- 題目: 抽水機狀況預測
- Team name: NTU_r05921077_ML666
- Members: 陳立杰 r05921077, 陳彥谷 r0492110, 賈才 r05942105, 陳俞兆 b02901044
- Work division
  陳立杰: Preprocessing/Feature Engineering 和 Random Forest
  陳彥谷: xgboost
  賈才: xgboost
  陳俞兆:

- Preprocessing
  我們 preprocessing 總共可以分三個部分:
  (1) PreProcess_train_value_and_test.py: 處理 train_value.csv 和 test.csv
  因為 random forest 和 xgboost 在做的時候可以給每個資料一個數字來代表他，所以在處理非數字的欄位的時候，可以直接給同一個欄位每個獨特的字串一個數字來代表，這樣就不用把一個欄位用 one-of-n encoding 展開成多個欄位。
  我們的作法是，先利用 pandas 的 dataframe 找到 train_value.csv 跟 test.csv 每個欄位的所有獨特字串，把他們連接起來，之後再用 set 來過濾掉可能重複的字串，這個部分特別注意必須用 python2.7 版才能夠保持他的 order 才能夠 reproduce，假設一個欄位有 n 個獨特的字串，就從 0 到 n-1 對映每個字串到一個整數，然後就可以把 test_value.csv 和 test.csv 的這個欄位的所有字串根據對映的數字轉換成整數了，其中有些資料可能是空格，可以直接把每個欄位的空格用該欄位的 median 填上去，之後可以輸出 ntrain_value.csv 跟 ntest.csv 了。
  (2) PreProcess_train_label.py: 處理 train_label.csv
  因為 train_label 的 status_group 這個欄位也是字串，所以也是用相同的方式找到所有獨特的字串，即 3 個 functional needs repair、non functional 和 functional，然後把所有字串轉成整數 0,1 和 2 後，就可以輸出為 ntrain_label.csv 了。
  (3) Combime_train_value_and_label.py: 將 train_label.csv 跟 train_value.csv 合併
  根據 id 這個欄位把 training data 跟 label 做合併並排序之後，再把空的資料填上該欄位的 median，就可以輸出 training 時唯一會用到的檔案 train_data.csv 了。

- Feature Engineering
  在所有 feature 中，一定得刪掉的 feature 只有一個為 id，因為這個

feature 不代表任何意義，另外還有一個 feature 比較特別的是 date_recorded，因為這個 feature 雖然說他是字串，但是實際上他是數字而且這些數字可能有些意義，所以我們並沒有直接在 preprocessing 的時候直接把這個欄位的獨特字串對映到一個數字，而是用另一種方式將字串 encoding 成數字。

我們的做法是: 因為在所有資料中，date_recorded 最早的年份為 2010，所以所有年份只有最後一個數字會不一樣，然後月份就取兩個數字，如果是個位數的話就在前面補 0，同理日也是一樣。例如 2013/3/17 就會被 encoding 成 30317，這樣做後準確度就會稍微提升。

其他我們也有根據資料的特性做一些 feature engineering，但是效果都不是很好:

1. installer: 獨特的數值太多，刪掉。
2. wpt_name: 獨特的數值太多，刪掉。
3. construction_year 中很多筆資料他不是 null 而是放 0 來代表未知，所以用這個欄位的 median 來取代 0。
4. contruction_year: 最小為 1960，將全部 data 都減掉 1960，1960=1、1961=1，以此類推。
5. gps_height 中也有很多值是 0，試著將他們用該欄位的 median 取代
6. recorded_by: 只有一個值，刪掉。
7. wpt_name: 太多獨特的值了，刪掉。
8. water_quality : 和 quality_group 太像了，刪掉。
9. installer: 太多獨特的數值，刪掉。
10. num_private: 太多空格了，刪掉。
11. scheme_name: 太多空格了，刪掉。

另外就是將一些特性比較接近的資料只用一個來代表，例如:

1. region 跟 region_code 非常接近
2. quantity 跟 quantity_group 非常接近
3. quality_group 跟 quality 非常接近
4. source_type 跟 source 非常接近
5. payment 跟 payment_type 非常接近
6. waterpoint_type_group 跟 waterpoint_type 非常接近
7. extraction_type_group 跟 extraction_type 非常接近

我們隨意取一個來代表，但結果也沒有比全部都加的結果還好。所以最後就保持原本的方法，只刪掉 id 這個 feature，並且將 date_recorded 做 encoding 成數字。

- Model1 - Xtreme Gradient Boosting(XGBoost)

  Boosting 分類器屬於集成學習模型，基本概念是把成千上百個準確度較低的分類器集合起來，成為一個準確度較高的模型。

XGBoost 是 Friedman 所提出的 Gradient Boosting Machine 概念的實現，Gradient Boosting Machine 在生成每一棵樹的時候採用梯度下降的方式，以之前生成的所有樹為基礎，向著最小化目標函數的方向走，因此在數據集較大較複雜的時候，需要耗費大量的時間進行疊代運算，才能達到令人滿意的準確度。

