Machine Learning Final Project Report

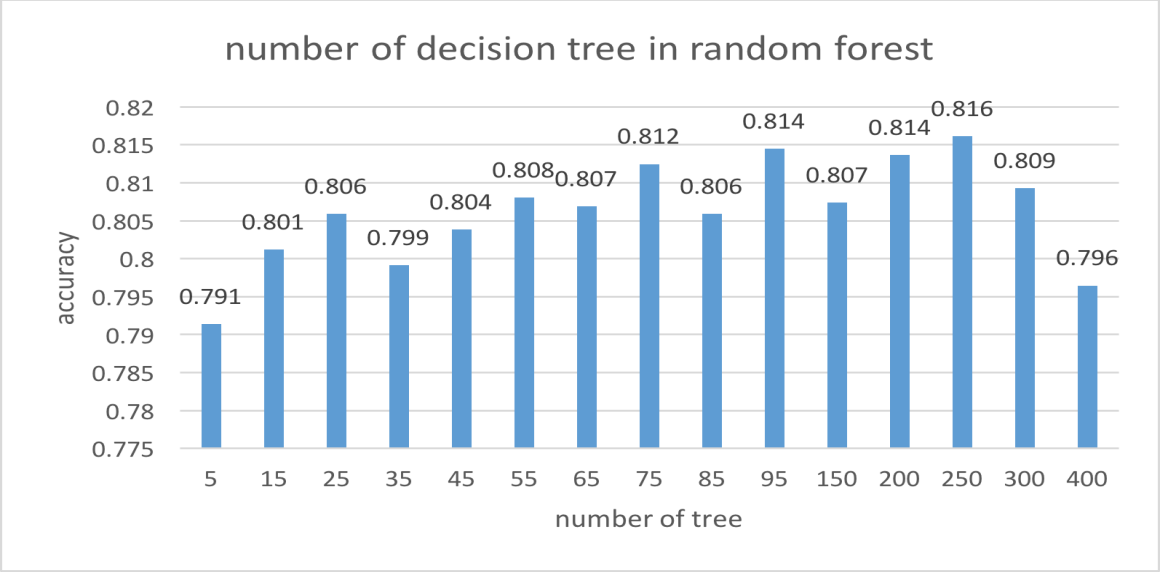
1. 選擇題目: Pump It Up (抽水機狀況預測)

2. Team name, members and your work division.
   Team Name: NTU_r05922096_唐三藏西天取經
   工作分配:
   - r05922096 李哲安 工作: Gradient Boosting Tree實驗
   - r04922101 黃振綸 工作: DNN model, 調整Gradient Boosting Tree 參數
   - r05922088 劉立傑 工作: 嘗試第二題,報告部分排版
   - r05922118 劉鎛漪 工作: Random Forest 實驗

3. Model
   這次嘗試的model有Random Forest, Gradient Boosting Tree, DNN:
   - Random Forest
     - Preprocessing/Feature Engineering:
       1. 移除 id: 和預測無關
       2. 移除 region_code, district_code, ward: 因為這些和location相關的feature可以從經度(longitude)和緯度(latitude)的feature得知
       3. 移除recorded_by: 所有data皆為同樣的值
       4. 移除 extract_type_group, extract_type_class: 和extract_type feature相同
       5. 移除water_quality: 和quality group feature 相同
       6. 移除quantity_group: 和quantity group feature相同
       7. gps_height, construction_year: 將值為0的data換成mean value
       8. date_record: 將date換算成timestamp

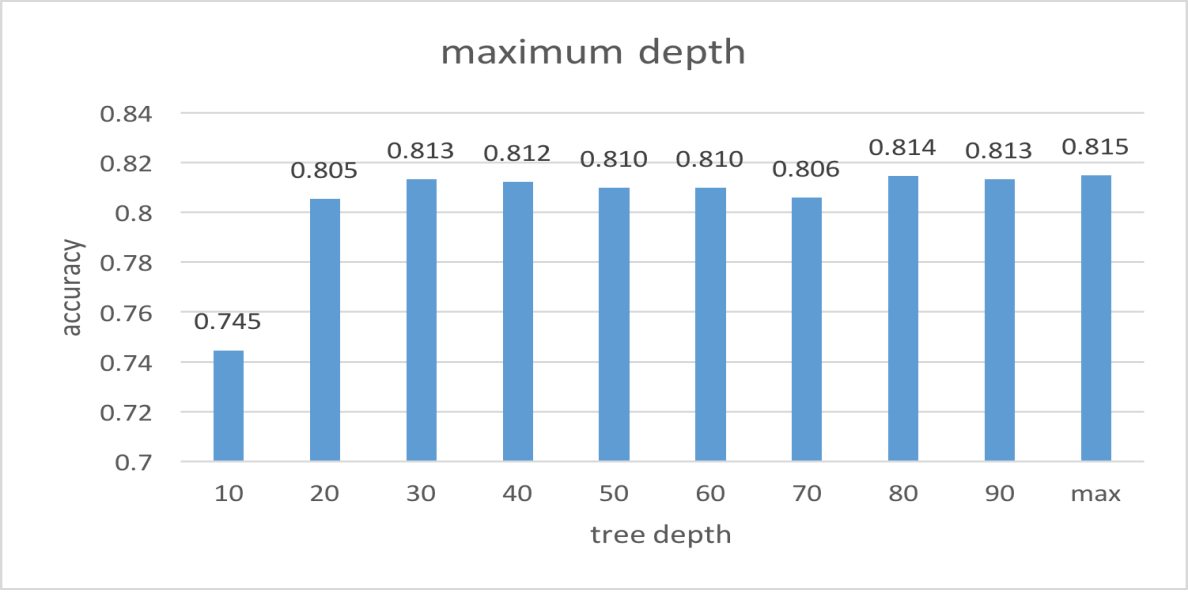
       其餘的data若有分類做one-of-n encoding，但同一feature的catogorial，只取有超過100的分類，避免input dimension過大的問題．得到的training input dimension為271．

