

# notebook

March 20, 2022

```
[ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
```

```
[ ]: data = pd.read_csv('Data/data.csv')
data.head()
```

```
[ ]:
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	\
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
0	0.11840	0.27760	0.3001	0.14710	
1	0.08474	0.07864	0.0869	0.07017	
2	0.10960	0.15990	0.1974	0.12790	
3	0.14250	0.28390	0.2414	0.10520	
4	0.10030	0.13280	0.1980	0.10430	

	texture_worst	perimeter_worst	area_worst	smoothness_worst	\
0	17.33	184.60	2019.0	0.1622	
1	23.41	158.80	1956.0	0.1238	
2	25.53	152.50	1709.0	0.1444	
3	26.50	98.87	567.7	0.2098	
4	16.67	152.20	1575.0	0.1374	

	compactness_worst	concavity_worst	concave points_worst	symmetry_worst	\
0	0.6656	0.7119	0.2654	0.4601	
1	0.1866	0.2416	0.1860	0.2750	
2	0.4245	0.4504	0.2430	0.3613	
3	0.8663	0.6869	0.2575	0.6638	
4	0.2050	0.4000	0.1625	0.2364	

	fractal_dimension_worst	Unnamed: 32
0	0.11890	NaN

1	0.08902	NaN
2	0.08758	NaN
3	0.17300	NaN
4	0.07678	NaN

[5 rows x 33 columns]

## 1 Tiền xử lý dữ liệu

```
[ ]: a = data.isnull().sum()
a[a != 0]
```

```
[ ]: Unnamed: 32      569
dtype: int64
```

Tìm hiểu xem các cột có kiểu dữ liệu gì ?

```
[ ]: data.dtypes
```

```
[ ]: id                int64
diagnosis             object
radius_mean           float64
texture_mean          float64
perimeter_mean        float64
area_mean             float64
smoothness_mean       float64
compactness_mean      float64
concavity_mean        float64
concave points_mean   float64
symmetry_mean         float64
fractal_dimension_mean float64
radius_se             float64
texture_se            float64
perimeter_se          float64
area_se              float64
smoothness_se         float64
compactness_se        float64
concavity_se          float64
concave points_se     float64
symmetry_se           float64
fractal_dimension_se  float64
radius_worst          float64
texture_worst         float64
perimeter_worst       float64
area_worst            float64
smoothness_worst      float64
compactness_worst     float64
```

```

concavity_worst          float64
concave points_worst      float64
symmetry_worst           float64
fractal_dimension_worst   float64
Unnamed: 32              float64
dtype: object

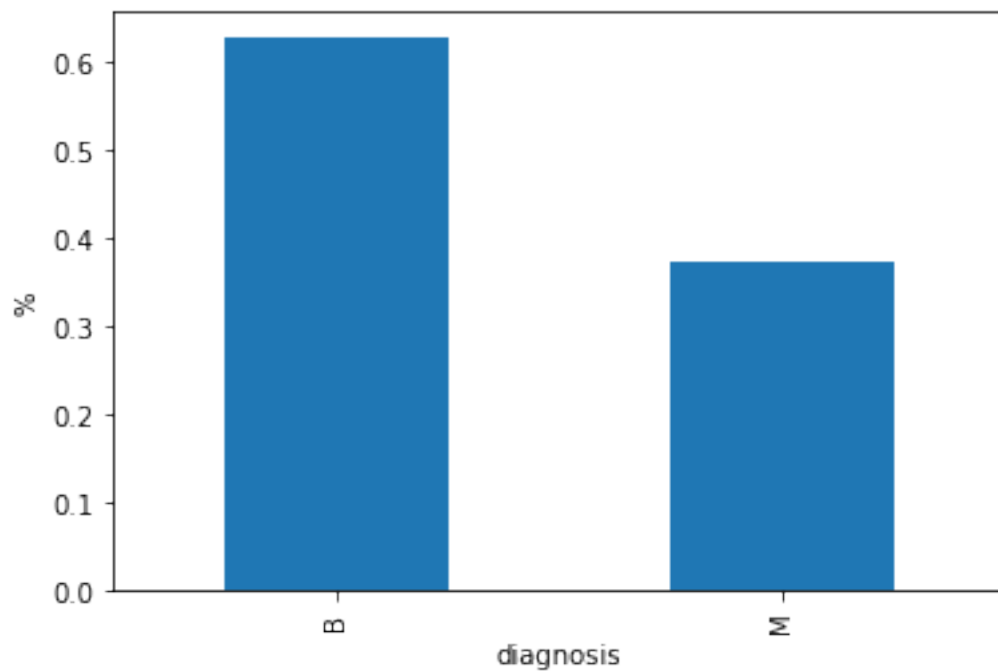
```

Nhận thấy cột id và cột Unnamed không có tác dụng cho việc phân loại nên chúng ta sẽ xoá cột này đi. `diagnosis` dạng categorical em sẽ chuyển về dạng numeric sau. Các cột dữ liệu còn lại đều có kiểu numeric.

