Assignment2 COVID-19 30-day Mortality Prediction Report

106062314 蔡政諺

Implementation

https://colab.research.google.com/drive/1-LoZ-cgyT0D0oEK0BbWZ57D2E6aExLVI

Descriptions

本次作業我使用 Keras 實作 neural network。

首先對資料進行前處理：

1. 對 missing value 做插值。

```
# impute missing values
X = X.fillna(X.mean())
```

由於 class label 中沒有 missing value，此處只對數字類的 missing value 取該 column 的平均做插值。

2. 將 class label 轉成 one-hot。

```
# encode class input to one-hot
X['sex'] = X['sex'].map({'MALE': 0, 'FEMALE': 1})
ed_diagnosis_list = ['sx_breathing_difficulty', 'sx_flu', 'sx_fever', 'sx_cough', 'sx_others']
for element in ed_diagnosis_list:
    X[element] = X['ed_diagnosis'].map(lambda x: 1 if x == element else 0)
```

在 sex 這個 column，我把 MALE map 成 0、FEMALE map 成 1。

在 ed_diagnosis 這個 column，由於有五種 label，我將它們各自 expand 成一個 column，做成五個 column 的 one-hot，才不會使這五個 label 有 regression 的關係。

3. 將不需要的 column 刪掉。

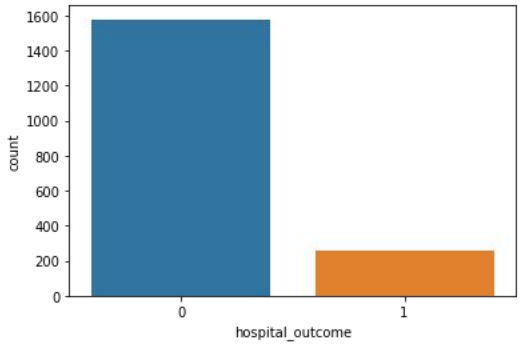
```
# drop useless information
X = X.drop(['PATIENT ID', 'admission_datetime', 'ed_diagnosis'], axis=1)
Y = Y.drop(['PATIENT ID'], axis=1)
```

不需要的 column 包含 PATIENT ID、admission_datetime、ed_diagnosis，不需要 ed_diagnosis 的原因是已經將它轉換成 one-hot 了，故將原本的 column 移除。

做完前處理後，用 train_test_split 將 34%的 data 做為 testing，其餘做為 training。

```
# split training and testing data
X = np.asarray(X)
Y = np.asarray(Y)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.34, random_state=0)
print("X_train shape:", X_train.shape)
print("Y_train shape:", Y_train.shape)
print("X_test shape:", X_test.shape)
print("Y_test shape:", Y_test.shape)
```

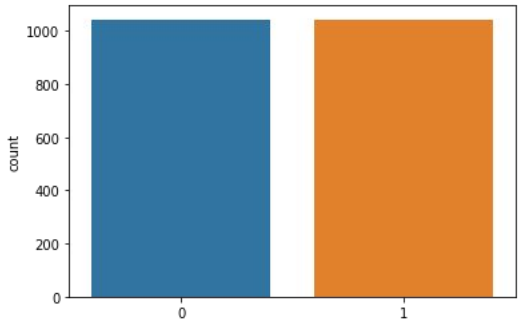
```
X_train shape: (1210, 50)
Y_train shape: (1210, 1)
X_test shape: (624, 50)
Y_test shape: (624, 1)
```



由於 training data 的分布非常不均衡，而像這樣的 imbalanced data 會導致 training 難以收斂，因此我使用 SMOTE 演算法對 label 為 1 的 data 進行 oversampling。

```
# handle imbalanced data (oversampling)
smote = SMOTE(ratio='minority')
X_train, Y_train = smote.fit_sample(X_train, Y_train)
print("X_train shape:", X_train.shape)
print("Y_train shape:", Y_train.shape)
print("X_test shape:", X_test.shape)
print("Y_test shape:", Y_test.shape)
sns.countplot(Y_train, label="Sum")
```