     - Model Description and Experiments
       我們使用sklearn的random forest
       將training data切成training set 和 validation set，比較decision tree的數量對validation set accuracy的影響:



這張圖上可以看出tree的數量越多，performance越好，因為越不會有overfitting 的情況產生，但當decision tree達到一定的數量，這裡為250，performance就沒有顯著的提升，反而降低。

1. 用decision tree為100顆的random forest，比較max depth在validatoin set accuracy的影響:



調整decision tree的maximum depth，對performace沒有提升的效果，tree depth太小performance反而還下降，維持原本的default max depth較好，即leaf node只含有同一種類的class。

- Gradient Boosting Tree
  - Preprocessing/Feature Engineering:

    對訓練資料的前處理分為以下五個步驟:

    1) 首先我們先把training和testing的feature連接接在一起。

    2) 處理date_recorded的部分，把原始的date format切割成多個feature，一共切割成四個部份如下:
       a) year_recorder: 西元年份 2001~2013
       b) weekday_recorder: 星期幾 0~6
       c) yearly_week_recorder: 第幾週 0~52
       d) month_recorder: 第幾個月 1~12

       並增加一個age的feature來表示從construction_year到data紀錄日期之
       間過了幾年。

    3) 再來是處理非數字feature的部分，這邊我們把所有type為object的feature轉為數字來表示，也就是說k種不同的值則轉為0~k-1的數字來表示。一共處理了以下29類feature:
       ['funder', 'installer', 'wpt_name', 'basin', 'subvillage', 'region', 'lga', 'ward', 'public_meeting', 'recorded_by', 'scheme_management', 'scheme_name', 'permit', 'extraction_type', 'extraction_type_group', 'extraction_type_class', 'management', 'management_group', 'payment', 'payment_type', 'water_quality', 'quality_group', 'quantity', 'quantity_group', 'source', 'source_type', 'source_class', 'waterpoint_type', 'waterpoint_type_group']

    4) 除了以上兩點之外沒有做其他的處理，最後將連接在一起的data重新分割成train和test。

    5) 由於這題是分類問題 label的部分則直接轉為數字0~2。

  - Model Description

    我們使用了xgboost的library來幫助我們建立gradient boosting tree，並且參考了DrivenData github上 https://github.com/drivendataorg/pump-it-up/blob/master/mrbeer的code來做修改，主要有修改的地方為tree的最大深度

，下圖是model的主要參數設定:

```
'eval_metric': 'mlogloss',
'objective': 'multi:softmax',
'num_class': 3,
'max_depth': 64,
```

由於這次是分類問題所以選擇了softmax，而validation的部分根據DrivenData上計算公式這邊選了multiclass logloss，最後答案只有三類所以class數為3，Tree的最大深度設定為64層。

```
'subsample': 0.82,
'colsample_bytree': 0.5,
```

上圖是subsample的參數設定，data的subsample rate設定為0.82，而feature的subsample rate設定為0.5。

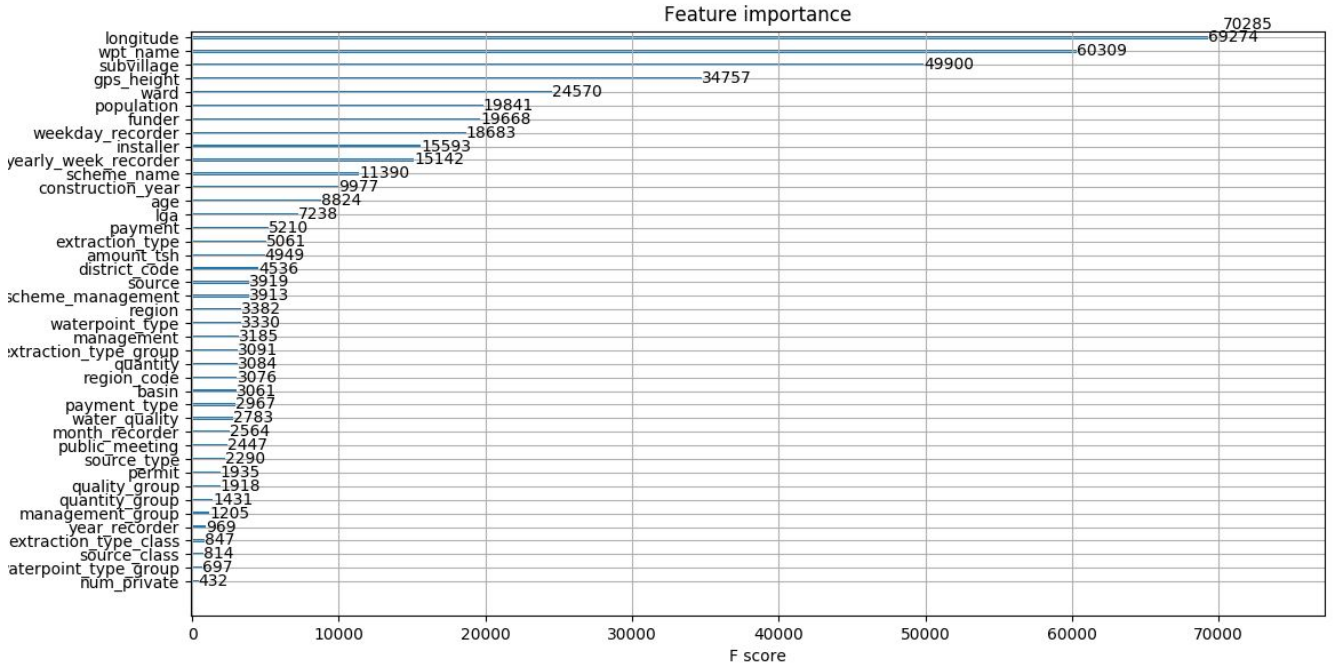
由於使用的是boosting的演算法，所以首先我們利用sklearn的cross_validation跑了四次xgboost後分別記下最好的iteration，平均得到best_boost_round=40，最後拿完整Data訓練時拿best_boost_round的1.2倍下去做training後得到我們最終的model。