```
[ ]: data = data.drop(['id', 'Unnamed: 32'], axis=1)
```

Biến cần phân lớp (target variable) ở đây là biến `diagnosis`, biến có 2 giá trị M = malignant, B = benign có dạng categorical

```
[ ]: # Tỷ lệ các lớp trong tập nhãn.
data.diagnosis.value_counts(normalize=True).plot(kind="bar")
data['diagnosis'].value_counts(normalize=True) * 100
plt.xlabel('diagnosis')
plt.ylabel('%');
```



Tỷ lệ lớp B nhiều hơn so với lớp M nhưng nhìn chung mọi thứ vẫn ổn vì không bị lệch quá nhiều.

```
[ ]: data.describe()
```

```

[ ]:      radius_mean  texture_mean  perimeter_mean  area_mean  \
count    569.000000    569.000000    569.000000    569.000000
mean      14.127292    19.289649     91.969033    654.889104
std        3.524049     4.301036     24.298981    351.914129
min        6.981000     9.710000     43.790000    143.500000
25%       11.700000    16.170000     75.170000    420.300000
50%       13.370000    18.840000     86.240000    551.100000
75%       15.780000    21.800000    104.100000    782.700000
max       28.110000    39.280000    188.500000   2501.000000

      smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
count    569.000000    569.000000    569.000000    569.000000
mean        0.096360     0.104341     0.088799     0.048919
std         0.014064     0.052813     0.079720     0.038803
min         0.052630     0.019380     0.000000     0.000000
25%         0.086370     0.064920     0.029560     0.020310
50%         0.095870     0.092630     0.061540     0.033500
75%         0.105300     0.130400     0.130700     0.074000
max         0.163400     0.345400     0.426800     0.201200

      symmetry_mean  fractal_dimension_mean  ...  radius_worst  \
count    569.000000    569.000000  ...    569.000000
mean        0.181162     0.062798  ...    16.269190
std         0.027414     0.007060  ...     4.833242
min         0.106000     0.049960  ...     7.930000
25%         0.161900     0.057700  ...    13.010000
50%         0.179200     0.061540  ...    14.970000
75%         0.195700     0.066120  ...    18.790000
max         0.304000     0.097440  ...    36.040000

      texture_worst  perimeter_worst  area_worst  smoothness_worst  \
count    569.000000    569.000000    569.000000    569.000000
mean      25.677223    107.261213    880.583128     0.132369
std        6.146258     33.602542    569.356993     0.022832
min       12.020000     50.410000    185.200000     0.071170
25%       21.080000     84.110000    515.300000     0.116600
50%       25.410000     97.660000    686.500000     0.131300
75%       29.720000    125.400000   1084.000000     0.146000
max       49.540000    251.200000   4254.000000     0.222600

