Github Link: https://github.com/uuugaga/ML_Fianl_project.git

Model Link:
https://drive.google.com/file/d/1exJ4QAGHXSYWwXOAkm3r379otC86mroz/view?usp=share_link
Model 很小，已附在 github 上，照理說可以直接 git clone 專案，直接跑 inference

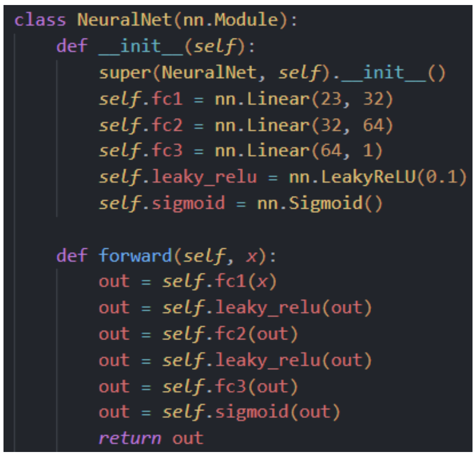
Brief introduction:

這份作業我選擇用一個簡單的 nn model，平均有不錯的效果，但這個 Competition 已截止，沒有上傳限制，所以就利用 kaggle 的 api，讓 model 不斷的 train 與 submit，若比較高則把 model 存下來，整份作業共 submit 了一千多次。(為此 model 並沒有學的很深，因為會 overfitting，導致效果不佳)

Methodology:

Data pre-process: 我取了” loading”, “attribute_{0-3}”, ” measurement_{0-17}”，把他們轉成 float，其中 attribute_{0-1}格式為”material_{number}”，我只取{number}的部分，而這個 dataset 有 10%的資料空缺，我用 median 的方式填值(使用 sklearn.impute.SimpleImputer)

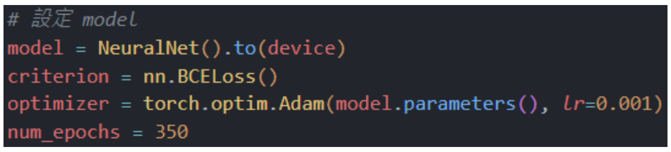
Model architecture: 一個簡單的 nn model



```
class NeuralNet(nn.Module):
    def __init__(self):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(23, 32)
        self.fc2 = nn.Linear(32, 64)
        self.fc3 = nn.Linear(64, 1)
        self.leaky_relu = nn.LeakyReLU(0.1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        out = self.fc1(x)
        out = self.leaky_relu(out)
        out = self.fc2(out)
        out = self.leaky_relu(out)
        out = self.fc3(out)
        out = self.sigmoid(out)
        return out
```

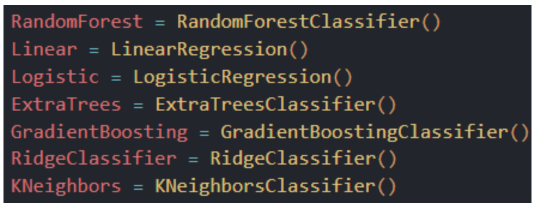
Hyperparameters:



```
# 設定 model
model = NeuralNet().to(device)
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
num_epochs = 350
```

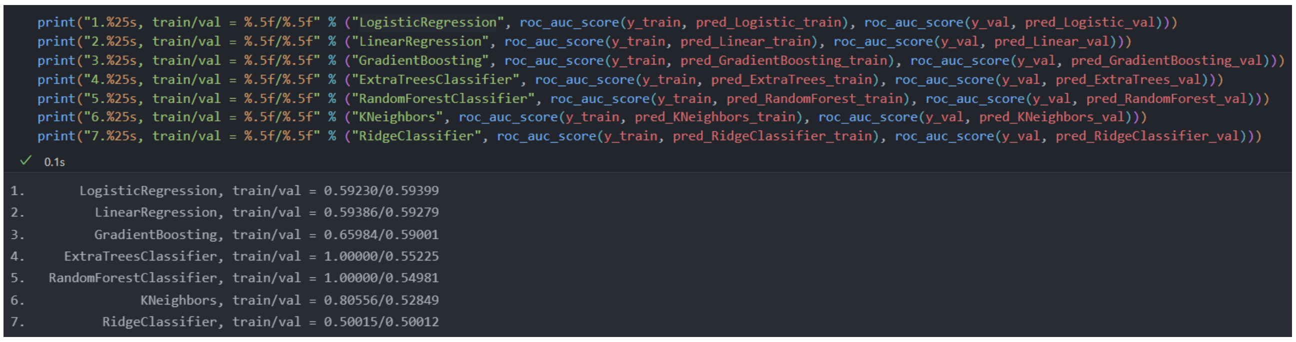
Research:

我嘗試了許多現成的 Classifier



```python
RandomForest = RandomForestClassifier()
Linear = LinearRegression()
Logistic = LogisticRegression()
ExtraTrees = ExtraTreesClassifier()
GradientBoosting = GradientBoostingClassifier()
RidgeClassifier = RidgeClassifier()
KNeighbors = KNeighborsClassifier()
```

效果都沒有 nn model 好，最好的 LogisticRegression 也僅達到 baseline 而已



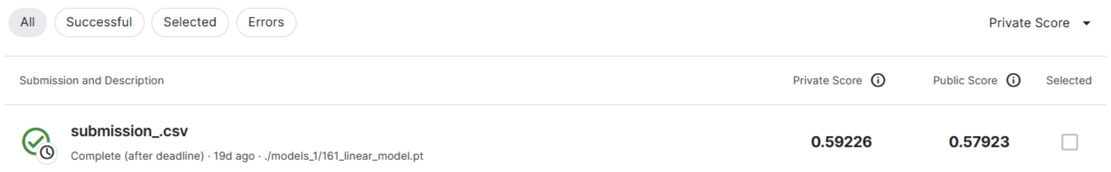
```python
print("1.%25s, train/val = %.5f/%.5f" % ("LogisticRegression", roc_auc_score(y_train, pred_Logistic_train), roc_auc_score(y_val, pred_Logistic_val)))
print("2.%25s, train/val = %.5f/%.5f" % ("LinearRegression", roc_auc_score(y_train, pred_Linear_train), roc_auc_score(y_val, pred_Linear_val)))
print("3.%25s, train/val = %.5f/%.5f" % ("GradientBoosting", roc_auc_score(y_train, pred_GradientBoosting_train), roc_auc_score(y_val, pred_GradientBoosting_val)))
print("4.%25s, train/val = %.5f/%.5f" % ("ExtraTreesClassifier", roc_auc_score(y_train, pred_ExtraTrees_train), roc_auc_score(y_val, pred_ExtraTrees_val)))
print("5.%25s, train/val = %.5f/%.5f" % ("RandomForestClassifier", roc_auc_score(y_train, pred_RandomForest_train), roc_auc_score(y_val, pred_RandomForest_val)))
print("6.%25s, train/val = %.5f/%.5f" % ("KNeighbors", roc_auc_score(y_train, pred_KNeighbors_train), roc_auc_score(y_val, pred_KNeighbors_val)))
print("7.%25s, train/val = %.5f/%.5f" % ("RidgeClassifier", roc_auc_score(y_train, pred_RidgeClassifier_train), roc_auc_score(y_val, pred_RidgeClassifier_val)))
```

✓ 0.1s

```
1.       LogisticRegression, train/val = 0.59230/0.59399
2.         LinearRegression, train/val = 0.59386/0.59279
3.         GradientBoosting, train/val = 0.65984/0.59001
4.     ExtraTreesClassifier, train/val = 1.00000/0.55225
5.   RandomForestClassifier, train/val = 1.00000/0.54981
6.               KNeighbors, train/val = 0.80556/0.52849
7.          RidgeClassifier, train/val = 0.50015/0.50012
```

(上圖的結果已排序過)

Result:



Submission and Description	Private Score	Public Score	Selected
submission_.csv Complete (after deadline) · 19d ago · ./models_1/161_linear_model.pt	0.59226	0.57923	☐