Machine Learning Final Project (2019 Spring)

Wan-Cyuan Fan, NTUEE Qun-Wei Lin, NTUCSIE Cian-Ren Huang, NTUCSIE

B04502105 R07922164 R07922160

RandomForestReg

Abstract

在實驗的數據中，一共有三筆的不同數據: penetration_rate, mesh_size, alpha，因此我為每一組data製作一個Random Forest Regressor並使用不同的number of tree, max_depth做不同的嘗試，另外我在實作上，為了避免random的切割會影響結果，所以我重複做了三次再將之取平均，有點類似AdaBoost的感覺。

1. Pre-processing data

我們知道在整個資料10000維中，前面5000維度是所謂的MSD軌跡位移的second moment，而後面5000維度則是50組的VAC data速度的correlation。因此在切割data時我先使用MSD_feature_size = 100,VAC_feature_size = 100只進入部分的data。並且將所有data依照3:1的比例切成training testing set，載入後再進行training。

2-1. Experiment (feature and random sample)

在實驗上，因為RandomForestReg能夠調整的參數不多，因此我們針對樹的高度以及樹的數量來做調整看結果：

先做兩的簡單的試驗：

epx 1: 使用所有X_data取前10個feature當作實驗，搭配sk-learn套件RandomForestRegressor，n_estimators=500, max_depth=20。

```
<==================================================>
Training RandomForestRegressor model...
^[[A^[[Breg_pr done!
reg_ms done!
reg_alp done!
Processing time (sec) : 349.6165
<==================================================>
[Ein]
Processing [3/3]
WMAE =  108.3937234866972
NAAE =  3.830385736102859
<==================================================>
[Eout(valid)]
Processing [3/3]
WMAE =  113.15564290668529
NAAE =  4.404679407901245
<====================Done====================>
```

exp 2: 使用random sample方法，每一次只取1000個data，一樣前10個features，n_estimators=300, max_depth=20。

```
Training RandomForestRegressor model...
reg_pr done!
reg_ms done!
reg_alp done!
Processing time (sec) : 1.3925
<==================================================>
[Ein]
Processing [3/3]
WMAE =  57.890507014684246
NAE =  1.3140500830161903
<==================================================>
[Eout(valid)]
Processing [3/3]
WMAE =  100.27097253576613
NAE =  3.5725478948763345
<==================================================>
```

我們發現在簡單的random sample的情況下，可以有效下降Ein，並且降低在Ein上的overfitting，以下不同實驗以表格比較：

	Exp 1	Exp 2	Exp 3
樹的數量	300	300	300
樹的深度	20	20	20
Feature	10	10	10
Sample data	1000	2000	3000
Ein (WMAE/ NAE)	57.89/1.314	48.00/0.97	58.06/2.00
Eout	100.27/3.57	82.38/2.956	84.48/3.39

我們可以發現以Eout結果來看random sample可以下降WMAE的track，而NAE效果不顯著。另外也可以減少overfitting的效果。

接下來我們嘗試使用更大的feature數量，以及更深的樹深度及數量，因為我們觀察出來目前的data,feature數量不夠支持我們training後在Eout上表現不錯。

	Exp 1	Exp 2	Exp 3	Exp 4
樹的數量	600	600	600	500
樹的深度	20	20	25	50
Feature	200	300	300	300
Sample data	2000	4000	4000	3000
Ein (WMAE/ NAE)	36.08 1.02	35.88 1.02	35.88 1.18	23.96 0.45
Eout	68.15 1.82	68.15 1.819	68.99 1.812	63.67 1.429
Time	~ 40 mins	> 1 hr	> 1 hr	> 1 hr

從上面實驗我們可以發現，這樣的randomforestreg 演算法對於這樣的dataset是有一定極限的，並且會有嚴重的overfitting效果，在Ein只有20多的情況下，Eout高達60多，差距甚大。

2-2. Experiment (平均與**AdaBoost**)

嘗試解決這樣的問題，我們使用多次平均類似AdaBoost的方式。以樹數量300，深度20，10 features來比較看看：

	Exp 1	Exp 2	Exp 3	Avg
樹的數量	300	300	300	300
樹的深度	20	20	20	20
Feature	10	10	10	10
Sample data	3000	3000	3000	3000
Ein (WMAE/ NAE)	58.37 2.00	60.00 3.59	56.98 2.12	130.86 4.52
Eout	84.47 3.39	85.65 3.52	83.56 3.08	82.92 3.31

我們可以從表格實驗發現，再多次的平均在做WMAE,NAE的計算時，可以有效下降overfitting的效果，使得Eout下降，也可以避免某些特別突出的Eout數值達到平衡。