可以看到 training data 中，label 為 0 與 1 的資料筆數達到平衡了。



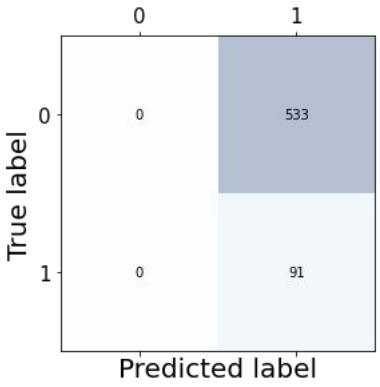
接下來進入 training phase。

```
Training
    Dummy
    Logistic Regression
    Naive Bayes
    KNN
    SVC
    Decision Tree
    Random Forest
    XGBoost
    Ensemble
    Neural Network
```

首先使用非 neural network 架構的 model 做評估：

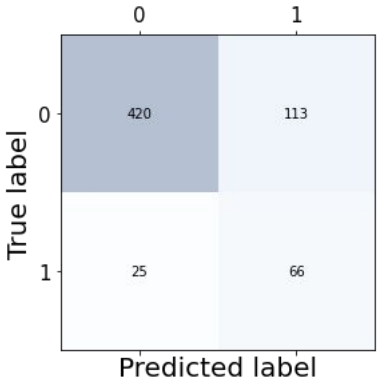
1. Dummy

   所有結果都輸出為 1，這個結果可以視為 baseline，與後續的方法做比較。



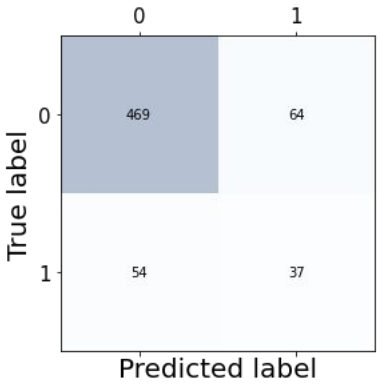

```
Accuracy: 0.14583333333333334
Precision score: 0.14583333333333334
Recall score: 1.0
P+R: 1.1458333333333333
```

2. Logistic Regression



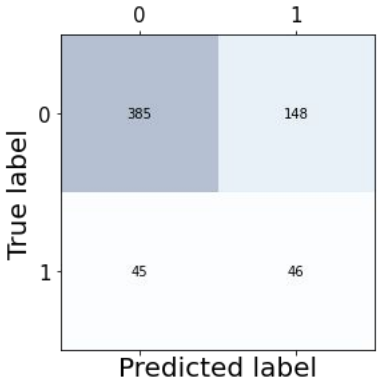

```
Accuracy: 0.7788461538461539
Precision score: 0.3687150837988827
Recall score: 0.7252747252747253
P+R: 1.093989809073608
```

3. Naive Bayes



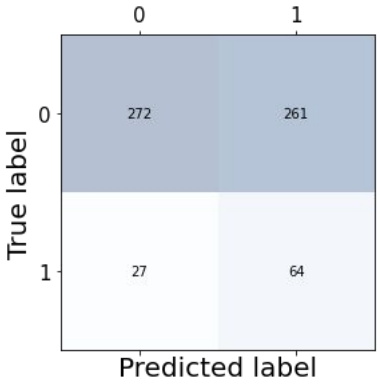

```
Accuracy: 0.8108974358974359
Precision score: 0.36633663366336633
Recall score: 0.4065934065934066
P+R: 0.7729300402567729
```

4. KNN



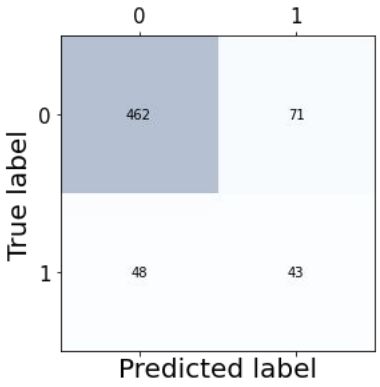

```
Accuracy: 0.6907051282051282
Precision score: 0.23711340206185566
Recall score: 0.5054945054945055
P+R: 0.7426079075563612
```

