目錄

一　簡介

1　題目：抽水機狀況預測 (Pump it Up: Data Mining the Water Table)

2　隊名：NTU_b03901030_ML

3　成員：

b03901022 卓伯鴻

b03901026 許凱傑

b03901030 蕭晨豪

b03901039 童寬

4　分工：

b03901022 卓伯鴻	Sberbank(simple baseline), Report
b03901026 許凱傑	Pump it Up(preprocessing), Confusion Matrix, Report
b03901030 蕭晨豪	DengAI(simple baseline), Feature Heatmap, Feature Importance
b03901039 童寬	Pump it Up(baselines), Graphic Training Label Distribution

二　資料前處理

在觀察我們得到的 feature 後，我們發現某些 feature 有相當的重複性(e.g. quantity & quantity_group)，或是明顯的不重要(e.g. recorded by)，在許多比較以及討論後，我們將把 feature 分成主要的兩大類：continuous 和 discrete，剩餘的為 useless，在訓練模型的過程中將不被使用。

1　刪除沒用的 feature

在讀 data 的第一行時會先建一個 dictionary，把每個 feature 對應到第幾欄建立好，再把 useless_features (下圖)刪掉。

之後會把每一筆資料都存成一個 dictionary，用 feature 名字當作 key。

```
useless_features = ['id', 'recorded_by', 'extraction_type_group',
                    'quantity_group', 'source_type',
                    'waterpoint_type_group', 'wpt_name', 'subvillage']
```

2　修正 continuous feature

首先把 continuous feature 中遺失或不合理的資料設為平均值。

再把 date_recorded 轉換成離現在多少天，還有把 construction_year 轉換成離現在多少年，最後將所有 continuous feature 各自做 normalization。

```
continuous_features = ['amount_tsh', 'date_recorded', 'gps_height', 'longitude',
                       'latitude', 'population', 'construction_year']
```

3　處理 discrete feature

我們使用 sklearn 的 DictVectorizer 中的 fit_transform 把它轉換成 one-hot encoding，再視情況轉換成用一個數字代替 one-hot encoding。

```
discrete_features = ['basin', 'lga', 'public_meeting', 'permit', 'funder',
                     'extraction_type_class', 'management', 'management_group',
                     'quantity', 'source', 'source_class', 'waterpoint_type', 'payment']
```

4　Training Set Label

我們透過一個 dictionary(下圖)，把 training data 的 label 轉換成一個數字，如果之後使用的是 DNN model 來做 training，再把 label 做 one-hot encoding。

```
label_dict = {'functional': 0, 'non functional': 1,
              'functional needs repair': 2}
```

三　模型介紹

1　Random Forest

(1) 原理

random forest 是一個包含多個決策樹的分類器，並且其輸出的類別是由個別樹輸出的類別的眾數而決定。這個方法是根據以下的演算法而建造每棵樹：

i　用 N 來表示樣本(sample)的個數，M 表示特徵(feature)數目。

ii　輸入特徵(feature)數目 m，用於確定決策樹(decision tree)上一個節點的決策結果；其中 m 應遠小於 M(m << M)。

iii　從 N 個樣本(sample)中以有放回抽樣的方式，取樣 N 次，形成一個訓練集（　即 bootstrap sampling），並用未抽到的樣本(sample)作預測，評估其誤差。

iv　對於每一個節點，隨機選擇 m 個特徵(feature)，決策樹(decision tree)上每個節點的決定都是基於這些 feature 確定的。根據這 m 個 feature，計算其最佳的分裂方式。

v　每棵樹都會完整成長而不會剪枝(Pruning)，這有可能在建完一棵正常樹狀分類器後會被採用。

(2) 應用

```
clf = RandomForestClassifier(min_samples_split=8,
                             n_estimators=1000,
                             oob_score=True,
                             n_jobs=-1)
```

而使用這樣的方法具有一些優點：首先，它可以在決定類別時，評估變數的重要性，對於擁有許多特徵(feature)的問題，能加權重要的特徵是相當重要的事。再者，在建造 random forest 時，它可以在內部對於一般化(normalize)後的誤差產生不偏差的估計。最後，它包含一個好方法可以估計遺失的資料，並且，如果有很大一部分的資料遺失，仍可以維持準確度，這便可有效解決測資有許多缺漏的問題。