      compactness_worst  concavity_worst  concave points_worst  \
count    569.000000    569.000000    569.000000
mean        0.254265     0.272188     0.114606
std         0.157336     0.208624     0.065732
min         0.027290     0.000000     0.000000
25%         0.147200     0.114500     0.064930
50%         0.211900     0.226700     0.099930

```

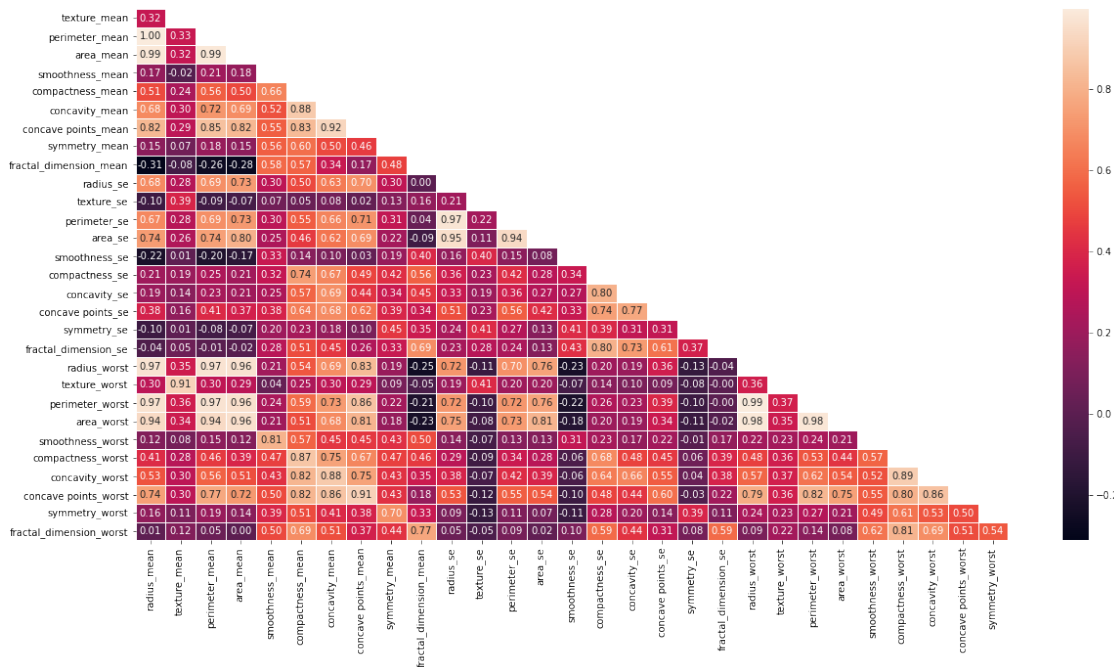
75%	0.339100	0.382900	0.161400
max	1.058000	1.252000	0.291000

	symmetry_worst	fractal_dimension_worst
count	569.000000	569.000000
mean	0.290076	0.083946
std	0.061867	0.018061
min	0.156500	0.055040
25%	0.250400	0.071460
50%	0.282200	0.080040
75%	0.317900	0.092080
max	0.663800	0.207500

[8 rows x 30 columns]

Để có cái nhìn trực quan hơn về những cột dữ liệu trên thì em sẽ trực quan tính tương quan giữa các biến dữ liệu.

```
[ ]: mask = np.triu(np.ones_like(data.corr(), dtype = bool))
plt.figure(figsize=(20,10))
sn.heatmap(data=data.corr().iloc[1:, :-1], mask=mask[1:, :-1], linewidths = 1,
            annot = True, fmt = ".2f");
```



Em có một vài nhận xét về biểu đồ này: - Có nhiều cặp biến có độ tương quan rất cao. Có thể thấy những cặp biến có độ tương quan > 0.9. VD: **perimeter\_mean** và **perimeter\_worst** có độ tương quan 0.97, và còn rất nhiều cặp biến như vậy. - Khi có nhiều biến tương quan như vậy sẽ

dẫn đến hiện tượng suy diễn thống kê không chắc chắn, giảm độ chính xác của mô hình. Nên em sẽ xóa đi khi có độ tương quan lớn hơn 0.9. - Thật sự thì độ tương quan bao nhiêu là cao thì em không biết chắc, em chọn 0.9 vì đây là con số phổ biến mà nhiều bài toán vẫn sử dụng.