5. SVC



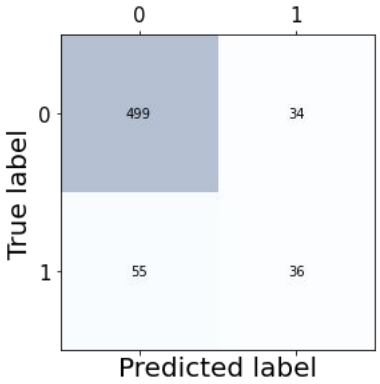

```
Accuracy: 0.5384615384615384
Precision score: 0.19692307692307692
Recall score: 0.7032967032967034
P+R: 0.9002197802197802
```

6. Decision Tree



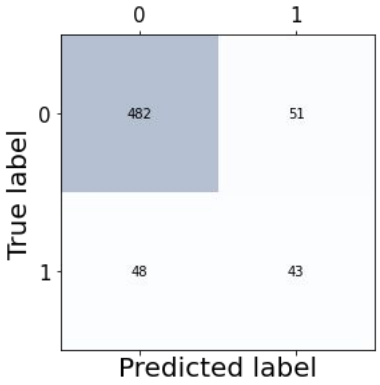

```
Accuracy: 0.8092948717948718
Precision score: 0.37719298245614036
Recall score: 0.4725274725274725
P+R: 0.8497204549836128
```

7. Random Forest




```
Accuracy: 0.8573717948717948
Precision score: 0.5142857142857142
Recall score: 0.3956043956043956
P+R: 0.9098901098901098
```

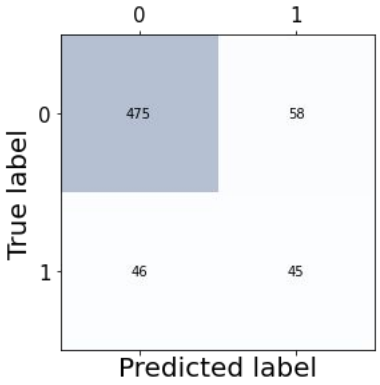
8. XGBoost




```
Accuracy: 0.8413461538461539
Precision score: 0.4574468085106383
Recall score: 0.4725274725274725
P+R: 0.9299742810381109
```

9. Ensemble

使用上面的 7 個 model 做 voting，票數最高的結果作為答案。




```
Accuracy: 0.8333333333333334
Precision score: 0.4368932038834951
Recall score: 0.4945054945054945
P+R: 0.9313986983889897
```

上面的方法在大多數情況皆無法超越 dummy model 的 performance。因此，我改為使用 Keras 實作 neural network。

```
# build model
model = keras.models.Sequential([
        keras.layers.Flatten(input_dim=X.shape[1]),
        keras.layers.BatchNormalization(),
        keras.layers.Dense(units=32, activation='relu'),
        keras.layers.Dropout(0.5),
        keras.layers.BatchNormalization(),
        keras.layers.Dense(units=32, activation='relu'),
        keras.layers.Dropout(0.5),
        keras.layers.BatchNormalization(),
        keras.layers.Dense(units=32, activation='relu'),
        keras.layers.Dropout(0.5),
        keras.layers.BatchNormalization(),
        keras.layers.Dense(units=1, activation='sigmoid')
])
```

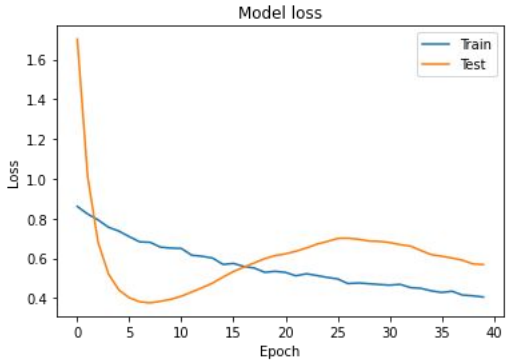
我建了一個共三層 hidden layer 的 network，每層有 32 個 neuron，並加入 dropout 做 regularization，避免 overfitting。此外，我也加入 batch normalization layer，使 training 的過程更加穩定。每層 hidden layer 的 activation function 使用 ReLU，output layer 則使用 sigmoid，使輸出結果為一個 0 到 1 的機率。

```
model.compile(optimizer='adam', loss="binary_crossentropy", metrics=['AUC'])
model.summary()
```

我使用 adam 做為 optimizer，binary crossentropy 做為 loss function (因為是做 binary classification)。

```
# train model
history = model.fit(X_train, Y_train, batch_size=256, epochs=40, validation_data=(X_test, Y_test))
```

