B02901024 電機四 鄭佳維

Machine Learning Report~

Linear regression

1. Linear Regression function by Gradient Descent:

```
//initialize parameters
double bparameter_now=0;
double wparameter_now[18][9]={0};
double iteranswer[12][471]; // Value of (b + sum(w * x_n) )
double realanswer[12][471]; // Value of y
 // Loss function will be realanswer - iteranswer  !!

//real PM2.5 answer storage(y)
for(int month=0;month<12;month++){
  for(int hour=9;hour<480;hour++)
    realanswer[month][hour-9]=traindata[month][9][hour];
}
//Iterations of Gradient Descent
for(int itercounter=0;itercounter<MAX_ITERATION;itercounter++){
//calculate all iterated PM2.5
for(int month=0;month<12;month++){
  for(int hour=0;hour<471;hour++){
    double iteratedPM = bparameter_now;
      for(int i=0;i<18;i++)
       for(int j=0;j<9;j++)
       {
          iteratedPM += wparameter_now[i][j] * traindata[month][i][hour+j];
       }
    iteranswer[month][hour] = iteratedPM;
  }
}
//calculate new b with Gradient descent
double bcount=0;
for(int month=0;month<12;month++)
  for(int hour=0;hour<471;hour++){
    bcount += (realanswer[month][hour] - iteranswer[month][hour]);
}
bparameter_now = bparameter_now + ( N_VALUE * bcount);
//calculate each new w with Gradient descent
for(int i=0;i<18;i++)
  for(int j=0;j<9;j++){
     double wcount=0;
     double tempans;
     for(int month=0;month<12;month++)
       for(int hour=0;hour<471;hour++){
         tempans = realanswer[month][hour] - iteranswer[month][hour];
         tempans *= traindata[month][i][hour+j]; //Chain Rule term
         wcount += tempans;
     }
     wparameter_now[i][j] = wparameter_now[i][j] + (N_VALUE * wcount);
}
}
```

2. Method Description

我 Kaggle 最好的成績即是以 Linear Regression 得到，因此在這邊我會一起說明兩者使用的方式。

首先是最基本的 Linear Regression，我選擇了最終要預測的資料(第十小時之 PM 2.5)作為 y，而前 9 個小時的各 18 項觀測資料作為 feature (X)，來進行線性的 Model。

而我的 Model 使用的便是：

$$y = b + \sum_{i=1}^{144} w_i \times x_i$$

如此搭配 Gradient Descent，每個 Iteration 都去調整 b 和所有 w 的值，去使 Model 更加貼近 Training data。其中，我的 b 和各個 w 的初始值皆是設定為 0。

最後，在 Kaggle 最好的成績上，我和 linear regression 版本唯一的不同僅僅是移除了一些和 PM 2.5 較不相關的 Feature，這個部分會在此報告的最後面進行近一步的討論。

3. Discussion on Regularization

其實最一開始的優化，我就是去進行 Regularize 的動作。

Regularize 就是在 Loss function 處加上一個 $\sum w_i$ 項，目的是為了讓各個 train 出來的參數(w_i)值能夠小一些，反映在曲線上就是指 train 出來的函式能夠比較平滑，而通常較為平滑的曲線都能夠比較準確的預測出。

我調整了各個不同的 λ 值(即$\sum w_i$的係數)，我得到了以下結果：

λ value	Public Score
0.01	9.92566
0.0001	6.99684
0.000001	5.85853

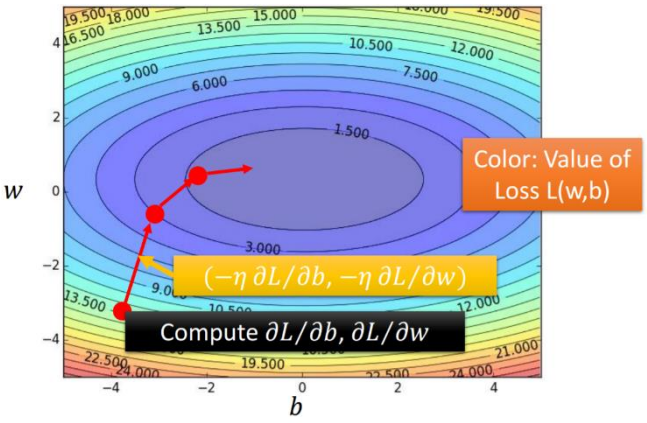
即使 λ 已經非常低，終於壓在 baseline 裡面，依然比我完全不進行 regularization 時得到的分數還要差。然而透過觀察我最終得到的 w 值，確實也可以發現各 w 值比不做 regularize 時小了一些。

因此，我認為這份作業中，雖然進行 regularize 確實有助於讓最終函式顯得平滑，但是對於整題 model 的效益而言，並沒有很大！

4. Discussion on Learning rate

Learning rate 大概是整個 Regression 最重要的一個參數值了。

Learning rate 的選擇決定了在 Gradient descent 的過程當中，每次參數去變化時所更動的幅度(因為其為和 Gradient 的比例關係)。



如右圖，可以看出 Linear Regression 下整個 Gradient Descent 的概況。在 Learning Rate 太大的情況下，每次參數的移動幅度可能太大，導致越過了 optima 而到了對面去，這樣在一次次 train 之下會導致發散的結果。而若是 Learning Rate 太小，則會導致移動的過程相當緩慢，需要非常多 Iteration 才能找到 optima。

對應到我上面的程式，N_VALUE 即是 Learning Rate，在經過一番嘗試以後，我選擇以6×10^{-10}作為我的 Rate。如上所述，因為 Learning Rate 太大參數會馬上趨近於 nan，太小則會使程式運作太慢，6×10^{-10}是我嘗試後所選取的折衷點。

5. Other Discussions

(1). 加入x^2項

我有嘗試過更改 model 為：

$$y = b + \sum_{i=1}^{144} w_i \times x_i + \sum_{i=1}^{144} w'_i \times x_i^2$$

不過這樣做的結果是發生了 overfitting 的現象，在 test case 上面表現完全不如原本來的出色，於是我便改回了只有一次方項的 Model。

我的 Kaggle 上超過 10 的數據都是由這個二次方 Model 所產生。

(2). Normalization

我進行 Normalization 的方法為，針對氣象局的 18 個 Attribute 各自去做 Normalize，完成後再去用 Normalize 以後的數據來 train 同樣的 model。

這樣的結果是在相當少(約 50000 次)的 Iteration 下便能達到和我原本不標準化時(1000000 次)差不多的水準，但是分數上還是比較差。然而，問題是沒過多久就會 overtrain，一樣出現 overfit 的情況。

(3). Training feature 的篩選

我跑了氣象局 18 項觀測資料和 PM 2.5 的相關係數，將低度相關的資料剃除在我的 feature 之外。低度相關資料為 CH4、RAINFALL、WD_HR 三項，於是我把他們從 model 當中移除。

移除的結果是整個 public score 結果提升了相當多，而我最後 kaggle 上的成績即是使用了這個成果。