2023全國智慧製造大數據分析競賽決賽
團隊測驗報告

報名序號:112096
團隊名稱:Brute Force Buddy

註1:請用本PowerPoint 文件撰寫團隊測驗報告, 請轉成PDF檔案繳交。
註2:依據競賽須知第八條, 第5項規定:
決賽簡報之書面及口頭報告、服裝, 均不得使用學校系所標誌、提及學校系所、教授姓名及任何可供辨識參賽者身分的資料, 違者取消參賽資格, 或由主辦單位及評審會議決定處理方式
註3:請於11/25(六) 12:41前繳交團隊測驗報告及測驗結果, 至主辦單位指定網站。

【提醒】

11/25(六)請繳交兩種檔案:

1. 簡報檔, 檔名命名規則如下, 使用英文命名:
- **ProjectA:報名序號_ projectA _report.pdf, 例如:112999_projectA_report.pdf**
- **ProjectB:報名序號_ projectB _report.pdf, 例如:112999_projectB_report.pdf**

2. 決賽測驗結果檔, 檔名命名規則如下, 使用英文命名:
- **ProjectA:報名序號_projectA_ans.csv, 例如:112999_projectA_ans.csv**
- **ProjectB:報名序號_projectB_ans.csv, 例如:112999_projectB_ans.csv**

選擇(LGBM+XGB)或Fully Connected Network?

模型1: Fully Connected Network

一、資料前處理(說明資料前處理過程)

資料一 : Fully Connected Layer(加入source欄位)

首先, 我們將主辦方給的5個cvs檔分別加入source欄位, 例如:1.csv會新增一個欄位叫做source每個觀測值的source都會是1, 以此類雖.....
再將5個csv檔進行合併, 所以總共會有6876*5筆觀測值。
接著, 針對source欄位進行onehot encoding, 因此我們會有('f1', 'f2', 'f3', 'f4', 'source_1', 'source_2', 'source_3', 'source_4', 'source_5')九個特徵。
對其中四個特徵(f1,f2,f3,f4)分別進行z-score標準化, 再將標準化後的值丟入 fully connected network

資料二 : Fully Connected Layer(不加source欄位)

不加入source欄位將5個csv檔進行合併, 所以總共會有6876*5筆觀測值。
對四個特徵(f1,f2,f3,f4)分別進行z-score標準化再將標準化後的值丟入fully connected network