```
[ ]: corr_matrix = data.corr().abs()

mask = np.triu(np.ones_like(corr_matrix, dtype = bool))
tri_df = corr_matrix.mask(mask)

dropped_feature = [x for x in tri_df.columns if any(tri_df[x] > 0.9)]
data = data.drop(dropped_feature, axis = 1)
```

```
[ ]: # Kiểm tra xem dữ liệu hiện tại còn bao nhiêu cột
data.shape
```

```
[ ]: (569, 21)
```

Dữ liệu từ 33 cột sau một vài bước từ bỏ những cột không cần thiết và loại bỏ các cột tương quan cao (> 0.9) thì dữ liệu hiện tại bao gồm 21 cột (bao gồm 1 cột nhãn). Chúng ta sẽ lưu kết quả vào file mới là preprocess\_data và khám phá xem những cột còn lại là những cột gì

```
[ ]: data.to_csv('Data/preprocess_data.csv')
data.columns
```

```
[ ]: Index(['diagnosis', 'smoothness_mean', 'compactness_mean', 'symmetry_mean',
          'fractal_dimension_mean', 'texture_se', 'area_se', 'smoothness_se',
          'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
          'fractal_dimension_se', 'texture_worst', 'area_worst',
          'smoothness_worst', 'compactness_worst', 'concavity_worst',
          'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst'],
          dtype='object')
```

Em sẽ chia data thành 2 tập train và test

```
[ ]: from sklearn.model_selection import train_test_split
X = data.drop('diagnosis', axis=1)
y = data['diagnosis']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
                                                    random_state = 0)
```

Vì đây là bài toán phân loại nhị phân nên em sẽ dùng mô hình phân loại nhị phân cơ bản Decision Tree Classifier trước để có một base model

```
[ ]: from sklearn.tree import DecisionTreeClassifier

lr_model = DecisionTreeClassifier()
lr_model.fit(X_train, y_train)
```

```
[ ]: DecisionTreeClassifier()
```

```
[ ]: from sklearn.metrics import accuracy_score, confusion_matrix

# Nhận được dự đoán từ mô hình vừa huấn luyện
y_pred = lr_model.predict(X_test)

# Một vài metrics để đánh giá mô hình trên tập test

# Độ chính xác trên tập train
print(f'Độ chính xác: (Trên tập huấn luyện) {accuracy_score(y_train, lr_model.
    ↪predict(X_train)):.2f}')
```

```
# Độ chính xác trên tập test
print(f'Độ chính xác: (Trên tập kiểm thử) {accuracy_score(y_test, y_pred):.2f}')
```

```
print()
# Confusion matrix để xem rõ hơn về việc dự đoán đúng sai của mô hình
print(confusion_matrix(y_test, y_pred))
```

Độ chính xác: (Trên tập huấn luyện) 1.00

Độ chính xác: (Trên tập kiểm thử) 0.93

```
[[61  6]
 [ 2 45]]
```

```
[ ]: from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
B	0.97	0.91	0.94	67
M	0.88	0.96	0.92	47
accuracy			0.93	114
macro avg	0.93	0.93	0.93	114
weighted avg	0.93	0.93	0.93	114

Nhìn qua những chỉ số đánh giá trên có thể dễ dàng nhận thấy chỉ sử dụng mô hình đơn giản Decision Tree Classifier vẫn cho kết quả dự đoán rất cao, tất cả chỉ số **precision**, **recall** và **f1-score** đều cao  $\geq 0.85$ . Độ chính xác trên tập test đạt 92%