最後拿最終model對test data做prediction，得到的結果丟上DrivenData後在leaderboard上的分數為0.8232。

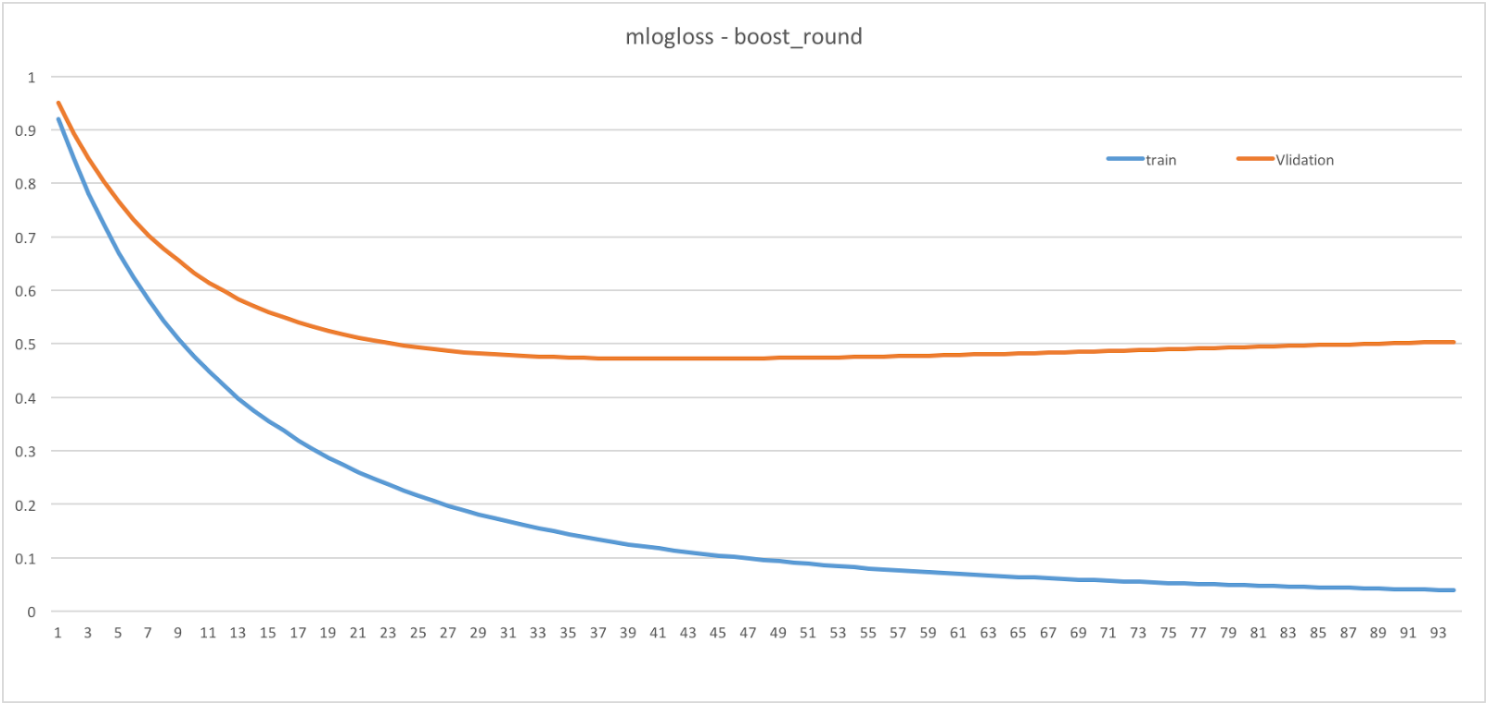
- Experiments and Discussion

  一開始我們設定TreeDepth=6後，在到best_boost_round大約落在600附近，而用這組參數預測出來的結果在DrivenData上只有拿到0.8093的分數。

  後來我們嘗試了加深MaxTreeDepth後，找出的到best_boost_round逐漸下降，最後大概落在40左右，而且DrivenData上的分數也隨著加深MaxTreeDepth提升。最後MaxTreeDepth=64時拿到的分數最高，分數為0.8230。

