# Machine Learning 2016

## HW1 Report

b02901065 電機四 李洺曦

## 一、模型與程式設計

由於 test_X.csv 中給我們九天的資料要我們算出第 10 天的 pm2.5 值，而第 10 天則是沒有任何資料，因此直覺上我將前九天每天的 18 個 features 都當作參數，也就是有 18*9 個 features，並將第 10 天的 pm2.5 當作 answer 去 train，取資料的方式則是連續的，即取 1/1,0 時~9 時為一組，1/1,1 時~10 時為一組…，可跨天，如 1/1,23 時~1/2,8 時為一組，但不可跨月，共有 12*(24*20-9)=5652 筆資料。

資料存於 class dataset 中的 var_set，資料型態為 numpy.array，shape 為(5652, 162)，下圖為設置 linear equation 的過程

假設 equation 中各項為[x1, x2,x2^2, x3, x4]，order 則為[1, 2, 1, 1]，經過下面過程後，model_order 為 model_order[:, :, 0] = [1, 1, 1, 1]，model_order[:, :, 1] = [0, 1, 0, 0]，為三維陣列(層數為最大 order)並當作 order 參數傳入 equation 中。

```
17 #automatically build a array recording order
18 max_order = np.amax(order)
19 model_order = np.array([1 if order[i] > 0 else 0 for i in ra
   nge(var_shr_count * hr_count)])[np.newaxis, :, np.newaxis]
20 order -= 1
21 for j in range(1, max_order):
22     model_order = np.dstack((model_order, np.array([1 if ord
   er[i] > 0 else 0 for i in range(var_shr_count * hr_count)])[
   np.newaxis, :, np.newaxis]))
23     order -= 1
```

係數與常數項先設為 0:

```
25 #set all coefficients and bias to zero
26 coe = np.zeros((1, var_shr_count * hr_count, max_order))
27 bias = 0
```

在 datatype.py 中定義了 linear_equ 的 class，初始傳入上述 coe 與 order，將 self.coe 設為 coe 與 order 的 elementwise multiplication，即可讓沒有二次項的變數其二次項係數為 0，達到輕易修改 linear equation model 的目的:

```
84         self.coe = coe  * order
85         self.bias = 0.0
```

在 linear_equ 中的 ans 則為帶入變數值求解:

```
89     def ans(self, var):
90         answer = np.dot(var, np.transpose(self.coe[:,:,0]))
91         for i in range(1, self.max_order):
92             answer += np.dot((var ** (i + 1)), np.transpose(
   self.coe[:,:,i]))
93         return self.bias + answer
```

var 的 shape 為(5652, 162)，coe 的 shape 為(1, 162, max_order)，算 answer 的方法即將 coe 各層轉置後與 var 的次方做矩陣乘法，再將各層沿第 3 為相加起來，answer.shape = (5652, 1)，代表 5652 筆資料在當前 equation 的 y^。

執行以下程式碼取得 gradient:

```
66      #calculate gradient of coefficients
67      err_coe = equ.err_pd_coe(data)
68
69      #regularization if needed
70      err_coe += 2 * smoother * coe
71
72      #calculate gradient 0of bias
73      err_bias = equ.err_pd_bias(data)
```

gradient 的算法類似 answer，將實際的 pm2.5 與 y^相減，shape = (5652, 1)，做轉置後與 var 的次方做矩陣乘法，並將各層 stack 起來*-2 形成 coefficient 的 gradient，shape = (1, 162, max_order)。bias 則將實際的 pm2.5 與 y^相減*-2:

```
105     def err_pd_coe(self, data):
106         temp = data.get_var()
107         gra = np.dot(np.transpose((data.get_train_pm() - dat
    a.get_f_ans())[:, :]), temp)[:, :, np.newaxis]
108         for i in range(1, self.max_order):
109             gra = np.dstack((gra, (np.dot(np.transpose((data
    .get_train_pm() - data.get_f_ans())[:, :]), temp ** (i + 1))
    [:, :, np.newaxis])))
110         return -2 * gra
111
112     def err_pd_bias(self, data):
113         return -2 * np.sum(data.get_train_pm() - data.get_f_
    ans())
```

之後將 coe 與 bias 乘上 learning_rate 相減:

```
83          coe -= learning_rate_of_coe * err_coe
84          bias -= learning_rate_of_coe * err_bias
```

更新 coefficient 並重算 y^:

```
88      #change coefficient and bias
89      equ.change_coe(coe, bias, model_order)
90
91      #refresh y^
92      data.refresh_ans(equ)
95  def change_coe(self, new_coe, new_bias, order):
96      self.coe = new_coe * order
97      self.bias = new_bias
```