3. Summary :

使用randomforestReg**效果上，可以簡單做到還算不錯的分類效果，但如果要再往**上提升WMAE,NAE數值的話會花上較多的時間運算，並且可能會有很嚴重的overfitting的結果出現。

嘗試多組合下的結果我們得知RFR大約在63/1.4可能為最好的結果。

Deep learning structure

Abstract

在實驗的數據中，我們知道不知單有很多比的data而是data中的feature有時間上前後的關係，因此我想嘗試利用deep learning的CNN架構，萃取出其中的的特徵值並且做預測。

1. Pre-processing data

我們使用10000筆的data，並且在feature上我們取MSD_feature_size = 512,VAC_feature_size =512，合併後成1024的Tensor運算。並且切割training, validating 10000,300的數量做訓練。

```
total training data size : (47500, 1024)
total training label size : (47500, 3)
total testing data size : (2500, 1024)
train data size : (10000, 1024)
train label size : (10000, 3)
valid data size : (300, 1024)
valid label size : (300, 3)
Processing time (sec) : 82.5653
```

2. Model (Encoder / Fully connected)

```
(model): Sequential(
  (0): Conv1d(1, 8, kernel_size=(3,), stride=(2,), padding=(1,))
  (1): BatchNorm1d(8, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): MaxPool1d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
  (3): Conv1d(8, 64, kernel_size=(3,), stride=(2,), padding=(1,))
  (4): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (5): MaxPool1d(kernel_size=3, stride=1, padding=1, dilation=1, ceil_mode=False)
  (6): Conv1d(64, 128, kernel_size=(3,), stride=(1,), padding=(1,))
  (7): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (9): ReLU(inplace)
)
(fc): Sequential(
  (0): Linear(in_features=8192, out_features=2000, bias=True)
  (1): BatchNorm1d(2000, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace)
  (3): Dropout(p=0.5)
  (4): Linear(in_features=2000, out_features=500, bias=True)
  (5): BatchNorm1d(500, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (6): ReLU(inplace)
  (7): Dropout(p=0.5)
  (8): Linear(in_features=500, out_features=64, bias=True)
  (9): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (10): ReLU(inplace)
  (11): Dropout(p=0.5)
  (12): Linear(in_features=64, out_features=10, bias=True)
  (13): BatchNorm1d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (14): ReLU(inplace)
  (15): Dropout(p=0.5)
  (16): Linear(in_features=10, out_features=1, bias=True)
)
```

在optimizer的部分我使用Adam,SGD測試，我們針對不同的Y設計不同的model，因此總共有三個model : model_pr, model_ms model_alp，如此進行training。

3. Result :

上面是Adam在兩個epoch的結果，因為後面變化幅度不大，幾乎都落在200多的loss

```
Epoch [1/40], Iter [90/100] W_loss 332.3949 N_loss 12.8879 , LR = 0.001000
Epoch [1/40], Iter [91/100] W_loss 311.2670 N_loss 2.5239 , LR = 0.001000
Epoch [1/40], Iter [92/100] W_loss 331.5755 N_loss 2.7259 , LR = 0.001000
Epoch [1/40], Iter [93/100] W_loss 330.7531 N_loss 2.6962 , LR = 0.001000
Epoch [1/40], Iter [94/100] W_loss 319.2183 N_loss 3.0655 , LR = 0.001000
Epoch [1/40], Iter [95/100] W_loss 311.2213 N_loss 10.7957 , LR = 0.001000
Epoch [1/40], Iter [96/100] W_loss 318.9836 N_loss 3.3908 , LR = 0.001000
Epoch [1/40], Iter [97/100] W_loss 319.2772 N_loss 7.5183 , LR = 0.001000
Epoch [1/40], Iter [98/100] W_loss 302.6189 N_loss 3.2771 , LR = 0.001000
Epoch [1/40], Iter [99/100] W_loss 305.8322 N_loss 3.7231 , LR = 0.001000
Epoch [1/40], Iter [100/100] W_loss 314.4024 N_loss 3.6779 , LR = 0.001000
<==============================================>
Epoch [2/40],  W_loss 278.3571 N_loss 5.6723 , LR = 0.001000
<==============================================>
valid file done! W_loss 279.4158 N_loss 5.6521
<==============================================>
```