在建造這一個模型方面，我們使用 sklearn 中 ensemble 的 RandomForestClassifier 來產生 random forest 的 model，並且產生 1000 棵 decision tree，每個 node 會在 8 個 sample 後才會 split。

2　Extreme Gradient Boosting (XGBoost)

(1) 原理

XGBoost 是一種「前向分佈加法模型」(forward stepwise additive modeling)，而這種模型其實就是一種貪婪演算法(greedy algorithm)，而 XGBoost 使用的目標函數是利用泰勒展開式(Tyler's

expansion)展開三項當作一個近似值使用，他的目標函數(objective function)如下：

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), f_k \in F$$

而他一般化(normalize)之後的損失函數(loss function)如下：

$$L(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$where \ \Omega(f) = \gamma T + \frac{1}{2}\lambda |w|^2$$

其中，w 是代表每個決策樹(decision tree)下的每個葉子(leaf)的權重(weight)，而又代表他決策樹(decision tree)的葉子個數(leaf numbers)。而 Ω(f)便是我們的結構分數(structure score)，它代表了當我們指定一個決策樹(decistion tree)的結構時，我們在目標函數上最多減少多少。

在分裂節點的演算法部分，XGBoost 開發者論文中提出，使用貪婪演算法(greedy algorithm)，在每一次判斷是否加入一個新的分割節點時，設定一個目標函數來計算分割前後的複雜度變異，因此，只要判斷「左葉(left leaf)複雜度分數」加上「右葉(right leaf)複雜度分數」減去「不分割前可拿到的複雜度分數」並扣掉「加入新節點(node)而產生的複雜度代價」的增加或是減少(亦即正值或負值)，便可判斷加入這個新的節點是否對整個決策樹(decision tree)有助益。

下圖即為上述演算法之 pseudo code。

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ *to* m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

(2) 應用

```
xgb_params_1 = {
    'objective': 'multi:softmax',
    'booster': 'gbtree',
    'eval_metric': 'merror',
    'num_class': 3,
    'eta': .2,
    'max_depth': 14,
    'colsample_bytree': .4,
}
```

透過 XGBoost 這個原本用 c 語言寫成的套件實作 extreme gradient boosting。此次的 num_class 為 3，因為是 single-label categorization，使用 softmax 當作 output 的 activation，以及把 eval_metric 設為 merror(i.e.　#(wrong cases)/#(all cases)).把 learning rate 設為 0.2，最大深度設為 14，然後每次建一棵樹採用的 feature 比例是 0.4。

```
cv_model = xgb.cv(dict(xgb_params), dtrain, num_boost_round=500, early_stopping_rounds=10, nfold=4, seed=i)
min_idx = np.argmin(cv_model['test-merror-mean'].values) + 1
model = xgb.train(dict(xgb_params_1), dtrain, num_boost_round=min_idx)
model.save_model("xgb.model_{:d}".format(i))
```

在 train 之前先透過 cross-validation 觀察在哪一個 round 會有最小的 merror，並把它設為真正在 train 的時候所需要的 round 數。