如果要使用 adagrad，則再每次 iteration 時將 gradient 的平方加起來，並在減去 gradient 時除上開根號:

```
78          accu_grad_squ += err_coe ** 2
79          accu_bias_squ += err_bias ** 2
80          coe -= learning_rate_of_coe * err_coe / (accu_grad_s
    qu ** 0.5)
81          bias -= learning_rate_of_coe * err_bias / (accu_bias
    _squ ** 0.5)
```

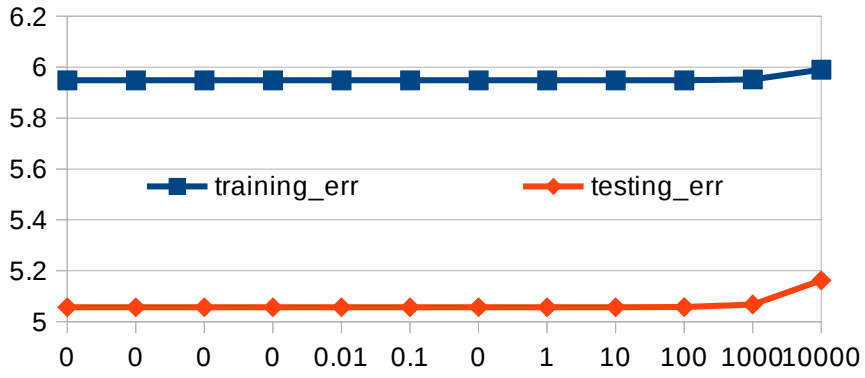
## 二、實驗與討論

1. Regularization

以下是實驗結果:

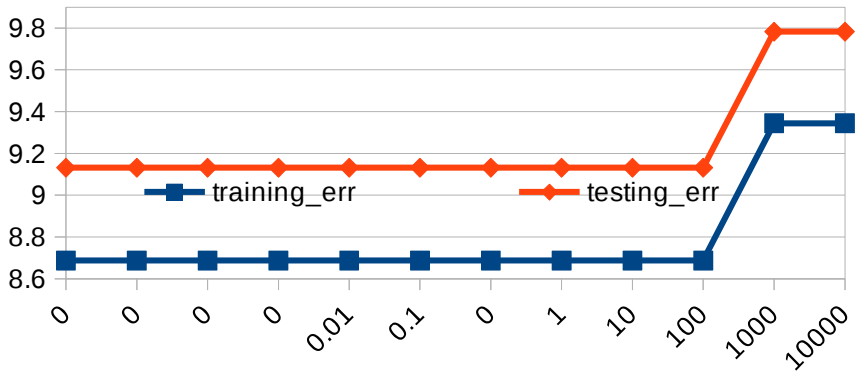
first order:

| lambda | train_err | test_err |
|---|---|---|
| 0.000001 | 5.94797 | 5.05681 |
| 0.00001 | 5.94797 | 5.05681 |
| 0.0001 | 5.94797 | 5.05681 |
| 0.001 | 5.94797 | 5.05681 |
| 0.01 | 5.94797 | 5.05681 |
| 0.1 | 5.94797 | 5.05681 |
| 0 | 5.94797 | 5.05681 |
| 1 | 5.94797 | 5.05682 |
| 10 | 5.94801 | 5.05691 |
| 100 | 5.94834 | 5.05787 |
| 1000 | 5.95175 | 5.06742 |
| 10000 | 5.99018 | 5.1624 |