Nhìn chung đây là tập dữ liệu chưa đủ lớn để có thể kết luận, em sẽ đánh giá mô hình này bằng K-Fold Cross Validation để đánh giá mô hình trên đầy đủ và chính xác hơn với tập dữ liệu nhỏ.

```
[ ]: from sklearn.model_selection import KFold
def eval_kfold(model, X, y, splits=3):
```

```

# Khởi tạo kfold
kfold = KFold(n_splits=splits)

fold_idx = 1
sum_acc = 0
for train_ids, val_ids in kfold.split(X, y):

    model = model
    # Train model
    model.fit(X.iloc[train_ids], y.iloc[train_ids])

    # Test và in kết quả
    scores = model.score(X.iloc[val_ids], y.iloc[val_ids])
    print(f'Độ chính xác mô hình trên tập train: {model.score(X.
↪iloc[train_ids], y.iloc[train_ids]):.2f}')
    print(f'Độ chính xác mô hình trên tập validation: {scores:.2f}')
    print("Đã train xong Fold ", fold_idx)
    sum_acc += scores
    # Sang Fold tiếp theo
    fold_idx = fold_idx + 1
    print(f'Độ chính xác trung bình sau {fold_idx-1} lần là: {sum_acc/
↪(fold_idx-1):.2f}')

```

```

[ ]: model = DecisionTreeClassifier()
eval_kfold(model, X_train, y_train)

```

```

Độ chính xác mô hình trên tập train: 1.00
Độ chính xác mô hình trên tập validation: 0.96
Đã train xong Fold 1
Độ chính xác mô hình trên tập train: 1.00
Độ chính xác mô hình trên tập validation: 0.93
Đã train xong Fold 2
Độ chính xác mô hình trên tập train: 1.00
Độ chính xác mô hình trên tập validation: 0.92
Đã train xong Fold 3
Độ chính xác trung bình sau 3 lần là: 0.94

```

Tiếp tục thử nghiệm với mô hình K-neighbors

```

[ ]: from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier()
knn_model.fit(X_train,y_train)

```

```

[ ]: KNeighborsClassifier()

```

```

[ ]: eval_kfold(knn_model,X,y)

```

```

Độ chính xác mô hình trên tập train: 0.95
Độ chính xác mô hình trên tập validation: 0.86

```



Đã train xong Fold 1  
 Độ chính xác mô hình trên tập train: 0.93  
 Độ chính xác mô hình trên tập validation: 0.92  
 Đã train xong Fold 2  
 Độ chính xác mô hình trên tập train: 0.93  
 Độ chính xác mô hình trên tập validation: 0.89  
 Đã train xong Fold 3  
 Độ chính xác trung bình sau 3 lần là: 0.89

Có vẻ như mô hình K neighbors cho ra kết quả còn thấp hơn cả decision tree, do đó, chúng ta sẽ tiếp tục thử nghiệm với mô hình Logistic

```
[ ]: from sklearn.linear_model import LogisticRegression
logit_model = LogisticRegression()
logit_model.fit(X_train,y_train)
```

```
C:\Users\nam04\anaconda3\envs\min_ds-env\lib\site-
packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max\_iter) or scale the data as shown in:  
<https://scikit-learn.org/stable/modules/preprocessing.html>  
 Please also refer to the documentation for alternative solver options:  
[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)  
 n\_iter\_i = \_check\_optimize\_result(

```
[ ]: LogisticRegression()
```

```
[ ]: eval_kfold(logit_model,X,y)
```

```
C:\Users\nam04\anaconda3\envs\min_ds-env\lib\site-
packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max\_iter) or scale the data as shown in:  
<https://scikit-learn.org/stable/modules/preprocessing.html>  
 Please also refer to the documentation for alternative solver options:  
[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)  
 n\_iter\_i = \_check\_optimize\_result(

Độ chính xác mô hình trên tập train: 0.96  
 Độ chính xác mô hình trên tập validation: 0.92  
 Đã train xong Fold 1  
 Độ chính xác mô hình trên tập train: 0.94  
 Độ chính xác mô hình trên tập validation: 0.97  
 Đã train xong Fold 2