3　DNN

```
model = Sequential()
model.add(Dense(256, activation='relu', input_dim=x_train.shape[1]))
model.add(Dropout(DROPOUT_RATE))
model.add(BatchNormalization())
model.add(Dense(256, activation='relu'))
model.add(Dropout(DROPOUT_RATE))
model.add(BatchNormalization())
model.add(Dense(512, activation='relu'))
model.add(Dropout(DROPOUT_RATE))
model.add(BatchNormalization())
model.add(Dense(512, activation='relu'))
model.add(Dropout(DROPOUT_RATE))
model.add(BatchNormalization())
model.add(Dense(y_train.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

疊了兩層 256 個節點(node)且使用 relu 當作 activation 的 dense layer，再疊兩層 512 個節點(node)且使用 relu 當作 activation 的 dense layer，在這些 fully connected layer 之後都有接上 BatchNormalization Layer 並分別在後面加上一層 dropout，optimizer 使用的是 adam。因為是 single-label categorization，最後使用 softmax 當作 output 的 activation，loss 使用的是 categorical_crossentropy。

由於 DNN 是一個非常容易 overfitting 的模型，所以我們使用的 dropout 比例都相當的高(值為 0.5)，以免在訓練的過程中，過度激發常常被使用的神經節點(node)，來降低 overfitting 發生的機會。

4　Ensemble (XGBoost by voting)