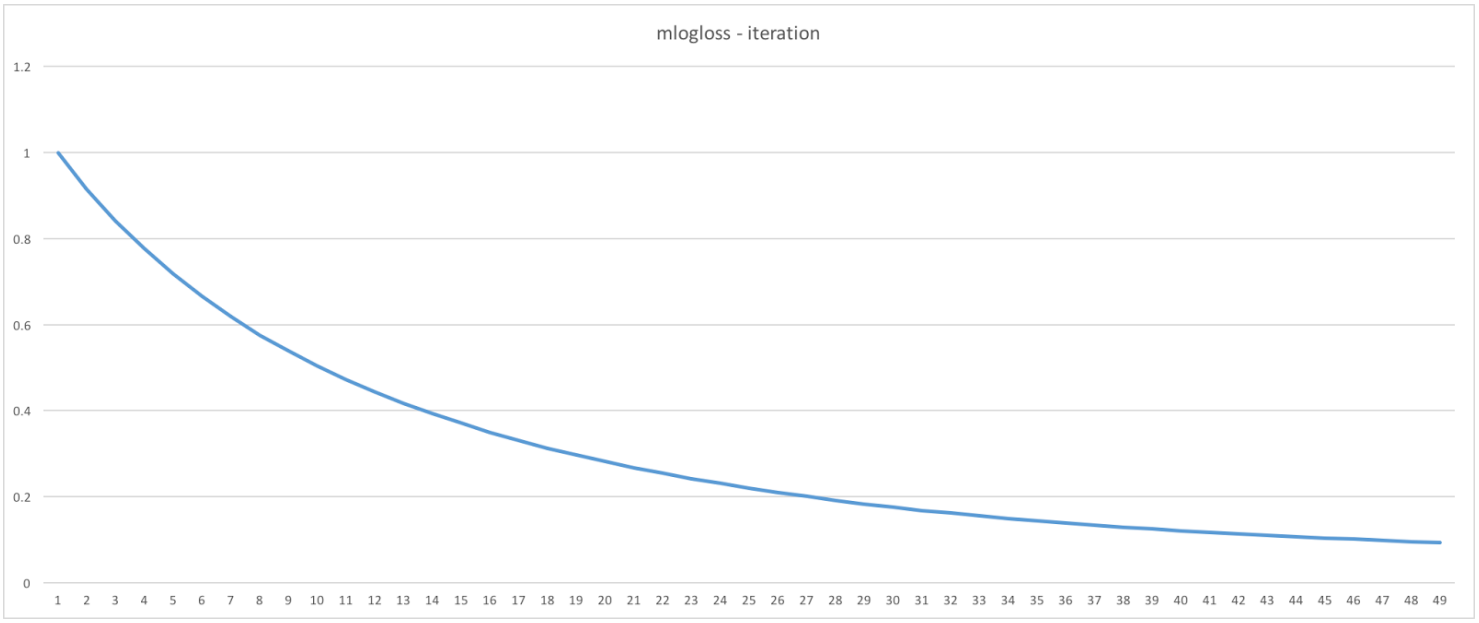


上圖是最終model分析後得到的feature importance，由feature importance可以看到位置相關的資訊(經度、名字)對於預測的結果影響很大，也就是說位於同一個地區的pump缺陷程度很有可能是差不多的。



上圖是其中一個corss_validation時不同iteration對應的mlogloss，藍色為training set而橘色為validation set，可以

看到在大約iteration=40時validation上有最低的mlogloss。

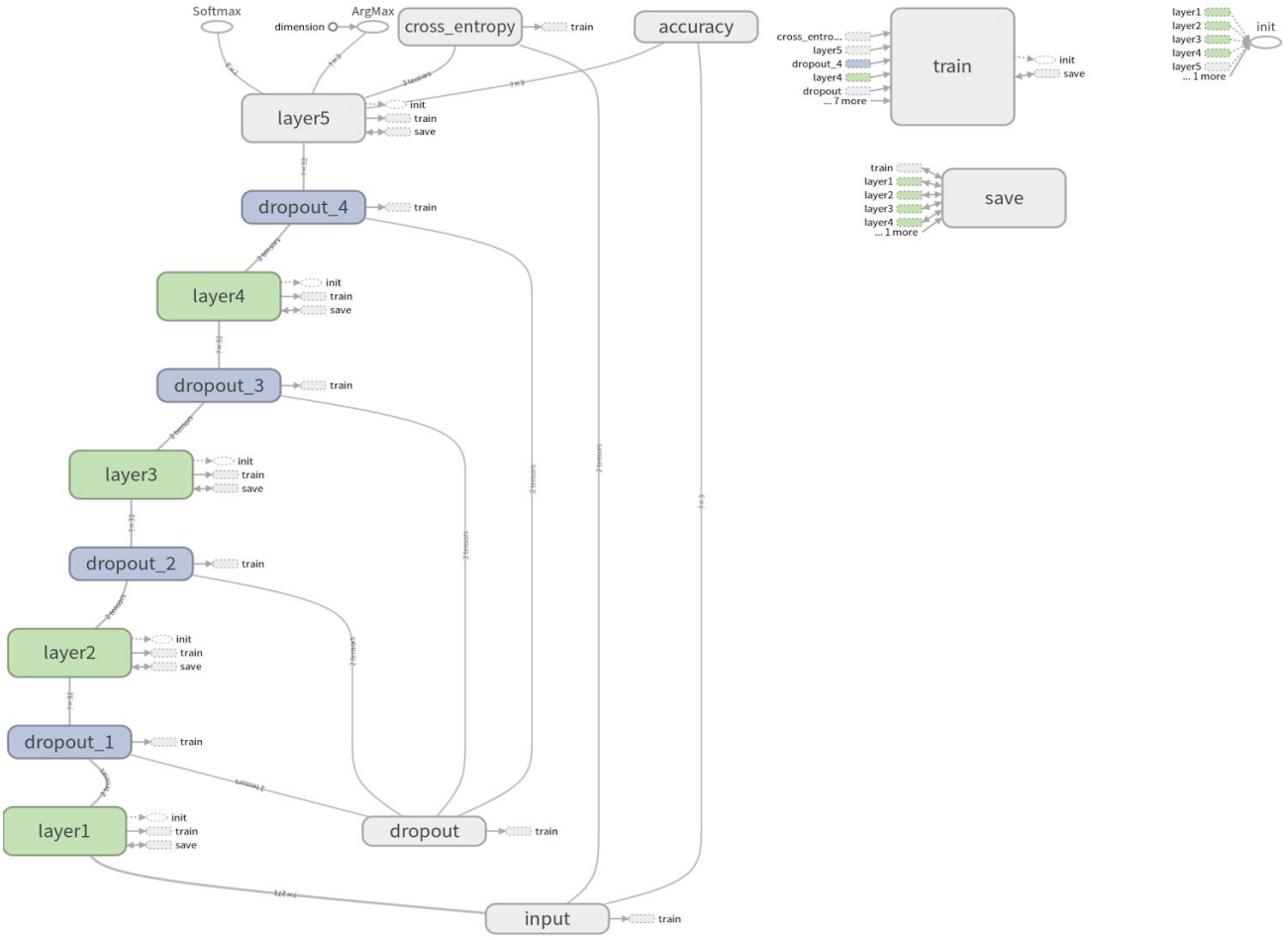


上圖是最後拿完整data做training，best_boost_round=49，
訓練過程對應對應的training set mlogloss。

	subsample = 0.75	subsample = 0.82
max_depth = 6	null	0.8093
max_depth = 32	0.8228	null
max_depth = 64	0.8226	0.8232
max_depth = 128	0.822	0.8232

