In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from numpy import asarray


print("Bibliothèques : OK")

df = pd.read_csv("../input/heart-failure-prediction/heart.csv")
df = df.dropna(axis=0)

#PRE-PROCESSING


#Remplace les valeurs de Sex en INT
df['Sex'].replace(['M','F'],[0,1], inplace = True)

#Remplace les valeurs de ChestPainType en INT
df['ChestPainType'].replace(['TA','ATA','NAP','ASY'],[0,1,2,3], inplace = True)

#Remplace les valeurs de RestingECG en INT
df['RestingECG'].replace(['Normal','ST','ST-T','LVH'],[0,1,2,3], inplace = True)

#Remplace les valeurs de ExerciseAngina en INT
df['ExerciseAngina'].replace(['N','Y'],[0,1], inplace = True)

#Remplace les valeurs de ST_Slope en INT
df['ST_Slope'].replace(['Up','Flat','Down'],[0,1,2], inplace = True)

#Separe mes features et ma target
y = df['HeartDisease']
X = df.drop('HeartDisease',axis=1)

#NORMALISATION - MinMaxScaler et Standard
X_minMax = MinMaxScaler().fit_transform(X)
X_standard = StandardScaler().fit_transform(X)
```

```
Bibliothèques : OK
```

Séparation du DataSet

In [2]:

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier


#Creation de mes tableaux d'entrainements et de tests
X_train,X_test,y_train,y_test = train_test_split(X_standard,y,test_size = 0.20, random_s
tate = 5)
```

CROSS-VALIDATION

In [3]:

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import validation_curve
```

GridSearchCV

In [4]:

```
from sklearn.model_selection import GridSearchCV
```

Matrice de Confusion

In [5]:

```
from sklearn.metrics import confusion_matrix
```

Learning Curve

In [6]:

```
from sklearn.model_selection import learning_curve
```

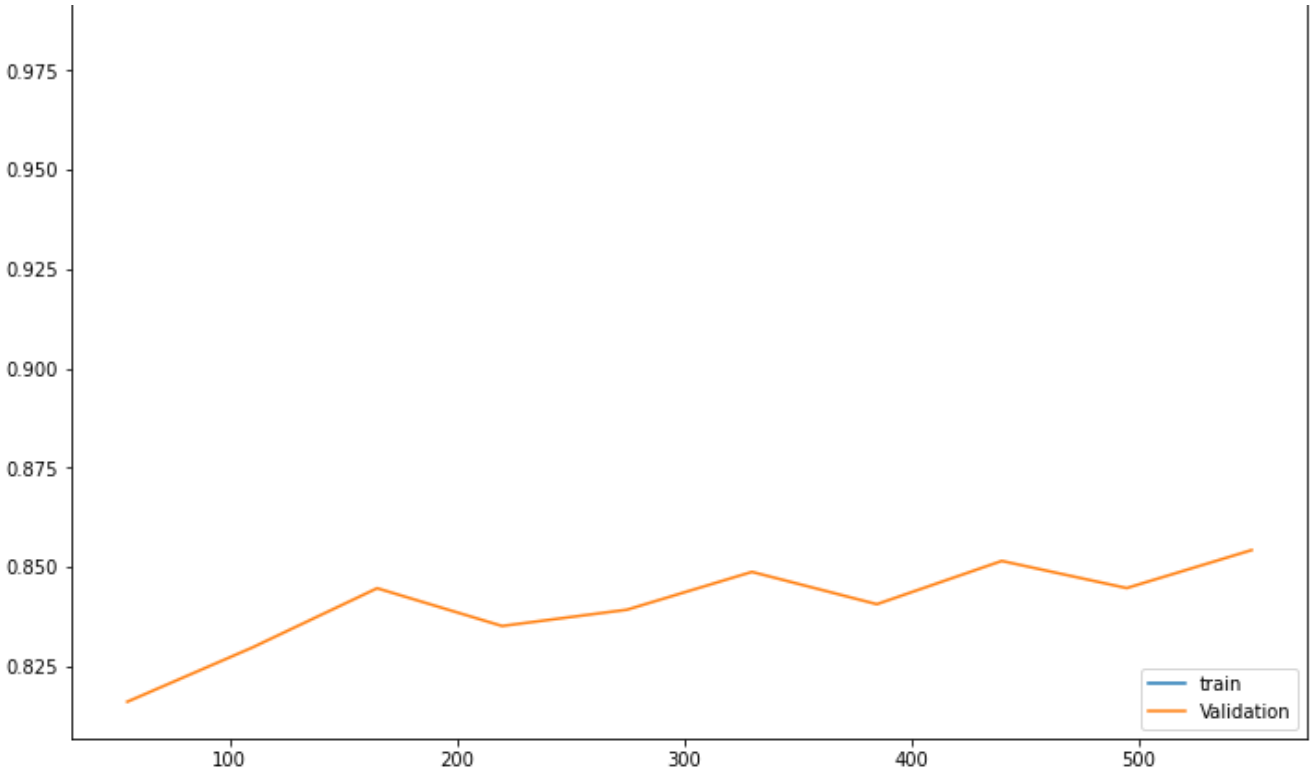
GENERALISATION D'EVALUATION