而 XGBoost 有以下幾個優良的特性：

1. 顯示的把樹模型複雜度作為正則項加到優化目標中。
2. 公式推導中用到了二階導數，用了二階泰勒展開。
3. 實現了分裂點尋找近似算法。
4. 利用了特徵的稀疏性。
5. 數據事先排序並且以 block 形式存儲，有利於並行計算。
6. 基於分布式通信框架 rabit，可以運行在 MPI 和 yarn 上。
7. 實現做了面向體系結構的優化，針對 cache 和內存做了性能優化。

項目實測中使用發現，Xgboost 的訓練速度要遠遠快於傳統的 GBDT 實現，10 倍量級。

- Model2 - Random Forest

Random forest 是將 decision tree 用 bagging 的方式來做，decision tree 是種可以高度 fit 一組 training data 到 0% error rate 的方法，但是他的缺點就是非常容易 overfitting，因為只要你的 depth 夠深他就可以針對所有資料給他一個 output，random forest 的方法大概就是它會分成好幾個 decision tree 來作，但給個 decision tree 用 sample 取得的 dataset 都不會一樣，然後 decision tree 每個 node 在作 split 的時候也會對 feature 作不同的限制來避免 overfitting，最後再將所有 decision tree 用 voting 的方式來決定最後每筆 data 的 output 為何，還可以用某些 tree 沒 sample 到的 data 來做 testing，稱作 out of bag validation，下列是比較正式的演算法:

1. 用 N 來表示 dataset 的個數，M 表示 feature 數目。
2. 輸入 feature 數目 m，用於確定決策樹上一個節點的決策結果；其中 m 應遠小於 M。
3. 從 dataset(共 N 筆資料)中以有放回抽樣的方式，sample N 次，形成一個訓練集（即 bootstrap 取樣），並從 dataset 中用抽出未用到的 data 作預測，評估其誤差。
4. 對於每一個節點，隨機選擇 m 個 feature，決策樹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。
5. 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）。

以下是 random forest 的一些特點:

1. 對於很多種資料，它可以產生高準確度的分類器。
2. 它可以處理大量的輸入變數。
3. 它可以在決定類別時，評估變數的重要性。
4. 在建造森林時，它可以在內部對於一般化後的誤差產生不偏差的估計。
5. 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。
6. 它提供一個實驗方法，可以去偵測 variable interactions。
7. 對於不平衡的分類資料集來說，它可以平衡誤差。
8. 它計算各例中的親近度，對於數據挖掘、偵測離群點（outlier）和將資料視覺化非常有用。
9. 使用上述。它可被延伸應用在未標記的資料上，這類資料通常是使用非監督式聚類。也可偵測偏離者和觀看資料。
10. 學習過程是很快速的。