上圖為嘗試不同的subsample和max_tree_depth後在排行榜
上得到的分數，在subsample=0.82, max_depth=64或128時
拿到的分數最高，0.8232分。

- DNN
  - Preprocessing/Feature Engineering:
    上述兩種model 的preprocessing 都有使用過
    對Gradient Boosting Tree preprocessing 額外做的處理是將 id 從feature 中移除
    對兩者都有額外做的處理是若某一feature 的值在任一data 中有大於1 或是小於0 的話, 即將此feature normalize, normalize 的方法為減掉mean 後除以stadard deviation
  - Model Description

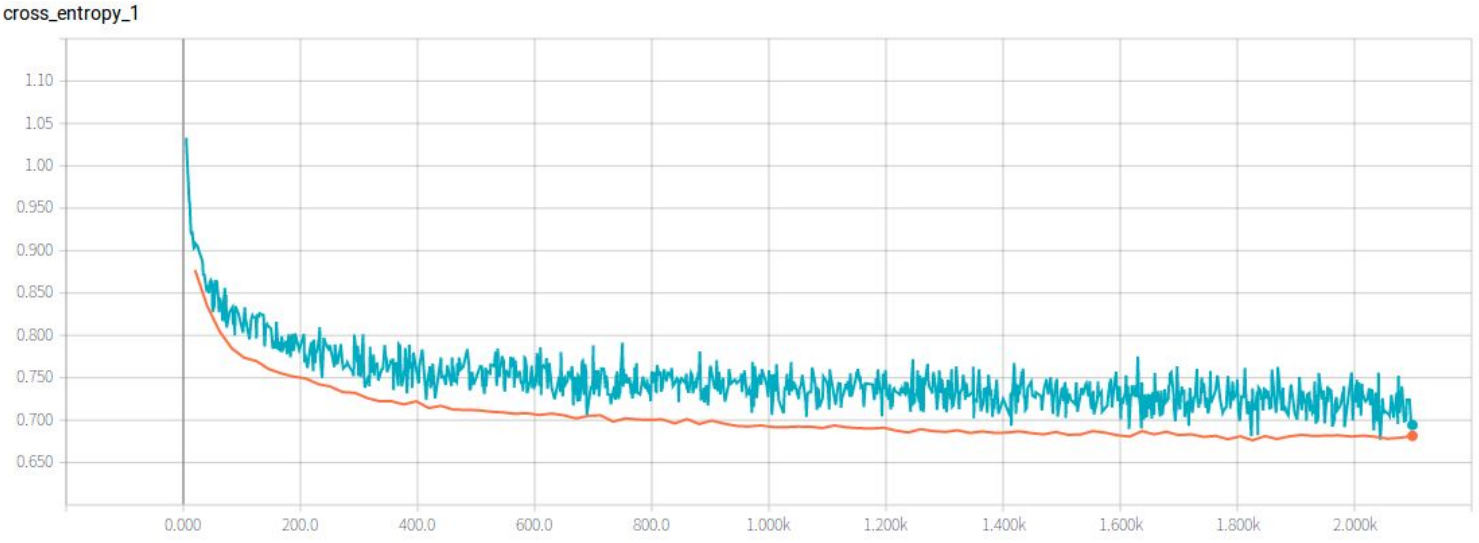


將此問題視為classification problem
使用五層的DNN, 每層的neuron 數為32
activation function 為relu
除了最後一層以外, 在每層皆加入drop out, drop out rate = 0.5
loss function 使用cross entropy
使用Adam mini-batch gradient descent 去optimize model