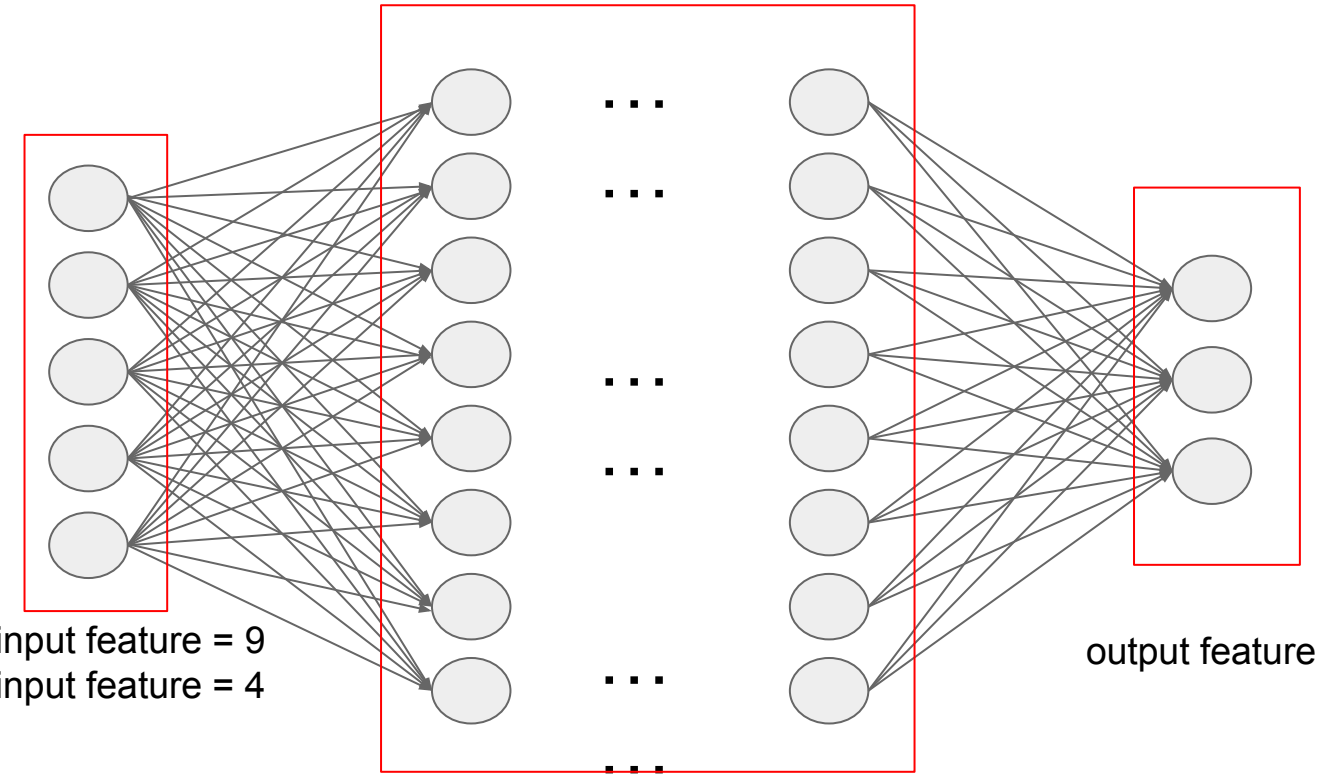
一、資料拆分

針對6876*5筆觀測值, 隨機選取20%資料當作測試集以用來評估模型的RMSE

並對模型進行5-fold cross-validation來評估模型穩定性,

再同一組參數下, 用在5-fold cross-validation表現最好(RMSE最低)的模型當作預選模型。

二、演算法和模型介紹(介紹方法細節)



資料一 : input feature = 9
資料二 : input feature = 4

headen layer
(ReLU function)

output feature = 3

batch size = 128
epoch = 2000
learning rate = 0.001
optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))
loss function = RMSE Loss

預選模型(不同組參數)

預選模型1(資料二)

```
hidden_layer1 = 40
hidden_layer2 = 30
hidden_layer3 = 20

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(in_features=4, out_features=hidden_layer1)
        self.fc2 = nn.Linear(in_features=hidden_layer1, out_features=hidden_layer2)
        self.fc3 = nn.Linear(in_features=hidden_layer2, out_features=hidden_layer2)
        self.fc4 = nn.Linear(in_features=hidden_layer2, out_features=hidden_layer3)
        self.fc5 = nn.Linear(in_features=hidden_layer3, out_features=len(y_col))

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = self.fc5(x)
        return x
```

預選模型2(資料二)

```
hidden_layer1 = 40
hidden_layer2 = 30
hidden_layer3 = 20

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(in_features=4, out_features=hidden_layer1)
        self.fc2 = nn.Linear(in_features=hidden_layer1, out_features=hidden_layer2)
        self.fc3 = nn.Linear(in_features=hidden_layer2, out_features=hidden_layer3)
        self.fc4 = nn.Linear(in_features=hidden_layer3, out_features=hidden_layer3)
        self.fc5 = nn.Linear(in_features=hidden_layer3, out_features=len(y_col))

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = self.fc5(x)
        return x
```

預選模型3(資料二)

```
hidden_layer1 = 30
hidden_layer2 = 20
hidden_layer3 = 30
hidden_layer4 = 20

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(in_features=4, out_features=hidden_layer1)
        self.fc2 = nn.Linear(in_features=hidden_layer1, out_features=hidden_layer2)
        self.fc3 = nn.Linear(in_features=hidden_layer2, out_features=hidden_layer3)
        self.fc4 = nn.Linear(in_features=hidden_layer3, out_features=hidden_layer4)
        self.fc5 = nn.Linear(in_features=hidden_layer4, out_features=hidden_layer4)
        self.fc6 = nn.Linear(in_features=hidden_layer4, out_features=len(y_col))

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = torch.relu(self.fc5(x))
        x = self.fc6(x)
        return x
```

預選模型(不同組參數)

預選模型4(資料一)

```
hidden_layer1 = 50
hidden_layer2 = 30
hidden_layer3 = 30
hidden_layer4 = 20


class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(in_features=9, out_features=hidden_layer1)
        self.fc2 = nn.Linear(in_features=hidden_layer1, out_features=hidden_layer2)
        self.fc3 = nn.Linear(in_features=hidden_layer2, out_features=hidden_layer2)
        self.fc4 = nn.Linear(in_features=hidden_layer3, out_features=hidden_layer4)
        self.fc5 = nn.Linear(in_features=hidden_layer4, out_features=len(y_col))


    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = self.fc5(x)
        return x
```

三、預選模型測試集上結果

模型	預選模型1 (資料二)	預選模型2 (資料二)	預選模型3 (資料二)	預選模型4 (資料一)
測試集 RMSE	0.8746	0.8683	0.8724	0.6734

