仿照 logistic regression 的模式進行操作。

*** Comparism on the methods**

首先考慮程式執行時間，因為每個 Iteration 花費的時間 Deep Learning 較 Logistic Regression 多出不少，因此 Logistic Regression 單位時間內能夠執行的 Iteration 數遠大於 Deep Learning。

然而，在 Performance 上，若花費的時間差不多，Deep Learning 較少 Iteration 出來的結果(Public set)就可以小小超越 Logistic Regression，若是允許 Deep learning 花費較多時間，其成效是比 Logistic Regression 來的好的，我 Kaggle 上最好的成績即是這樣來的。(而 Logistic Regression 執行時間拖久也沒辦法超過一個瓶頸值)

其實這樣的結果是可以預期的。以此題為例，在使用 Deep Learning 的情況下，我們等於是 maintain 了超過 57 倍的參數。這樣 Iterate 下來，在 performance 上超過相對陽春的 Logistic Regression，幾乎是必然！

3. More Discussions

(1). Normalization

我進行 Normalization 的方法為，針對所有 57 個 Feature 各自去進行 Normalize，完成後再去用 normalize 以後的數據來 train 同樣的 model。我 Normalization 使用的方法為：

$$x' = \frac{x - \mu}{\sigma}$$

這次的模型，Normalization 的效果相當顯著，原先進行 logistic regression，public set 最好的結果只能到 0.87，並沒有超過 baseline；而在做了 normalization 之後，public set 的結果便來到了 0.93。

原因在於，Normalize 以後的各個 feature 之範圍、數值會比較一致，對於後續的 training 影響力才會被調到差不多，不會有 feature 因為數字比較大而特別的 dominate。

(2). Validation Set 的使用

我這次的作業有使用 validation set。

方法為，我從原本 4001 個 training set 中，將最後的 501 筆切出來做為 validation set。Validation set 的功能有二：

第一個是在程式當中會不斷用 validation set 去檢測 training 的方向是否為正確的方向(透過 validation set 之 error 值)，若是連續錯誤則直接終止程式，將 validation set 所指出之表現最好的參數取出來用在 test set 上。

第二個則是在程式跑完以後，透過 validation set 做初步的成效估計，若是在 validation set 上表現太差，那麼也沒有丟到 Kaggle 裡面的意義了(畢竟一天只有五個機會)。如此可以幫我做 prediction 的篩選。

使用 Validation set 使我的 Kaggle 值基本上不會太差，也的確幫助提升了一些些 Logistic Regression 得到之分數。

(3). Mini batch

我這次的作業有嘗試 Mini batch。(Logistic Regression Part)

我 Mini batch 的做法為，在隨機看過部分的 training data 之後，就對 bias 和各個 weight 的值做出變動—如此的好處在於變動次數更多更快，另外每次的變動更有隨機性，可能可以跳出 local optima。

但是我跑 mini batch 的結果，和直接跑 logistic regression 一致(如上所述，我的 logistic regression 有個瓶頸值在)，因此最後到了 deep learning 我便沒有繼續使用。