- Experiments and discussion for xgboost

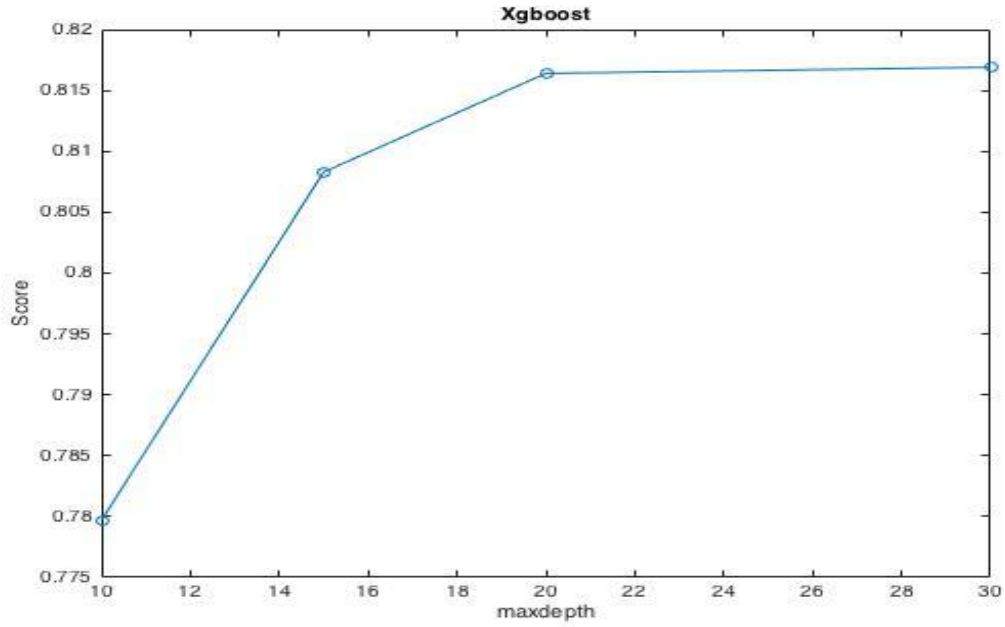
  xgboost 基本方法和默認參數

  1. params ：一個字典，裏面包含著訓練中的參數關鍵字和對應的值，形式是 params = { ‘booster’ :’ gbtree’ ,’ eta’ :0.1}
  2. dtrain 訓練的數據
  3. num_boost_round 這是指提升叠代的個數
  4. evals 這是一個列表，用於對訓練過程中進行評估列表中的元素。形式是 evals = [(dtrain,’ train’),(dval,’ val’)]或者是 evals = [(dtrain,’ train’)],對於第一種情況，它使得我們可以在訓練過程中觀察驗證集的效果。
  5. obj,自定義目的函數
  6. feval,自定義評估函數
  7. maximize ,是否對評估函數進行最大化
  8. early_stopping_rounds,早期停止次數 ，假設為 100，驗證集的誤差叠代到一定程度在 100 次內不能再繼續降低，就停止叠代。這要求 evals 裏至少有 一個元素，如果有多個，按最後一個去執行。返回的是最後的叠代次數（不是最好的）。如果 early_stopping_rounds 存在，則模型會生成三個屬性，bst.best_score,bst.best_iteration,和 bst.best_ntree_limit
  9. evals_result 字典，存儲在 watchlist 中的元素的評估結果。
  10. verbose_eval (可以輸入布爾型或數值型)，也要求 evals 裏至少有 一個元素。如果為 True ,則對 evals 中元素的評估結果會輸出在結果中；如果輸入數字，假設為 5，則每隔 5 個叠代輸出一次。

11. learning_rates 每一次提升的學習率的列表，
12. xgb_model ,在訓練之前用於加載的 xgb model。

■ 調整方法

1. max_depth : 與其他參數關系不大，初始值設置為 10，Score 為 0.7737，模型訓練時間較快。為找到一個最好的誤差值，調整參數進行對比。經測試 15 對應到的 score 為 0.8083，有不少進步，逐步調整至 20 得到 0.8164，30 則為 0.8169，隨著訓練時間增長，邊際效益逐漸變低。



2. subsample:找到最優的 max_depth 之後，可以調整 subsample。初始值設置為 1，然後調整到 0.8 誤差值變高，調整到 0.9 還是變高，就保持為 1.0。
3. min_child_weight ：方法同上，經測試不需調整。
4. colsample_bytree:要依據特征個數來判斷，經調整為 0.4。
5. 經過上面的調整，已經得到了一組參數，這時調整 eta 到 0.05，以得出一個最佳的 num_round

- Experiments and discussion for random forest

在 random forest 測試時，除了 feature engineering 外，主要大概有三個地方的參數可以調整。

第一個部分是 preprocessing 產生出來的資料，這個部份我們總共做了兩次不同的 preprocessing，這邊會造成的差異是當我們將每個獨特的字串給不同的編號會造成不同的結果，這只是 random 排列組合不同所產生的結果，第一個版本 preprocessing 的實測結果最好可以到 0.8236，在其他參數沒

有改變太多的情況下，另一個版本的 preprocessing 實測結果最好可以測到 0.8242，表示這 preprocessing 雖然是亂序排列但時還是會對結果產生一定的影響。

第二個部分是 training set 排列順序，這邊是利用不同的 random seed 讓來測試不同排列順序的結果，也會對結果造成影響。

第三個部分是 sklearn 中 random forest 參數的調整，我們總共用到了七個參數，criterion, min_samples_split, n_estimators, max_features, oob_score, random_state 和 n_jobs，他們的意思是:

- criterion: 用來衡量每次 split 的好壞，這個參數我們固定都選擇 gini 代表用 Gini impurity 來作，第二個參數。
- min_samples_split: 每個 internal node 在 split 最少需要的 smaple 數，這個參數對結果會有些影響，所以沒有固定的值。
- n_estimators: 決定總共要產生幾個 tree。
- max_features: 這個參數是決定每次在 split 時會用到幾個 feature，這邊選擇的是 auto，意思是一次選 sqrt(n_features)個 feature 來作 split。
- oob_score: 是否要用 use out-of-bag samples 來預測準度，這邊都設 true。
- random_state: 這個參數是 sklearn 內建的 random seed，其實也可以用 numpy 的 random seed 來設，意思差不多。
- n_jobs: 這個參數是決定 cpu 的個數，這邊固定選 -1，意思是為全部 cpu 一起跑。

Feature engineering 我們有試過幾種組合，第一種組合是將 wpt_name、subvillage、funder 和 installer 刪掉，測出來的結果是 0.8167，原本平均是 0.8185 左右，結果並沒有比較好，