最後，我們挑選了表現較好的 xgboost 方法，再進行選取合理的參數之後，獲得 12 個 xgboost 的 model 分別使用同樣的訓練資料(training data)來建構模型，並在測試資料(testing data)的時候再利用 voting 決定最後的分類。

四　實驗與討論

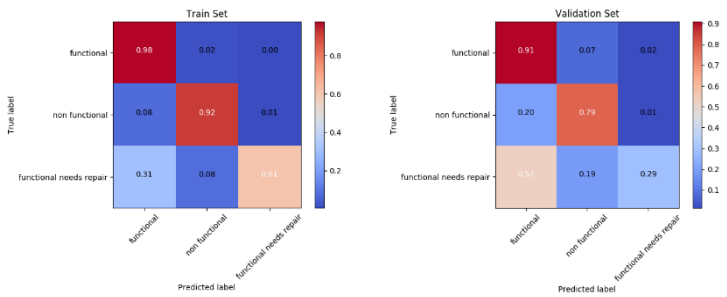
1　Graphic Training Label Distribution

2　Training and Validation Performance

	Training Accuracy	Validation Accuracy
Random Forest	0.928750	0.817003
XGBoost	0.993790	0.809764
DNN	0.635329	0.635048
Ensemble	0.994332	0.808418

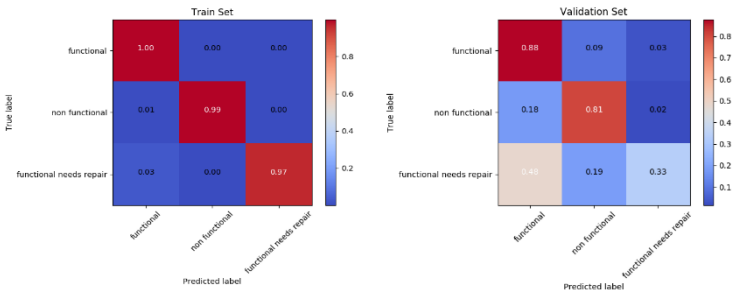
3　Confusion matrix

(1) Random Forest



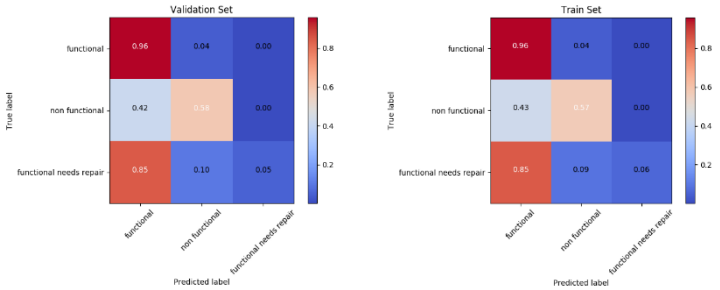
functional needs repair 在 training set 的 accuracy 只有 0.61，代表此模型無法正確判斷這個 label。因此，在 validation set 的結果是更差的 0.29，也是影響此模型好壞很大的一個因素。

(2) XGBoost



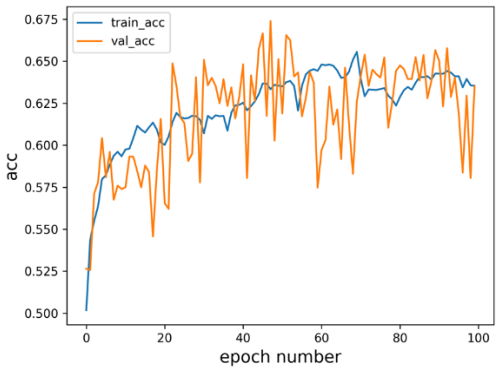
training set 上三個 label 的表現都很好，但 validation set 上 functional needs repair 的表現(0.33)還是很差，出現嚴重 overfitting。

(3) DNN



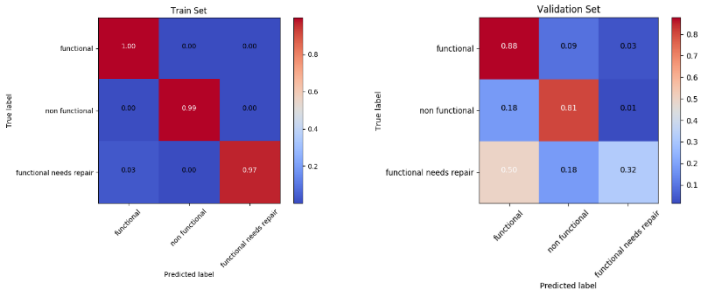
只有 functional 這個 label 在 training set 上能被正確預測，另外兩個 label 的表現都很差。

因為 DNN 表現不好，所以我們把每個 epoch 的訓練情形做了曲線圖分析如下：



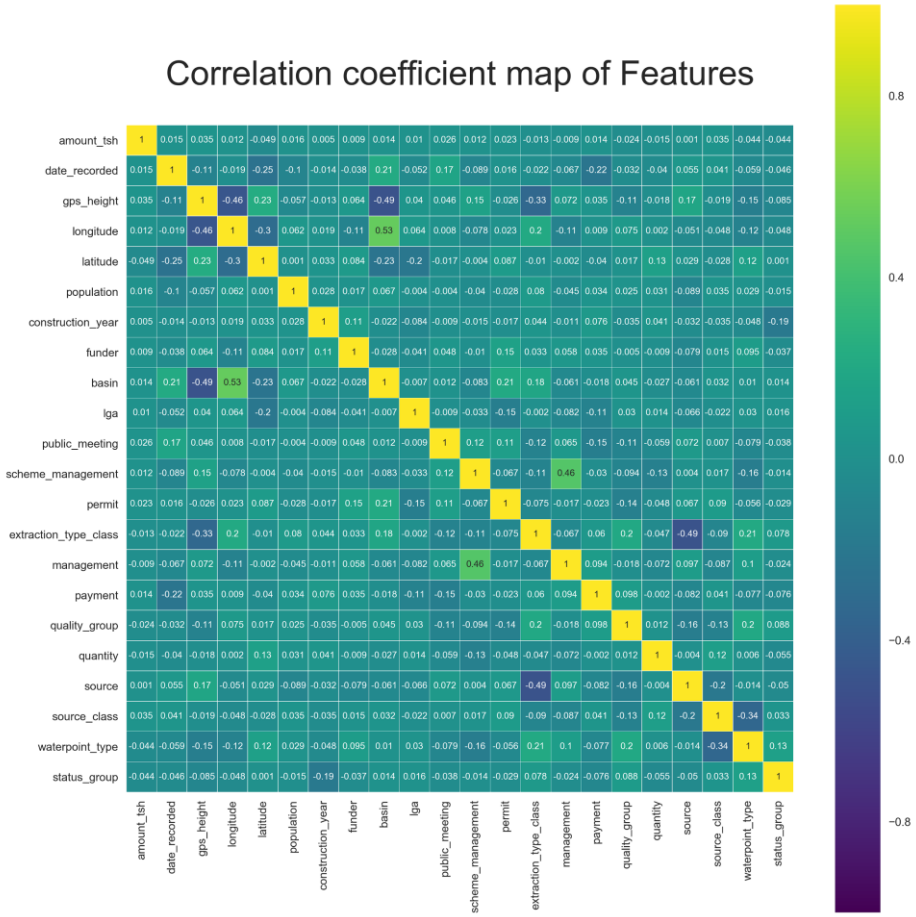
