Machine Learning: HW1 Report

November 14, 2016

Professor Hung-Yi Lee

電機三 B03901018 楊程皓

1. Linear regression function by Gradient Descent.

使用$y = b + \vec{w} \cdot \vec{x}$, x 取欲得pm 2.5 資料的前九小時的 NO2, pm 2.5 兩個欄位，即 $2 \times 18 = 36$ 項為input data. 故 w 亦有 18 項，總體 linear regression function 需要train 19 項參數。

Gradient descent code:

```
1    // function used for the above regression
2    void Table::quadraticRegression(const int& lenOfTrain, const double& eta, double&↩
         b, double* const w, double* const z, const double& deltaStop, const int ↩
        lambda) const {
3        // lenOfTrain is 9, in this case
4        // b for constant term, w for linear terms, z for quadratic terms.
5        // lambda is 0, in this case, since not using regularization
6        // G_b_t, G_w_t, G_z_t for adagrad
7        double preError, error = 200, gradient_b, regularization, G_b_t = 0;
8        double *gradient_w = new double[lenOfTrain],
9               *gradient_z = new double[lenOfTrain],
10              *G_w_t = new double[lenOfTrain],
11              *G_z_t = new double[lenOfTrain];
12
13       for (int i = 0; i < lenOfTrain; ++i) {
14          G_w_t[i] = 0;
15          G_z_t[i] = 0;
16       }
17
18       int counter = 0, idle = 0;
19       while (true) {
20          ++counter;
21          preError = error, error = 0, gradient_b = 0, regularization = 0;
22          for (int i = 0; i < lenOfTrain; ++i) {
23             gradient_w[i] = 0;
24             gradient_z[i] = 0;
25          }
26              // months is a data member of Table, each containing 20 * 24 hours ↩
                    data in the month
27              // Month::quadraticFunc will accumulate error, gradient_b, gradient_w↩
                    , gradient_z in the month with given parameters
28          for (int i = 0; i < Table::numMon; ++i)
29             months[i].quadraticFunc(lenOfTrain, b, w, z, error, gradient_b, ↩
                   gradient_w, gradient_z);
30              // calculate average error square
31          error /= (12 * (480 - lenOfTrain));
32          cout << counter << ": " << error << endl;
```

```
            // adagrad of b
            G_b_t += square(gradient_b);
            // gradient descent of b
            b -= eta * gradient_b / sqrt(G_b_t);

            // regularization, which is 0 with lambda is 0
            for (int i = 0; i < lenOfTrain; ++i)
               regularization += lambda * (square(w[i]) + square(z[i]));

            for (int i = 0; i < lenOfTrain; ++i) {
                // regularization, which is 0 with lambda is 0
               gradient_w[i] += 2 * lambda * w[i];
               gradient_z[i] += 2 * lambda * z[i];
               // adagrad of w, z
               G_w_t[i] += square(gradient_w[i]);
               G_z_t[i] += square(gradient_z[i]);
               // gradient descent of w, z
               w[i] -= eta * gradient_w[i] / sqrt(G_w_t[i]);
               z[i] -= eta * gradient_z[i] / sqrt(G_z_t[i]);
            }

            if (preError - error < deltaStop)
               ++idle;
            else
               idle = 0;
                // if iteration over 10k times or difference too small for three ↩
                    times, end of training
            if (counter == 100000 || idle == 3)
               break;
         }

         delete[] gradient_w;
         delete[] gradient_z;
         delete[] G_w_t;
         delete[] G_z_t;
      }
```

2. Method.

linear_regression 與 kaggle_best 是一樣的執行結果。

Linear model: input 參數為前九小時的NO2, pm 2.5 共18項，皆以線性組合為預測結果，如上所述，共須 train 19 個參數, 以此 method 產生 kaggle_best.csv。

使用 adagrad 調節 learning rate, 沒有使用 stochastic gradient descent, 沒有使用 feature scaling, 沒有使用regularization. Iteration 停止條件為 10k 次 or 連續三次 Lost function 平均的平方值降低幅度小於0.00001。

除了上述方式外，也試過linear 只取pm 2.5, 取pm 2.5 加其中一項, 18個column 全取, 取前X 小時(X = 1　9 都試過)，在 public score 效果上都比上述方法差。

3. Discussion on regularization.

在各個training model 中，加入regularization 的結果都沒有更好或差異極小，可能是因為跟regularization造成的Loss function 來比，在error 裡的各項造成的微分項仍然大很多，故造成的影響極小

4. Discussion on learning rate.

以 adagrad 調節 learning rate 的大小，使 iteration 能夠收斂，且η 的初始值設定比較不影響結果。之前試過 learning rate 固定的情況，結果皆不佳並有時發散，可能原因為在一段時間後的 iteration 每一步都算微調並越來越小，故走太大步無助於取得 Loss function 的最小值。

5. Others.

我以 C++ 寫, O3 優化，故執行起來比較快，兩份 script 執行起來應都在幾秒以內結束。

其實我有做 feature scaling, 但結果發現在 adagrad 的影響下， feature scaling 並無任何效果，故後來就沒有使用了。(如要使用，在指令加上 –featureScaling 即可)。

其實理論上，考慮所有前九小時18個 column 應該是最好的解，對 training data 而言的確，average error 可以到5.7 左右，但可能被某些相關性很小的 column 所影響，在 public test score 很慘...故沒有使用它作為 kaggle_best, 之後若要做的話, 可能要每個 column 看取前幾小時個別設定, 找出最好的那個。