在 testing error 上沒有下降的跡象，推測是一次 model 的 variance 較低，故使用二次實驗

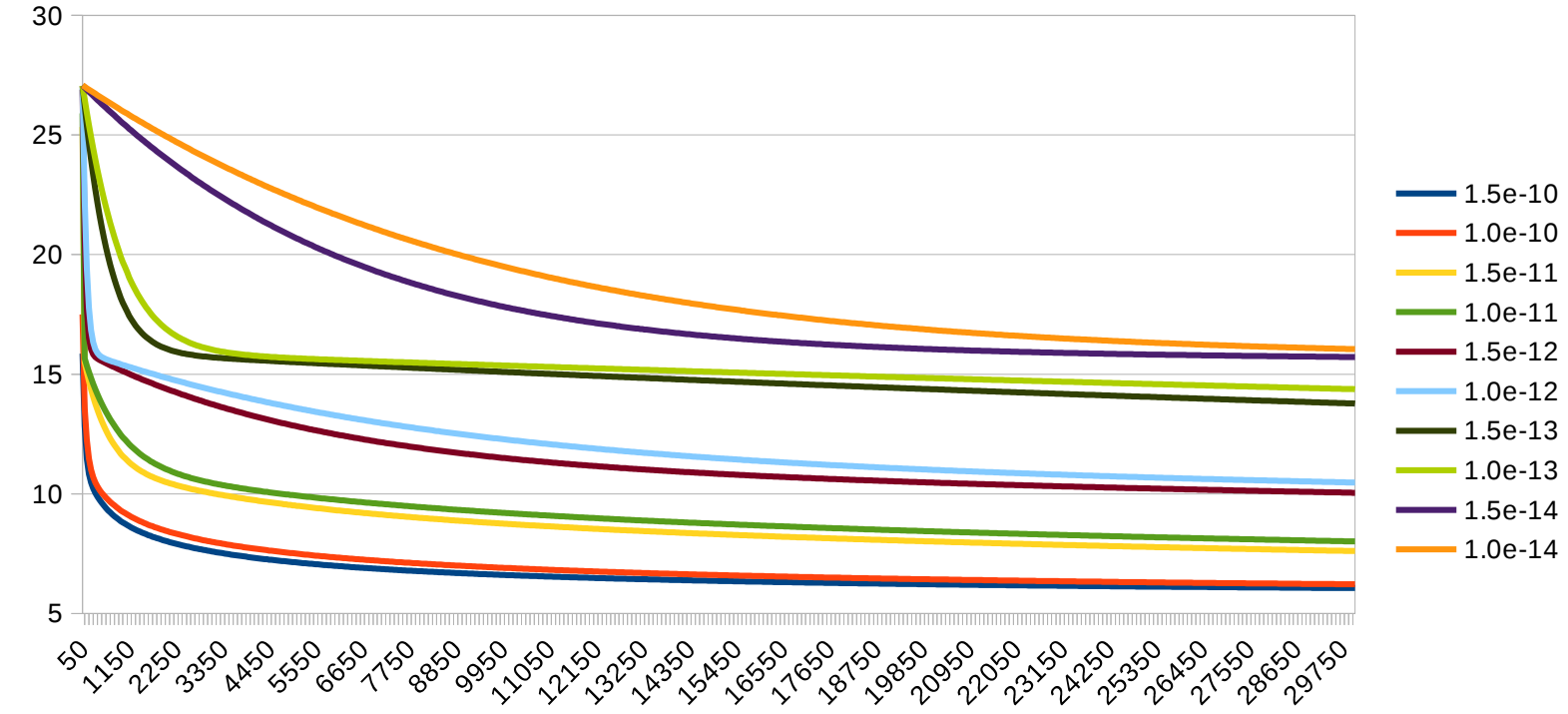
| lambda | train_err | test_err |
|---|---|---|
| 0.000001 | 8.68719 | 9.13129 |
| 0.00001 | 8.68719 | 9.13129 |
| 0.0001 | 8.68719 | 9.13129 |
| 0.001 | 8.68719 | 9.13129 |
| 0.01 | 8.68719 | 9.13129 |
| 0.1 | 8.68719 | 9.13129 |
| 0 | 8.68719 | 9.13129 |
| 1 | 8.68719 | 9.13129 |
| 10 | 8.68719 | 9.13129 |
| 100 | 8.68719 | 9.13129 |
| 1000 | 9.34411 | 9.78302 |
| 10000 | 9.34411 | 9.78302 |