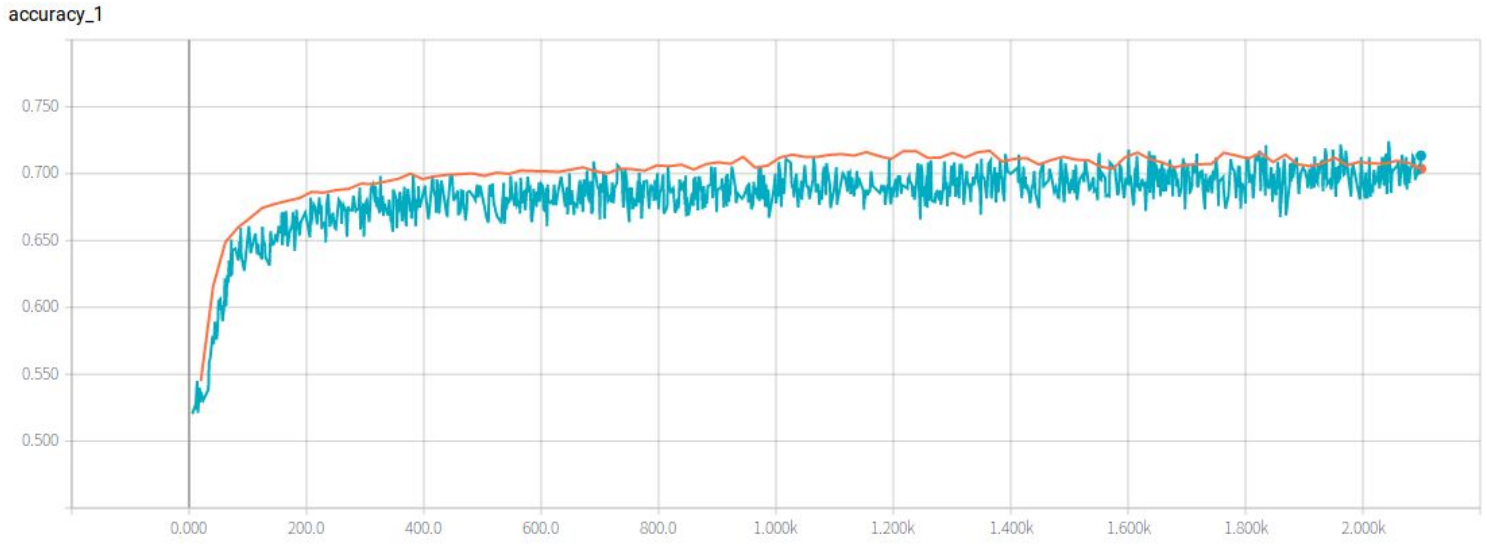
■ Experiments and Discussion

以下菊色線代表validation set, 藍色線代表training set

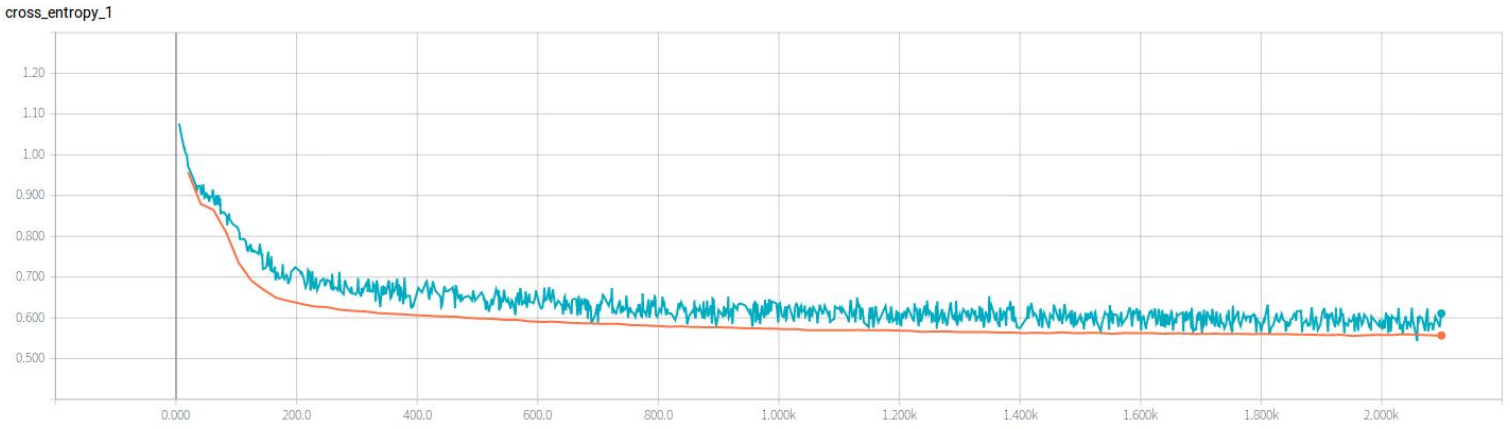
使用Gradient Boosting Tree preprocessing 訓練過程中得到的cross entropy 如下



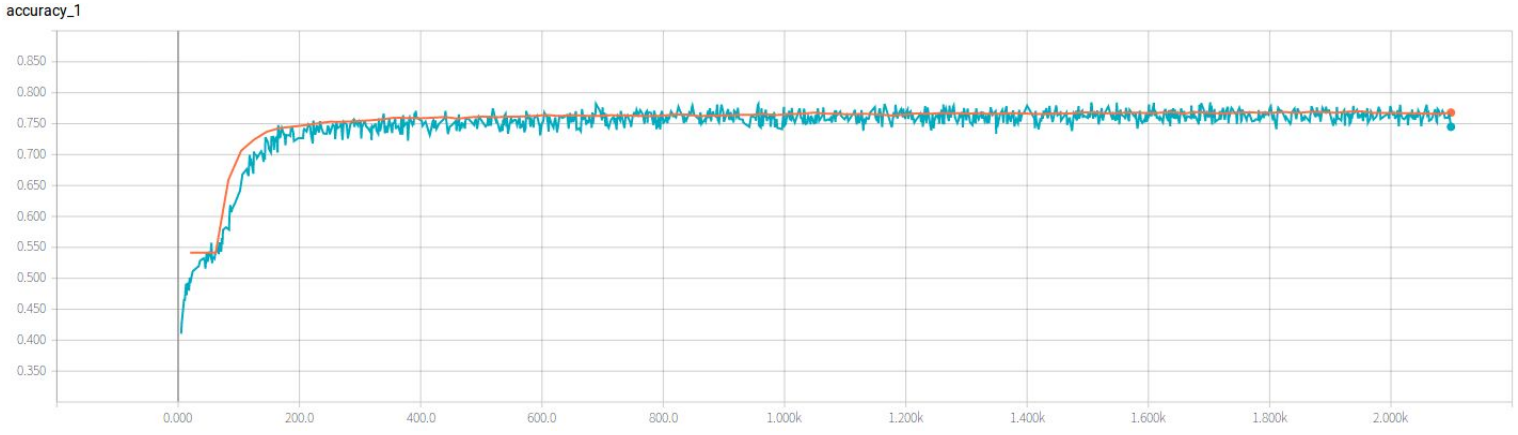
使用Gradient Boosting Tree preprocessing 訓練過程中得到的accuracy 如下