無法下降。因此我們改換成SGD嘗試：
我們發現其效果雖然下降了，但是依然在5個epoch後，保持不變，約落在WMAE 150上下。

```
Epoch [4/40], Iter [33/40] W_loss 163.7316 N_loss 2.0521 , LR = 0.000059
Epoch [4/40], Iter [34/40] W_loss 167.1451 N_loss 4.2545 , LR = 0.000059
Epoch [4/40], Iter [35/40] W_loss 174.0969 N_loss 4.1561 , LR = 0.000059
Epoch [4/40], Iter [36/40] W_loss 161.1934 N_loss 2.7992 , LR = 0.000059
Epoch [4/40], Iter [37/40] W_loss 158.4993 N_loss 3.3818 , LR = 0.000059
Epoch [4/40], Iter [38/40] W_loss 156.4544 N_loss 3.7985 , LR = 0.000059
Epoch [4/40], Iter [39/40] W_loss 166.1995 N_loss 14.8494 , LR = 0.000059
Epoch [4/40], Iter [40/40] W_loss 155.8297 N_loss 4.4196 , LR = 0.000059
<==============================================>
Epoch [5/40],  W_loss 155.8984 N_loss 6.3778 , LR = 0.000059
<==============================================>
valid file done! W_loss 155.7786 N_loss 6.3346
<==============================================>
```

4. Summary :

使用deep learning的架構時，我們發現可以大量處理很多資料，因為我們可以分批次送入訓練，但也會發現雖然可以閱讀更多的資料量，但實際預測效果卻不是很好，在WMAE皆超過100，雖然NAE可以壓到2以下，
但仍然不如前面randomforestreg效果要好，因此我們暫時放下使用deep learning的想法嘗試其他方式解決，雖然下降速度可以在大約5分鐘降到150左右。

Stacked Generalization

Abstract

在ensemble learning中，除了Bagging和Boosting會對data的橫向劃分之外，還有一个縱向加深的劃分方法，這稱之為Stacked Generalization (SG)，這在多個Kaggle比賽中被使用到，有很好的提升效果。

1. Pre-processing data

利用random forest來進行features extraction: 由於10000維的features實在太大，決定先用sklearn中的RandomForestRegressor先對data set進行一次完整的訓練，參數n_estimators = 100，樹的深度限制在100，而且必須對3種不同的target values都進行一次完整的訓練，因為不同的features可能對不同的target values有不同的影響，所以必須一一針對來選擇適當的features。

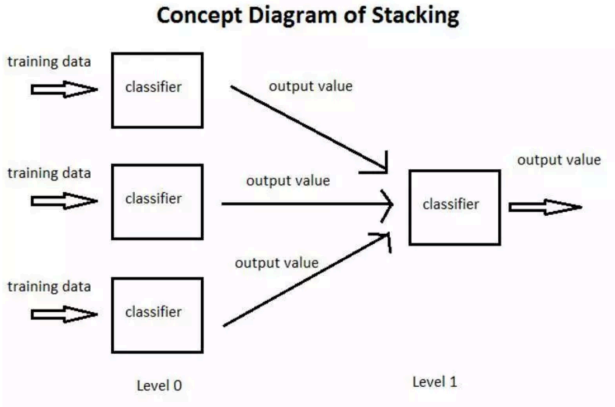
接著用SelectFromModel把比較重要的features萃取出來，把一些variance不滿足threashold的features先剔除掉。

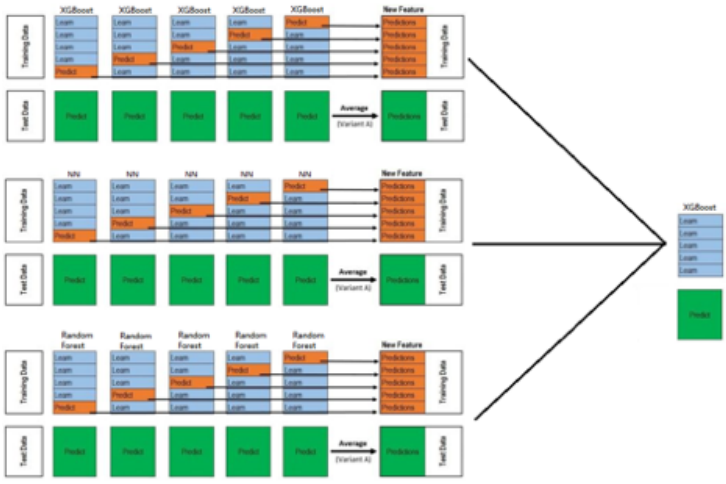
下表為針對不同target將features剔除後剩餘的features數量：

penetration_rate	1136
mesh_size	1480
alpha	1234

大幅壓縮了原本的features維度，所以接下來就在這些features上面進行其他model的訓練。

2. Model & Training





(1)第1層stack:

針對penetration_rate:

利用GridSearchCV針對不同的參數做選擇，第1層的base model是利用XG-Boost在這些降維過的features上訓練出來的model，發現利用1900棵trees，最大樹高7，最小權重和7是不錯的參數，在此將所有的taining data拆成5份，每一份data set有47500 / 5 = 9500個data，分別為data1、data2、...data5，將其分組並在這些data上面做trainging，也就是利用5-fold來做訓練，也就是5次的訓練之中，每次抽掉其中一組，只拿來做valid set的作用而不用來train model，而這組valid set也是用來進行early stop的重要工具，可以設定early_stopping_rounds = 100 來得到最佳的tress數量，當進行完上述步驟，就可以做出5個具有差異的model。

再訓練完其中一組model之後，必須對該組用來valid的data set和test data進行predict，如此一來，traing data中的每一個data都會得到一個predict value，而test data會被5個不同的model進行五次predict，進而得到5個不同的predict values，接著對這

5個predict values取平均，其後就把這些predict values當成全新的feature，重新加回到penetration_rate原本的1136個feature之中，現在penetration_rate變成可以使用1137個features來進行第2層model的訓練了，以此類推，也可以繼續操作其他不同的model（可能是參數不同或別種regression模型）來繼續增加第1層stack的features，降低最終模型的 Bias 和 Variance。

針對mesh_size和alpha:

利用GridSearchCV針對不同的參數做選擇，發現對mesh_size利用3000棵trees，最大樹高8，最小權重和10是不錯的參數，而對alpha利用1900棵trees，最大樹高8，最小權重和8是不錯的參數，一樣將所有的taining data拆成5份，且同樣也進行5-fold、

具有early stopping的訓練，現在mesh_size和alpha分別變成可以使用1481個features和1235個features來進行第2層model的訓練。

(2) 第2層stack:

而當做完上述的處理後，實際上又增加了其他不同的base model來進行第1層stack的features多樣性，例如NN、random forest等等的model。

第2層要做的事情基本上就是把第1層base model做出來的predict values當成features重新訓練一次，由於多了這些features，第2層的model訓練起來時速度明顯上升，loss curve和val_loss curve下降的速度變快了，且準確率也提升了，第2層採用了XGBoost當作最終的model，而stacking基本上可以繼續往上疊加層數繼續降低Bias 和 Variance。

3. Summary:

利用stacking的優點是可以最終模型的 Bias 和 Variance，但缺點可能需要耗費更多的計算資源，且在進行模型平均時需要特別小心，可能會比單一層的模型更容易發生overfitting的問題。

Comparison :

	RFR	CNN	SG
Data usage	4000	10000	47500
Feature usage	300	1024	3850
Best WMAE	63.67	153.04	43.535
Best NAE	1.429	3.98	1.04
Time consumption	~ 1.5 hours	~ 5 mins	~ 3 hours
Interpretability	較容易理解	無法確定其中特徵值的截取是否正確	實作較為複雜，但結構上是reasonable
Prons	運作上較為簡單，及三組不同的RFR結構就可以實作出來，另外在WMAE 80左右的要求下，所需要花的時間較少，大約2分鐘即可。	可以一次處理大量的資料，以從中獲得特徵值，以這個dataset而言，運算速度其實算快，大約3~4分鐘就可以從Loss 500下降到150左右。	較為細膩的操作，可以得到良好的WMAE,NAE loss，甚至可以降到45以下的水準，也可以將大筆資料分開stack，使得大量資料可以被讀取。
Cons	會有嚴重的overfitting效果，並且在WMAE 60之後很難再有更好的結果。	效果不佳，使用最直觀的encoder與FCL做出來的效果不是很好。	實作上較為複雜，時間消耗也更久，整體消耗運算資源。

Recommend THE BEST ONE of those approaches for each track:
在推薦不同的方法時，先考量需求：

1. 不在意時間與運算，希望WNAE,NAE最小： 兩個track 都使用SG 方法較適合。

2. 運算時間要求中間，但希望WMAE, NAE有一定的水準，甚至NAE表現要不錯：則兩個track 皆使用RFR適合。

Reference:

1. Stacked Generalization , “https://www.jianshu.com/p/46ccf40222d6?fbclid=IwAR1O2Ec6MEI9ckIFAJTQKImq_hrCILQZElZLk167k9AY62FFYjt-JcmaitpI”

Balance work loads:

Qun-Wei Lin, NTUCSIE : 其他許多models實作，以及SG 實作最高分。

Wan-Cyuan Fan, NTUEE : RFR /Deep learning model實作與嘗試，report 前半。

Cian-Ren Huang, NTUCSIE: 統整討論與report後半。