第二種是把下列 id、date_recorded、status_group、wpt_name、subvillage、funder、installer、scheme_name 和 ward 刪掉，實際測出來的結果為 0.8073，在沒做這些 feature engineering 之前平均也是大概可以跑到 0.8185 左右，準確度差距變的更大了。

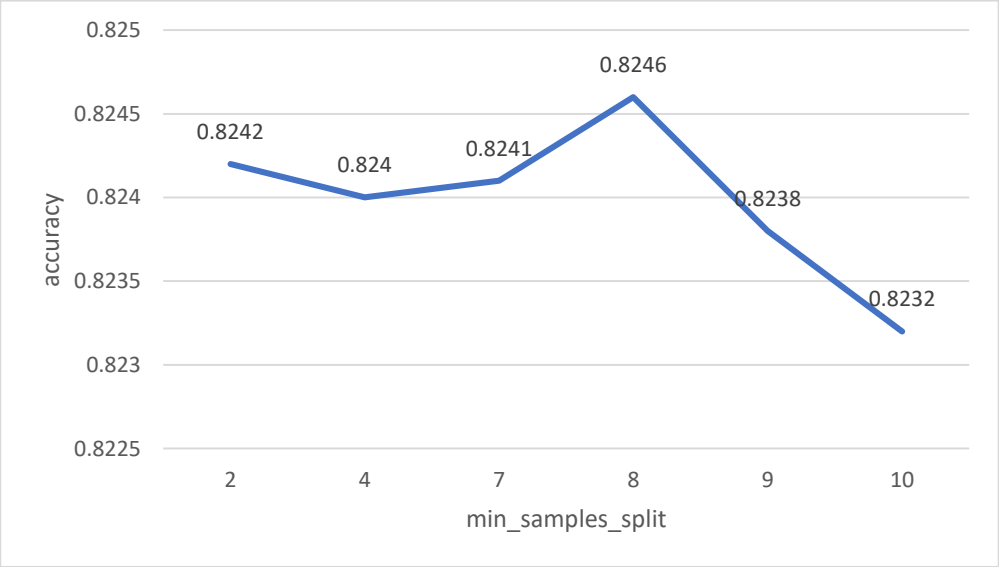
第三種組合是將 num_private、recorded_by、wpt_name、extraction_type_group、extraction_type、payment_type、water_quality、scheme_management、district_code、region、region_code、subvillage、ward、waterpoint_type_group、quantity_group 和 installer 這些 feature 刪掉，然後再做 contruction_year 和 gps_height 的 feature engineering，實際測出來的結果是 0.8061，跟原本的 0.8185 也差蠻多的，所以這個 feature engineering 也沒有比較好。

Training set 順序的實驗，當其他參數都相同時，第一種 training set 的排序方法測出來的結果是 0.8230，用另一種 training set 的排序方法測出來的結果為 0.8238，代表不同的 training set 的排序方法確實可以對最後的結果造

成一些影響，最後是在 numpy seed=580 時用 numpy 的 permutation 做排序可以得到最佳結果。

最一開始的時候，在 random forest 這邊我們並沒有試很多參數，只有調整 random forest 的 tree 之數目，我們發現其實 tree 的數目 n_estimators 在 200 其實就蠻夠用了，增加後可能有機會跑出比較好的結果，但平均來說其實差距不大，跑的時間反而會變更久。當 n_estimators=200 時測出來的結果分別為 0.8210、0.8188 和 0.8191，當 n_estimators=500 時測出的結果分別為 0.8202 和 0.8195，當 n_estimators=1000 時測出來的結果為 0.8198，可以看到這些數據準確度其實都差不多，最後我們的最佳解的 n_estimator 是設 1000 時得到的。

之後試了 random_state 這個參數，就像前面所說的，他就是一般亂數的種子，最後嘗試的結果是設 random_state=1 剛好可以測到最好的結果。然後我們又試了另一個參數 min_samples_split 發現他也蠻重要的，原本我們都沒調整這個參數，而他的 default 值是 2，這時我們一直卡在 0.822 左右但過不了 strong baseline 0.8228，把他稍微條大後會有明顯的成長，之後就能過 strong baseline 了，min_samples_split 大小對準確度的影響如下圖所示:



這邊可以看到 min_samples_split 在 8 的時候會緩緩到達最高點，之後準確度就會急速下降，其中 0.8246 就是我們的最佳分數，相較於 xgboost 準確度提高不少。

除此之外我們也有用 self training 來作 random forest，我們的做法是每當預測的機率大於 0.65 時就把該 testing data 再放進去 training data 裡在，重複跑 N 次 loop，loop 三圈的結果為 0.8180，四圈的結果為 0.8185，但原本沒有用 self training 時平均大概可以跑到 0.82，所以結果並沒有比較好，還要花更久的時間。