使用Random Forest preprocessing 訓練過程中得到的cross entropy 如下

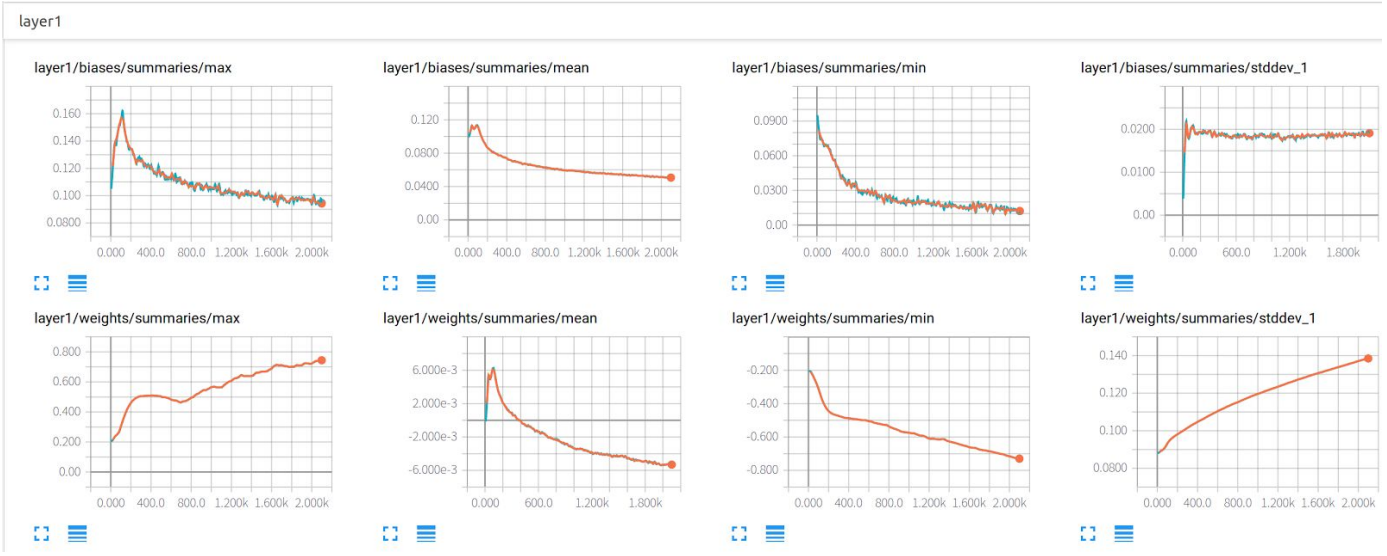


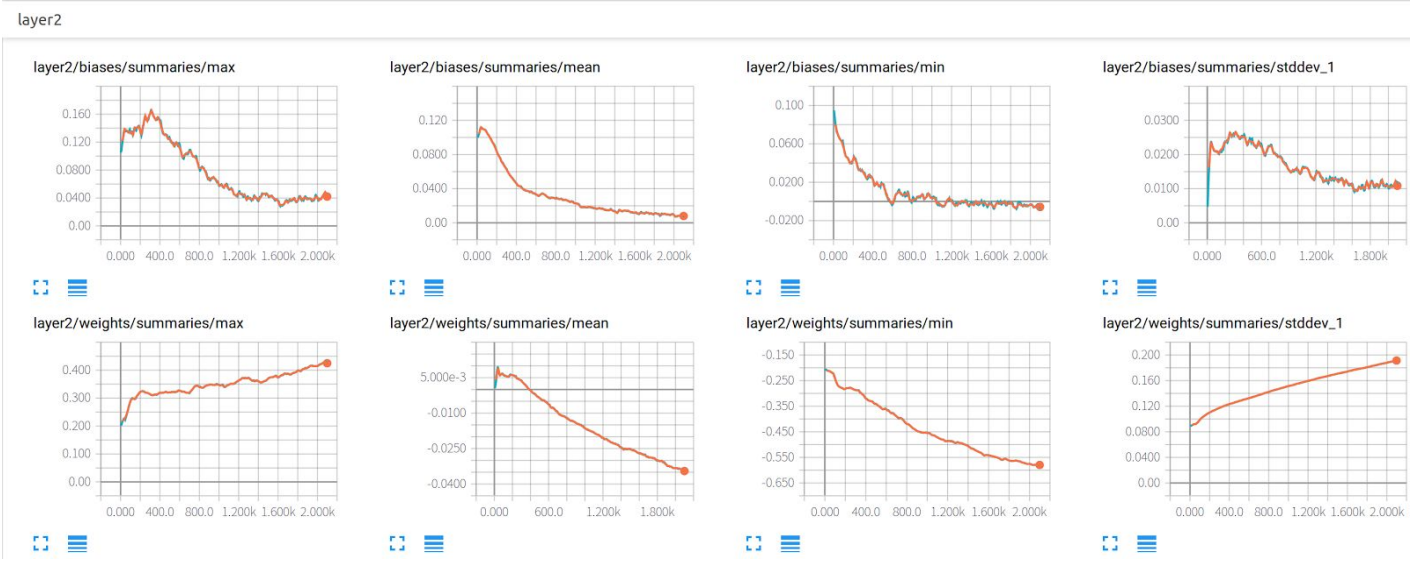
使用Random Fores tpreprocessing 訓練過程中得到的 accuracy 如下



在使用兩種不同的preprocessing 方法後, 發現同Random Forest 的preprocessing 使用在DNN 上較好, 推測因為另一種方法是直接enumerate 某一feature, 也就是說若此feature 若有4 種可能, 則其值可能會0 ~ 3, 如此一來則會造成feature 之間是彼此有較"相近"的關係

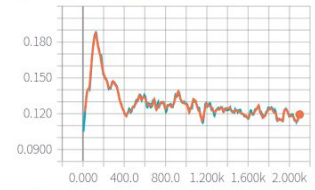
使用Random Forest preprocessing 訓練過程中每層layer 中的weight 及bias 變化如下