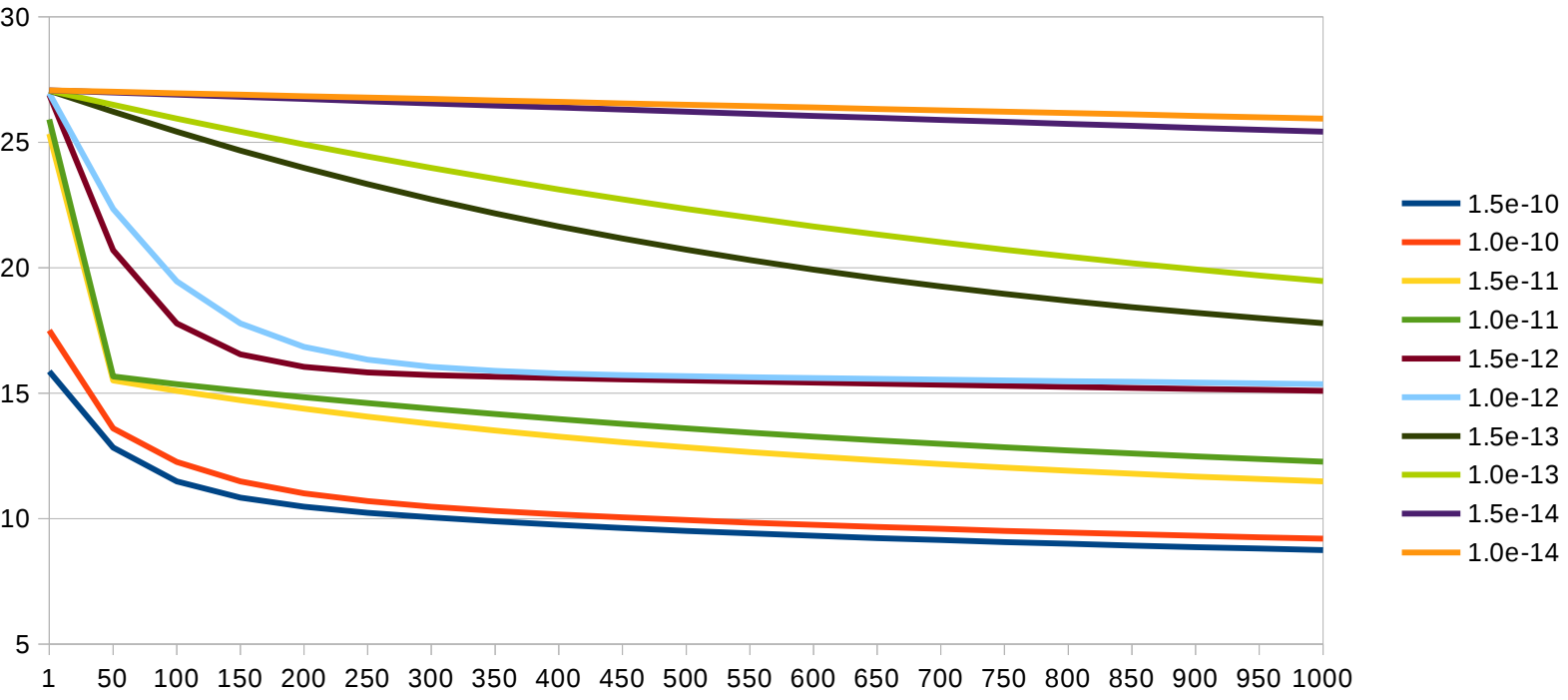
卻發生了 testing error 上升的現象，顯示應該是造成 overfitting 了，且 regularization 沒有顯著的效果，推測可能是 function bias 偏差太多，導致縮小 variance 造成了更嚴重的後果。

2. Learning Rate

以下為實驗結果(執行 30000 次，取第 1 次與每 50 次)：




在 1 次~1000 次的區間：




在變化率上大致呈現在愈小的學習速率下，成長率愈低的趨勢，但在 12 次方與 13 次方處較為合推測，合理推測 30000 次的 iteration 有點太少，無法很準確的看出整體的成長趨勢，但還是可以看出在 25000 次之後的成長率開始符合推測，而 13 次方與 14 次方在前期的大量下降，其原因很可能由一開始的 initial coefficient 猜測錯誤因而造成的巨幅下降所導致(見 1000 次趨勢圖)，由於速度過慢無法快速通過「陡坡區」，因此持續受到陡坡區的巨大斜率影響，而抵消了較低的成長率，造成整體成長較同次數的

其他 learning rate 設定快，但這也是因為仍然處於陡坡區所造成，整體看來，學習速率仍然不及其他較高的成長率設定。

3. N-fold Cross Validation

為了找出能比較符合 private set 的 linear model，我將資料以四季分為四筆來做 training 與 testing，以 first order linear equation 為基本，調整 learning rate 參數，以下為實驗結果:

| iteration | test_set | training_err | testing_err | average_err |
|---|---|---|---|---|
| 20w | test_set1 | 5.86964 | 4.90632 | 5.23184 |
| | test_set2 | 5.56934 | 5.9694 | |
| | test_set3 | 5.82633 | 4.6122 | |
| | test_set4 | 5.64502 | 5.43944 | |
| 40w | test_set1 | 5.8445 | 4.8483 | 5.25248 |
| | test_set2 | 5.5418 | 6.10673 | |
| | test_set3 | 5.80317 | 4.61336 | |
| | test_set4 | 5.61401 | 5.44153 | |
| 60w | test_set1 | 5.8369 | 4.82891 | 5.28305 |
| | test_set2 | 5.53432 | 6.21447 | |
| | test_set3 | 5.79604 | 4.63011 | |
| | test_set4 | 5.60641 | 5.45871 | |
| 80w | test_set1 | 5.83179 | 4.81862 | 5.304685 |
| | test_set2 | 5.5294 | 6.28496 | |
| | test_set3 | 5.79131 | 4.64316 | |
| | test_set4 | 5.60202 | 5.472 | |

可以看到，隨著 iteration 次數愈來愈多，traing_set 的 error 會逐漸減少，是相當合乎常理，但是在 average testing error 的表現卻是相反，推測為 iteration 的增加會減少 model 的 variance，但是 model 本身就有 bias 的情形下，variance 的減少反而使其在 testing set 的表現上愈來愈差。