Final Project

GitHub Link: https://github.com/pin-chen/Intro-ML-Final-Project

model link: https://github.com/pin-chen/Intro-ML-Final-Project/blob/main/final_model.pickle

test data link: https://github.com/pin-chen/Intro-ML-Final-Project/blob/main/x_test.npy

Final version

Environment details:

使用 Kaggle 進行 train 與 inference ，因此環境設置與 Kaggle 一樣

Python version: 3.7.12

Accelerator: GPU P100

Python 機器學習庫: sklearn

```
import numpy as np
import pandas as pd

import pickle

from sklearn import linear_model
from sklearn.linear_model import HuberRegressor
from sklearn.impute import SimpleImputer
from sklearn.impute import KNNImputer

import warnings
warnings.filterwarnings('ignore')
```

Implementation details:

select features

經幾次測試及 kaggle discussion 公開的 Logistic Regression 相關測試，'loading' 和 'measurement_17' 是比較重要的 features ，於這份資料而言增加其他 features ，而沒有妥善的處理，只會成為干擾項。另外對於 'loading' 和 'measurement_17' 的 miss values 的 predict ，根據部分 discussion ，其有一些複雜的關係，為簡化設計，我只選了 'measurement_9' 和 'measurement_14' ，以上共四項。

fill miss values

將所有資料依照 miss values 的種類進行劃分，以全部都有的資料訓練四種 HuberRegressor ，分別為任三種資料預測第四種資料的。

將只缺少一種 values 的資料透過訓練好的四種 model 預測補回去。

剩下的 miss values 都透過 KNNImputer 進行預測。

後來只留兩種，分別為預測 'measurement_9' 和 'measurement_14' 的，因為發現 KNNImputer 本身即具有步錯的效果。且預測完 'measurement_9' 和 'measurement_14' ，再使用 KNNImputer 填 'loading' 和 'measurement_17' 效果更佳。

train

model 使用 `linear_model.LogisticRegression(max_iter=500, C=0.0001, penalty='l2', solver='newton-cg')`

只選 'loading' 和 'measurement_17' 這兩個 features 進行，經測試，增加其他 features 都會成為干擾項。

Hyperparameters:

經實測參數造成的效果如下：

`penalty='l2', solver='newton-cg'` > `penalty='l2', solver='lbfgs'` >> `penalty='l2', solver='sag'` > `penalty='l2', solver='liblinear'` > `penalty='l1', solver='liblinear'`

`max_iter` 則是不要太低都差太多，`C=0.01~0.000001` 都差不多

進行 cross validation 時，採用根據 product code 進行分組，因 test data 的 product code 與 training data 完全不同，因此這樣分組可以看出較好的成果。

inference

Load test data, model, and sample submission.

因為我們在 train code 中，將 test data 的 miss value 填完了，因此需讀入 test data ，並根據 model 進行 predict.

再輸出成 csv

Results

Versions Update Process

最初開始進行時，並未查看 discussion ，直接先使用一個簡易的 CNN model ，細節如下：

Environment details:

使用 Kaggle 進行 train 與 inference ，因此環境設置與 Kaggle 一樣

Python version: 3.7.12

Accelerator: GPU P100

Python 機器學習庫: PyTorch

```
import csv
import numpy as np
import random
import os

from PIL import Image

import torch as t
import torch.nn as nn
import torch.nn.functional as F
from torchvision import transforms as  T
from torch.utils.data import Dataset, DataLoader
```

Implementation details:

這些版本的 model 大約如下，只有一些層之間的細節調整：

```
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv1d(1, 4, 3)
        self.conv2 = nn.Conv1d(4, 16, 3)
        self.conv3 = nn.Conv1d(16, 32, 3)
        self.conv4 = nn.Conv1d(32, 128, 5)
        self.fc1 = nn.Linear(512, 128)
        self.fc2 = nn.Linear(128, 64)
```

```
        self.fc3 = nn.Linear(64, 32)
        self.fc4 = nn.Linear(32, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = x.view(-1, 512)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        x = self.relu(x)
        x = self.fc4(x)
        return x
```