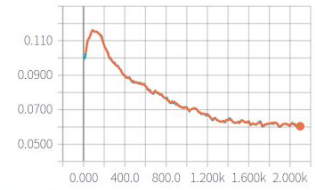


layer3

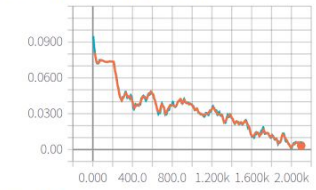
layer3/biases/summaries/max



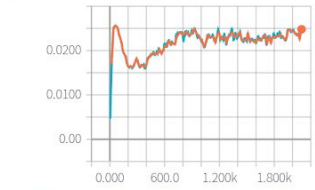
layer3/biases/summaries/mean



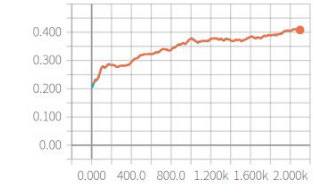
layer3/biases/summaries/min



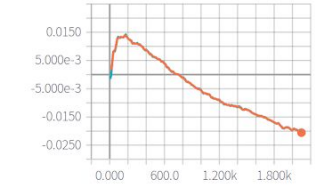
layer3/biases/summaries/stddev_1



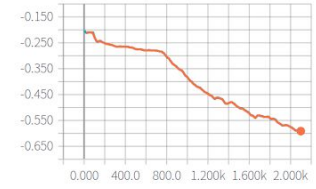
layer3/weights/summaries/max



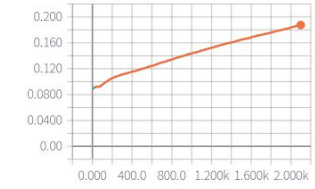
layer3/weights/summaries/mean



layer3/weights/summaries/min

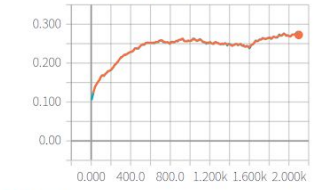


layer3/weights/summaries/stddev_1

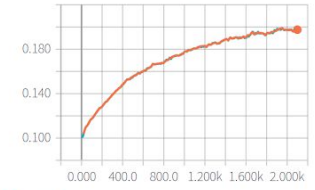


layer4

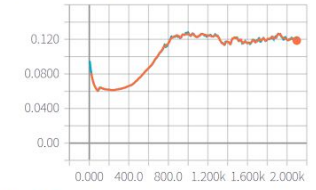
layer4/biases/summaries/max



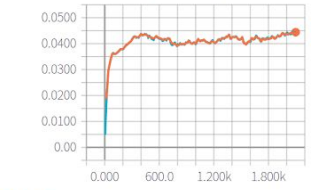
layer4/biases/summaries/mean



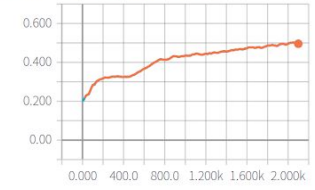
layer4/biases/summaries/min



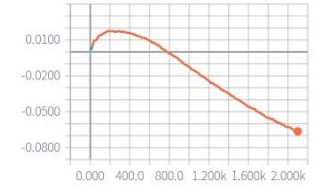
layer4/biases/summaries/stddev_1



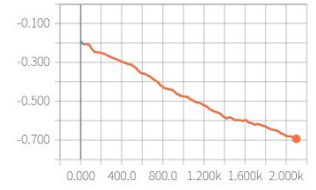
layer4/weights/summaries/max



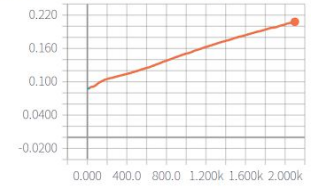
layer4/weights/summaries/mean



layer4/weights/summaries/min

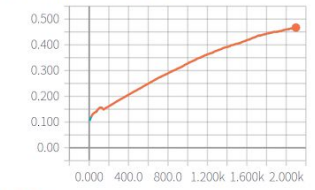


layer4/weights/summaries/stddev_1

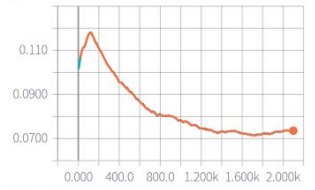


layer5

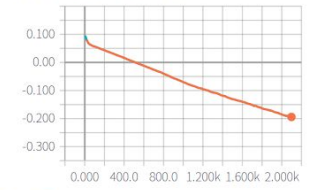
layer5/biases/summaries/max



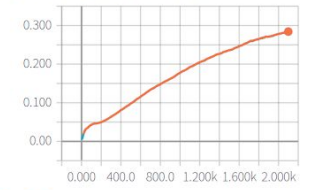
layer5/biases/summaries/mean



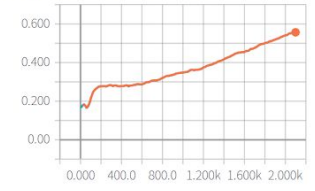
layer5/biases/summaries/min



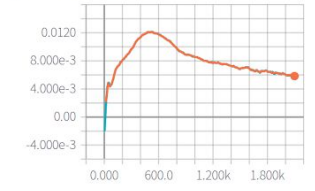
layer5/biases/summaries/stddev_1



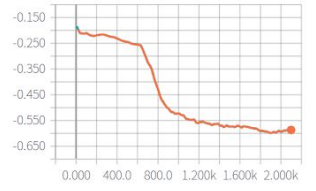
layer5/weights/summaries/max



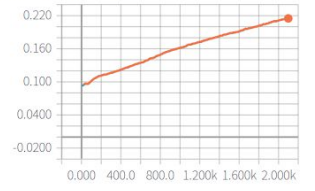
layer5/weights/summaries/mean



layer5/weights/summaries/min



layer5/weights/summaries/stddev_1



使用Random Forest 的preprocessing 在drivendata 上得到最好的score 為0.76 左右, 而另一種只有0.71 左右
調整了幾次不同的batch size, 當batch size 偏小的時候會得到較差的model, 最後將batch size 調整為training data size 的二十分之一
加入了early stop, 使用在validation set 有最低的loss 的model, 之後再train on validation set
即便使用了adaptive learning rate, 在使用較大的learning rate 的時候model 的loss 下降的比較快, 但最終的loss 卻也沒辦法降的比小learning rate 來的低, 在這也花了不少的時間調整learning rate