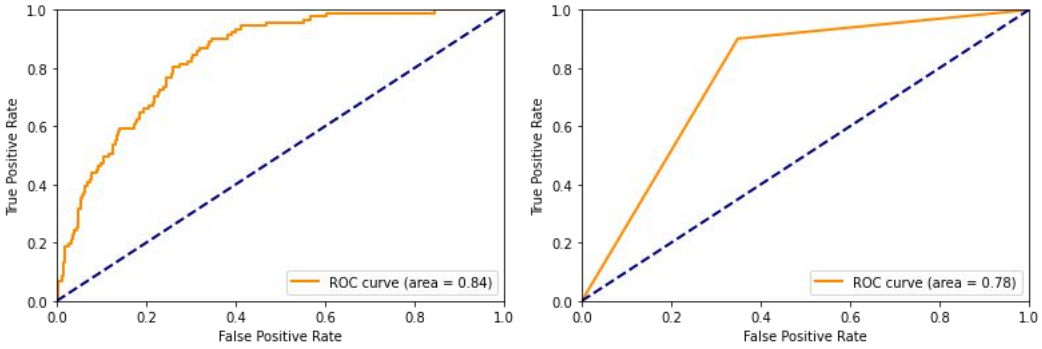
在 training 中，我設定 batch size 為 256，共 train 40 個 epochs，並拿剛才 train_test_split 切出來的 testing data 做 validation。



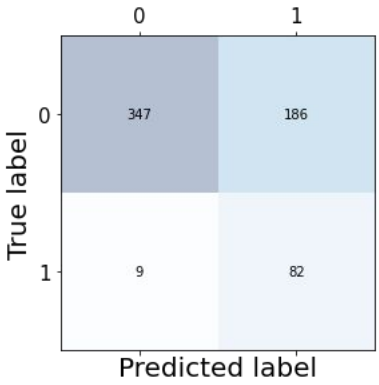
可以看到 training loss 穩定地在下降，testing loss 也有隨著 training 的過程收斂。

```
# make predictions
Y_pred = model.predict(X_test)
evaluate(Y_test, Y_pred, isFloat=True)
```

接著就可以對 validation data 做 predict，並與真實的 label 做比較。



左圖是拿 model predict 的結果與真實的 label 畫 ROC curve，右圖是拿 model predict 後 round 成整數的結果與真實的 label 畫 ROC curve。可以發現 round 之後的 AUROC 會比 round 後稍低一點，這是因為 round 後會喪失一些精準度。




```
Accuracy: 0.6875
Precision score: 0.30597014925373134
Recall score: 0.9010989010989011
P+R: 1.2070690503526325
```

在 validation data 上，precision 與 recall 分別達到大約 0.3 與 0.9 的表現，recall 很高表示對於 true label 為 1 的 data 中，我們有很高的比率做到正確的分類。

```
# make directory
if not os.path.exists(filepath + 'output'):
    os.makedirs(filepath + 'output')

# save model
model.save(filepath + "output/106062314_HW2_Model.h5")
```

最後，將 model 輸出成一個 h5 檔，即可保存此次 training 的結果。

- **How to use the model file**

1. 對 missing value 做插值。

```
# impute missing values
testing_data = testing_data.fillna(testing_data.mean())
```

2. 將 class label 轉成 one-hot。

```
# encode class input to one-hot
testing_data['sex'] = testing_data['sex'].map({'MALE': 0, 'FEMALE': 1})
ed_diagnosis_list = ['sx_breathing_difficulty', 'sx_flu', 'sx_fever', 'sx_cough', 'sx_others']
for element in ed_diagnosis_list:
    testing_data[element] = testing_data['ed_diagnosis'].map(lambda x: 1 if x == element else 0)
```

3. 將不需要的 column 刪掉。

```
# drop useless information
testing_data = testing_data.drop(['PATIENT ID', 'admission_datetime', 'ed_diagnosis'], axis=1)
```

4. 用 load_model 將 model 檔讀出來。

```
model_load = keras.models.load_model(filepath + "output/106062314_HW2_Model.h5")
```

5. 呼叫 predict，並將回傳值 round 成整數，即是對 testing data 的預測結果。

```
# make predictions
Y_pred = model_load.predict(testing_data).round()
```