另外一開始對於 miss value 的處理只有簡單的轉為 0 以及轉為平均值而已。

而這些版本的結果，對所有測資都 Predict 成 10^{-5} 以上，非常趨近於 0 的數字。

最初認為可能資料太少，訓練太淺所以效果不好，因為於 HW 5 時，我就將 training data 放大到 1 GB ，但顯然這次的資料不是可以自己產生的。因此先嘗試將資料拼成二維丟到 HW5 的 model 進行嘗試。

確實結果變好了，但出現更多甚至直接 Predict 成 0 的答案。至此認為這樣往下走的路是走不下去的。

而後開始翻閱其他人的 discussion ，有看到第一名是使用NN，但除此之外大部分都只使用 Logistic Regression ，原本是有想過是否使用 AdaBoost 或 decision tree 進行實作，畢竟因為資料維度不高，可能深度學習效果不高，且目前進行過的類似這樣資料形式分類問題是使用 decision tree 等進行實作的，但繳交答案不是 0 或 1 ，而是機率，且使用 ROC Curve 進行評分，故並未進行嘗試。

CNN 的內容過於複雜，決定先跳過。然後使用 Logistic Regression 的，基本上在 Logistic Regression 都只有很簡單的時實作，而重心則都放在 miss values 的復原，這是我沒有思考過的點。

EDA which makes sense: https://www.kaggle.com/code/ambrosm/tpsaug22-eda-which-makes-sense

如上述這篇對於所有資料的細節都進行了完整的分析。我後續實作方式主要是從這邊進參考。

Less can be more: Feature Engineering Ideas: https://www.kaggle.com/competitions/tabular-playground-series-aug-2022/discussion/342126

上述這篇提出了幾個 attributes 可能代表的意義，及透過相乘進行合併的可能性。據我簡單實測的結果影響不大，最後甚至他提到的 features 都被我丟掉了。

另外要進行資料處理時因 features 中有字串資料，原本是強制取後面的數字，根據 EDA which makes sense 這篇的說明可以使用 One-Hot Encoding 。

不要再做One Hot Encoding！！: https://axk51013.medium.com/%E4%B8%8D%E8%A6%81%E5%86%8D%E5%81%9Aone-hot-encoding-b5126d3f8a63

因此去找了相關的實作，嘗試了 Frequency Encoding 的效果不是太好，可能是因為此次資料的性質的關係，另外提及的 Target Encoding ，就沒有嘗試進行實作。

Handling Missing Values: https://www.kaggle.com/code/dansbecker/handling-missing-values/notebook

kaggle 在我們直接丟有 miss value 的資料給 model 時，跳出的處理方式，一開始選用 SimpleImputer 並去掉所有 attributes ，接著直接 Logistic Regression。

submission (12).csv Complete (after deadline) · 16h ago	0.58448	0.56881

有了卓越的效果提升，再換成 KNNImputer 。

submission (13).csv Complete (after deadline) · 15h ago	0.58895	0.58589

離 baseline 不遠了。

再來想到進行完 fill miss value 的操作，再交由 CNN 跑結果會如何？

submission (2).csv Complete (after deadline) · 17h ago · knn impute cnn	0.58834	0.58112

也呈現出良好的效果，所以之前大多趨於 0 的原因是 miss value 的噪音太大的緣故。

TPS-Aug22 9th solution: https://www.kaggle.com/code/takanashihumbert/tps-aug22-9th-solution/notebook

因為這次的資料所有 features 可能對於是否壞掉的影響都不夠大，因此滿多屬於干擾項，由釋出的最高名次 Logistic Regression 所使用到的 features 分別為 'loading' 和 'measurement_17' ，因此後我就只留這兩個 features 在 Logistic Regression 時，至於 fill miss value 還是根據其他資料一起進行，以此為基礎的最高成果如下。

submission (2).csv Complete (after deadline) · 15h ago	0.59014	0.58672

Perfect Positive Correlation with measurement_17: https://www.kaggle.com/competitions/tabular-playground-series-aug-2022/discussion/343939

再來透過幾篇對於間關係的 discussion (EDA which makes sense) ，進行 HuberRegressor 去 fill miss value ，細節如 Final version 所述，最後挑出的 features 也如 Final version 所述。

另外還是有嘗試使用 decision tree 進行時作，但就只能輸出 0 或 1 。`from sklearn import tree`

submission.csv Complete (after deadline) · 17h ago	0.50722	0.51252