Độ chính xác mô hình trên tập train: 0.96  
Độ chính xác mô hình trên tập validation: 0.94  
Đã train xong Fold 3  
Độ chính xác trung bình sau 3 lần là: 0.94

```
C:\Users\nam04\anaconda3\envs\min_ds-env\lib\site-  
packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed  
to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

```
C:\Users\nam04\anaconda3\envs\min_ds-env\lib\site-  
packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed  
to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

Có vẻ mô hình Logistic chạy khá tốt và đem lại kết quả khả quan hơn 2 mô hình trước. Em sẽ thử thêm 1 mô hình khác, em dùng Random Forest Classifier vì mô hình này tổng hợp nhiều Decision Tree nên thuật toán này có thể sẽ cho kết quả tốt hơn.

```
[ ]: from sklearn.ensemble import RandomForestClassifier  
# random forest model  
rf_model = RandomForestClassifier()  
  
rf_model.fit(X_train, y_train)
```

```
[ ]: RandomForestClassifier()
```

```
[ ]: eval_kfold(rf_model, X, y)
```

Độ chính xác mô hình trên tập train: 1.00  
Độ chính xác mô hình trên tập validation: 0.93  
Đã train xong Fold 1  
Độ chính xác mô hình trên tập train: 1.00  
Độ chính xác mô hình trên tập validation: 0.97  
Đã train xong Fold 2  
Độ chính xác mô hình trên tập train: 1.00

Độ chính xác mô hình trên tập validation: 0.96

Đã train xong Fold 3

Độ chính xác trung bình sau 3 lần là: 0.95

Độ chính xác trung bình đã tăng lên 95% và ở tập thứ 2 lên tới 98%, em nghĩ đây là mô hình có tiềm năng nên dùng GridSearch để tìm siêu tham số cho mô hình này.

Quá trình tìm những siêu tham số cho random forest có thể mất vài chục phút

```
[ ]: from sklearn.model_selection import GridSearchCV
n_estimators = [100, 120, 140, 160]
max_depth = [5, 10, 13]
min_samples_split = [2, 3, 4]
min_samples_leaf = [2, 3, 4]
criterion = ['gini', 'entropy']
rfc_model = RandomForestClassifier()
hyperF = dict(n_estimators = n_estimators, max_depth = max_depth,
              min_samples_split = min_samples_split,
              min_samples_leaf = min_samples_leaf,
              criterion = criterion)

gridF = GridSearchCV(rfc_model, hyperF, cv = 3, verbose = 10,
                    n_jobs = -1)
bestF = gridF.fit(X_train, y_train)
```

Fitting 3 folds for each of 216 candidates, totalling 648 fits

```
[ ]: best_rf = bestF.best_estimator_
bestF.best_params_
```

```
[ ]: {'criterion': 'entropy',
      'max_depth': 13,
      'min_samples_leaf': 2,
      'min_samples_split': 4,
      'n_estimators': 100}
```

```
[ ]: eval_kfold(best_rf, X, y)
```

Độ chính xác mô hình trên tập train: 1.00

Độ chính xác mô hình trên tập validation: 0.92

Đã train xong Fold 1

Độ chính xác mô hình trên tập train: 0.99

Độ chính xác mô hình trên tập validation: 0.97

Đã train xong Fold 2

Độ chính xác mô hình trên tập train: 1.00

Độ chính xác mô hình trên tập validation: 0.97

Đã train xong Fold 3

Độ chính xác trung bình sau 3 lần là: 0.95

- Nhận thấy khi sử dụng mô hình random forest sẽ cho được độ chính xác trung bình trên cả

3 tập cao hơn (95%).

- Từ kết quả trên, nhóm sẽ xây dựng ứng dụng trực quan hoá và chạy mô hình phân loại để dự đoán kết quả bằng mô hình random forest classifier