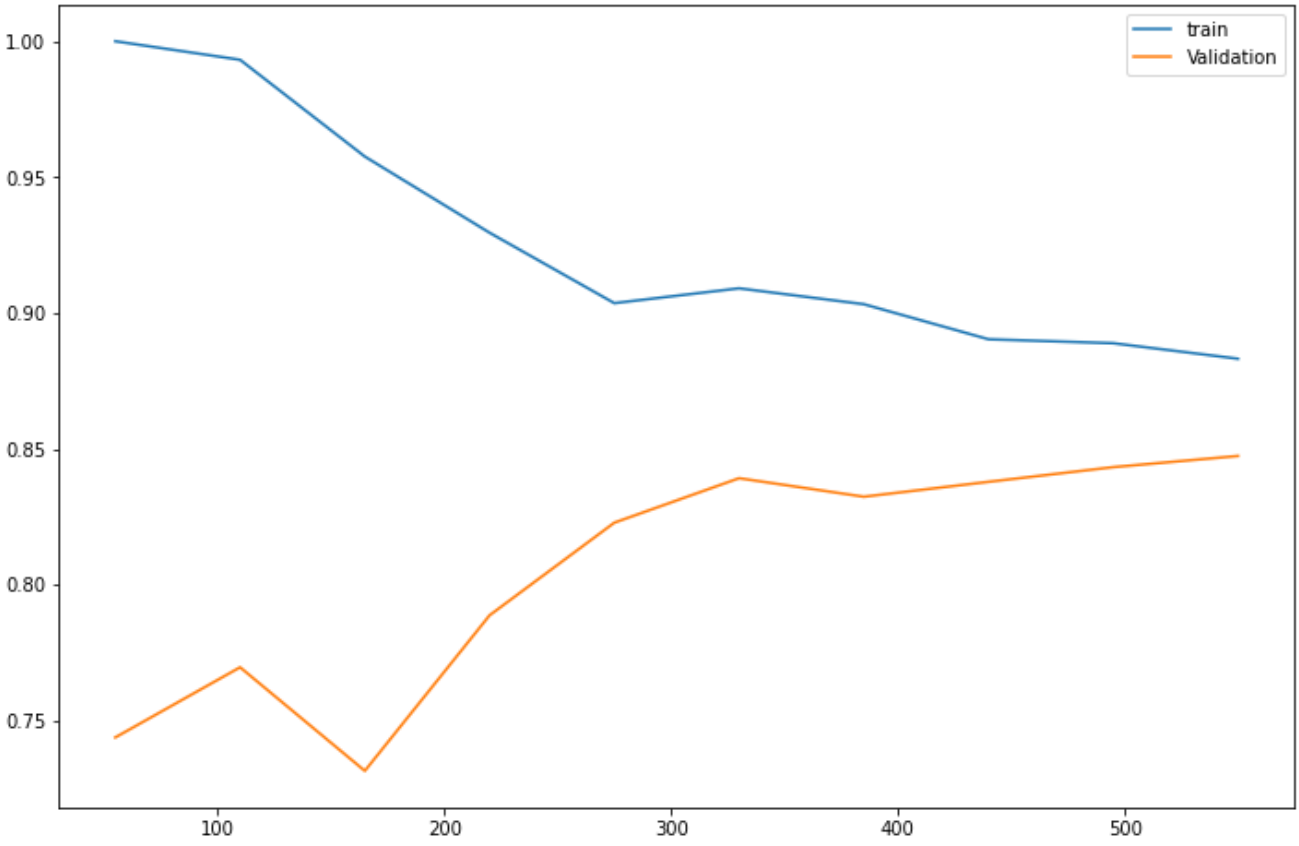
In [7]:

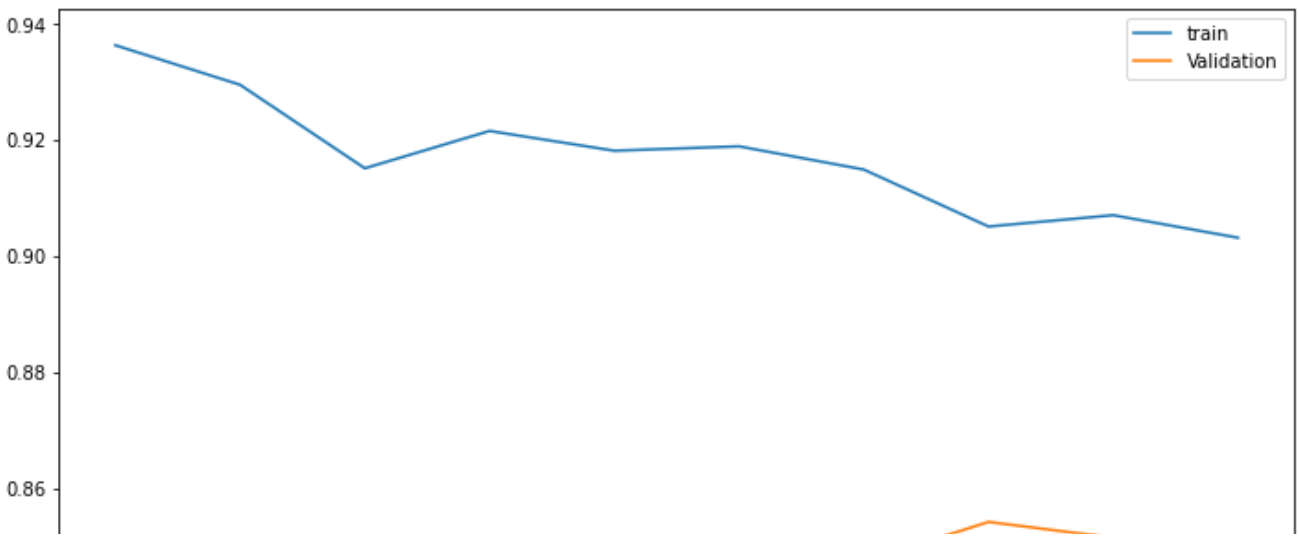
```
from sklearn.metrics import f1_score, classification_report