可以看出來 training accuracy 有緩慢上升，不過到大約 0.65 就飽和了，但是在 validation set 的表現就起伏很大，所以我們覺得 DNN 可能一下就 overfitting。

(4) Ensemble XGBoost by voting



ensemble 的結果跟單一 xgboost 的結果差不多，一樣在 functional needs repair 這個 label 上出現 overfitting。

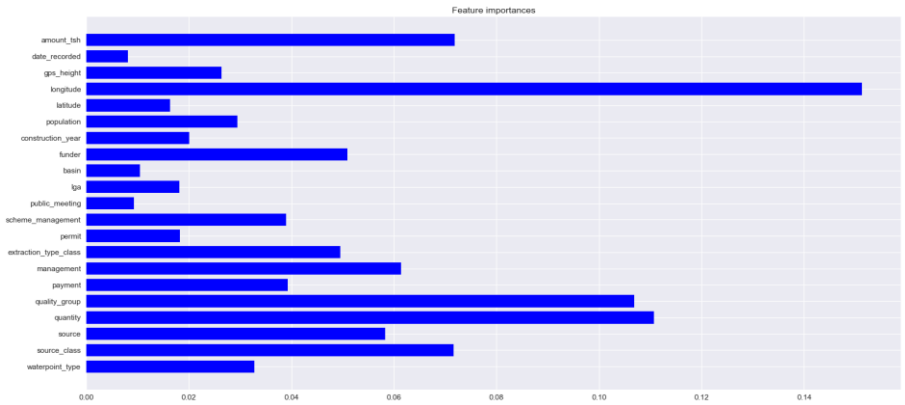
4 Feature Heatmap



可以看到我們的預測目標 status group 和大部分的 feature 相關係數都沒有特別高，即使是最相關的 construction year 也只有-0.19，另外各個 feature 間的相關係數也不高，這是一個好的現象因為這代表我們所選取的這些 feature 可以各自代表資料在某個層面的特性並且不互相重複。

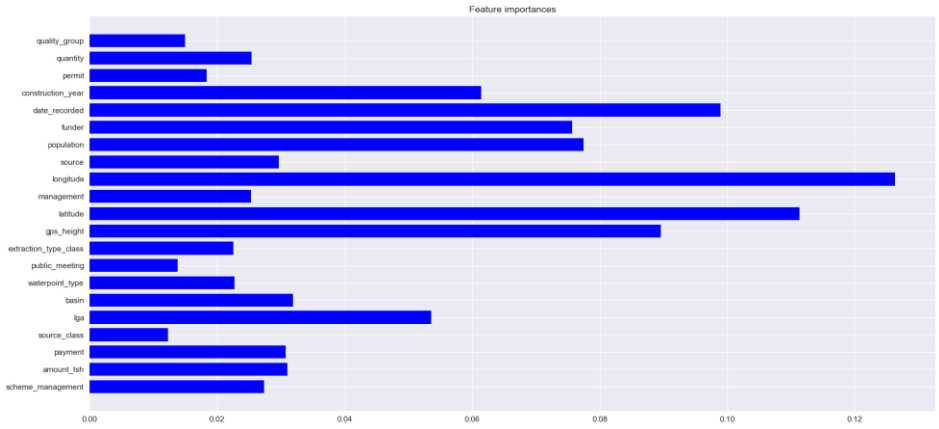
5 Feature Importance

(1) Random Forest



我們的 random forest model 使用的是 sklearn 裡的 RandomForestClassifier，使用其內建的 feature_importance_作圖，可以得到以上結果。從圖中可以看出 longitude, guality_group, quantity 和 amout_tsh 為主要決定樹的分支的幾個重要 feature。

(2) XGBoost



XGBoost model 的 feature importance 如上圖，可以看到 longitude 依舊為最重要的 feature，接下來 latitude 和 date_recorded 以及 gps_hight 也較為重要。

我們認為在兩個 feature importance 的圖中 longitude 皆很高，代表地理位置對於水井是否為 functional 來說是一重要因素，這是合理的，因為同一區域相近的水井抽取的是同一地下水層的水，因此相關性變會比較高。另外兩個 model 除了 longitude 以外注重的 feature 不太相同。因此將這兩個相關性不太高的 model 做 stacking/voting 類的 ensamble 方法可能有助於改進我們的 accuracy。