三、加入6.csv訓練整體資料

模型	模型4(資料二)
測試集 RMSE	0.71

模型5架構(不加source)

```
hidden_layer1 = 50
hidden_layer2 = 30
hidden_layer3 = 30
hidden_layer4 = 20


class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(in_features=4, out_features=hidden_layer1)
        self.fc2 = nn.Linear(in_features=hidden_layer1, out_features=hidden_layer2)
        self.fc3 = nn.Linear(in_features=hidden_layer2, out_features=hidden_layer2)
        self.fc4 = nn.Linear(in_features=hidden_layer3, out_features=hidden_layer4)
        self.fc5 = nn.Linear(in_features=hidden_layer4, out_features=len(y_col))


    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = self.fc5(x)
        return x
```

模型2: LightGBM + XGBoost

一、資料前處理(說明資料前處理過程)

- 資料集合併
  - 將 1.csv ~ 5.csv 之資料集合併為大資料集(data), 並隨機打亂rows
  - 另對f1~f4欄位進行標準化(std_data), 以供後續選擇模型時使用
  - 由於預測欄位(y1~y3)之間在分布上具有差異(見下圖), 無法單純視為一體, 故採用「一個模型預測一個欄位」的方式進行訓練

	y1	y2	y3
平均值	-0.374714	1.024343	-56.654087
標準差	1.092701	4.221256	7.310101

二、演算法和模型介紹(介紹方法細節)

- 預選演算法 (candidate_models)
  - (1) Linear Regression　(2) RandomForest　(3) AdaBoost
  - (4) GradientBoosting　(5) LightGBM　(6) XGBoost
- 針對每種演算法, 計算兩種指標:
  - 在三個預測欄位上各自的平均RMSE(10摺交叉驗證)
  - 切分訓練/驗證資料集後, 計算在預測欄位上的整體RMSE(三個欄位共同計算RMSE)
- 上述程序進行12次(函式:EvaluateModels(candidate_models, iters=12, cv_num=10))
- 實驗結果:
  - 在預測y1、y2上, LightGBM有最小的平均RMSE (y1: 0.2232 / y2: 0.8672)
  - 在預測y3上, XGBoost有最小的平均RMSE (y3: 1.2394)
  - 以訓練集訓練後, 在驗證集上, XGBoost有最小的整體RMSE (0.8756)

二、演算法和模型介紹(介紹方法細節)

- 選擇模型組合:兩種方式
  - LLX —> y1:LGBMRegressor / y2:LGBMRegressor / y3:XGBRegressor
  - XXX —> y1:XGBRegressor / y2:XGBRegressor / y3:XGBRegressor
- 兩種方式再進行一次評估
  - 切分訓練/驗證資料集後(每次切分皆不同), 計算兩種方式在預測欄位上的整體RMSE
  - 上述程序進行100次後對RMSE取平均值 (函式:SecondEvaluation(iters=100))
- 實驗結果:
  - LLX 平均RMSE:0.88955 (較小)
  - XXX平均RMSE:0.89914
- 故選擇LLX方式作為主要模型組合

三、預選模型測試集上結果

	預測y1	預測y2	預測y3	整體
測試集 RMSE	0.2231	0.8672	1.239	0.82

三、加入6.csv訓練整體資料

模型	模型4(資料二)
測試集 RMSE	0.82

模型參數:(其他預設)
(LGBM)y1:n_estimators=305, learning rate=0.05
(LGBM)y2:n_estimators=75, learning rate=0.09
(XGB)　y3:n_estimators=95, learning rate=0.23

最終模型選擇

為什麼最後選擇LGBM+XGB當作最終模型?

儘管Fully Connected Network在測試集上的表現較好, 但是因為在Fully Connected Network的模型裡, 我們認為若是test.csv與我們的訓練資料有偏差, 則在做標準化後可能會大幅度的影響預測的準確度, 在測試集上對特徵做(樣本)標準化與(母體)標準化或多或少也會影響預測結果, 反之, LGBM+XGB的模型是沒有經過標準化處理的, 所以我們認為在projectA採用機器學習方法會有比較穩定的結果。

執行環境/套裝選擇/執行方式

- 使用 tensorflow & keras 環境
  - 輸入「cd /envs/tf_keras」進入/envs/tf_keras後執行「pipenv shell」即進入tensorflow & keras環境
- 進入環境後, 進入/home/112096/Eric 的資料夾裡
- 執行「python3 projectA_FinalTest.py」後, 即可在目錄下看到112096_projectA_ans.csv的檔案, 此檔案為最終預測結果

執行環境/套裝選擇/執行方式

- 套件
  - 資料處理：numpy, pandas
  - sklearn相關
    - 模型：RandomForestRegressor、GradientBoostRegressor、AdaBoostRegressor、LinearRegression
    - 其他：train_test_split、cross_val_score、StandardScaler
  - xgboost.XGBRegressor、lightgbm.LGBMRegressor
  - 模型儲存與載入：joblib

補充說明