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoosting
Classifier, BaggingClassifier
from sklearn.svm import SVC

RandomForrest = RandomForestClassifier(random_state=0)
AdaBoost = AdaBoostClassifier(random_state=0)
SVM = SVC()
KNN = KNeighborsClassifier()
GB = GradientBoostingClassifier(random_state=0)
BC = BaggingClassifier(random_state=0)

dict_of_models = {'RandomForest' : RandomForrest,
                  'AdaBoost' : AdaBoost,
                  'SVM' : SVM,
                  'KNN' : KNN,
                  'GB' : GB,
                  'BC' : BC}


def evaluation(model):

    model.fit(X_train,y_train)
    yPred = model.predict(X_test)
    print(confusion_matrix(y_test,yPred))
    print(classification_report(y_test,yPred))

    N,train_score,val_score = learning_curve(model,X_train,y_train,train_sizes = np.lins
pace(0.1,1.0,10),cv=4,scoring='accuracy')
    plt.figure(figsize=(12,8))
    plt.plot(N,train_score.mean(axis=1), label='train')
    plt.plot(N,val_score.mean(axis=1), label='Validation')
    plt.legend()

for name,model in dict_of_models.items():
    print(name)
    evaluation(model)
#ok
```

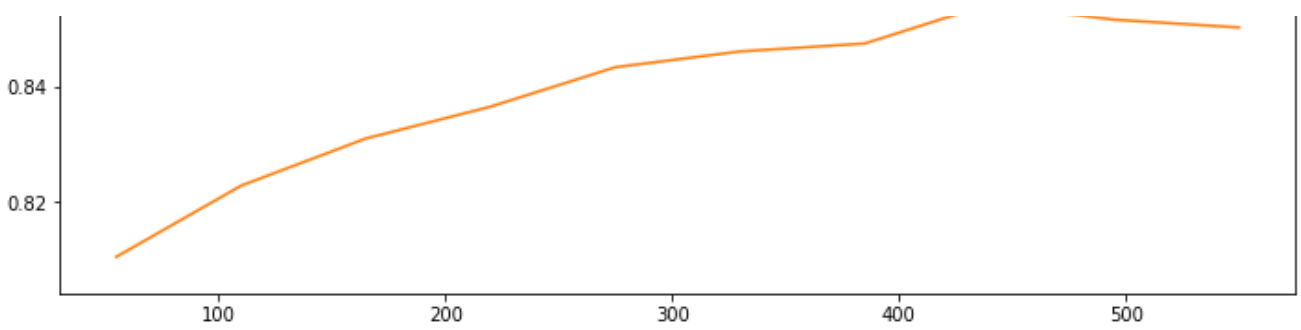
```
RandomForest
[[ 66   7]
 [  9 102]]
              precision    recall  f1-score   support
```

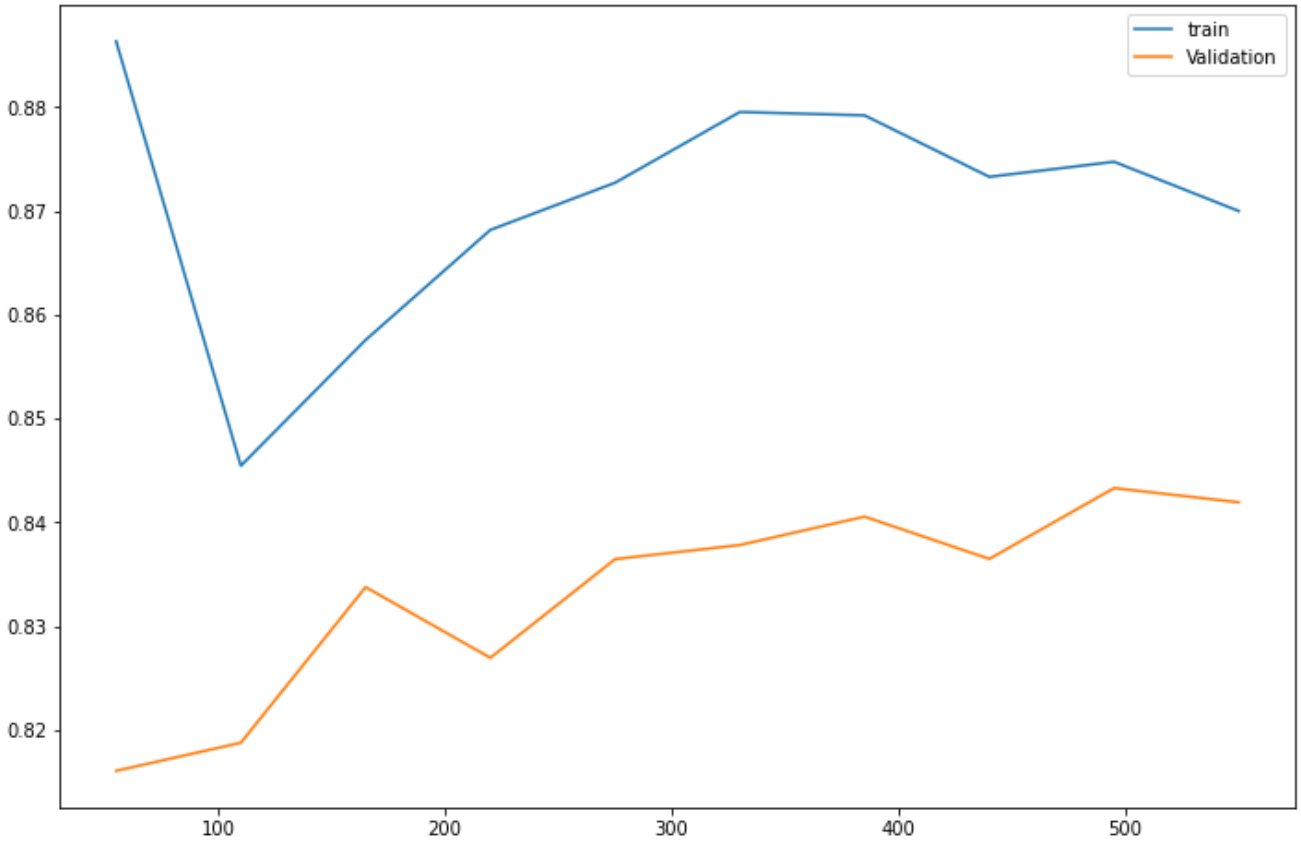
```
              precision    recall  f1-score   support

           0       0.88      0.90      0.89        73
           1       0.94      0.92      0.93       111

    accuracy                           0.91       184
   macro avg       0.91      0.91      0.91       184
weighted avg       0.91      0.91      0.91       184

AdaBoost
[[ 65   8]
 [ 11 100]]
              precision    recall  f1-score   support

           0       0.86      0.89      0.87        73
           1       0.93      0.90      0.91       111

    accuracy                           0.90       184
   macro avg       0.89      0.90      0.89       184
weighted avg       0.90      0.90      0.90       184

SVM
[[ 66   7]
 [  9 102]]
              precision    recall  f1-score   support

           0       0.88      0.90      0.89        73
           1       0.94      0.92      0.93       111

    accuracy                           0.91       184
   macro avg       0.91      0.91      0.91       184
weighted avg       0.91      0.91      0.91       184

KNN
[[ 67   6]
 [ 11 100]]
              precision    recall  f1-score   support

           0       0.86      0.92      0.89        73
           1       0.94      0.90      0.92       111

    accuracy                           0.91       184
   macro avg       0.90      0.91      0.90       184
weighted avg       0.91      0.91      0.91       184

GB
[[64  9]
 [12 99]]
              precision    recall  f1-score   support

           0       0.84      0.88      0.86        73
           1       0.92      0.89      0.90       111

    accuracy                           0.89       184
   macro avg       0.88      0.88      0.88       184
weighted avg       0.89      0.89      0.89       184

BC
[[66  7]
 [19 92]]
              precision    recall  f1-score   support

           0       0.78      0.90      0.84        73
           1       0.93      0.83      0.88       111

    accuracy                           0.86       184
   macro avg       0.85      0.87      0.86       184
weighted avg       0.87      0.86      0.86       184
```

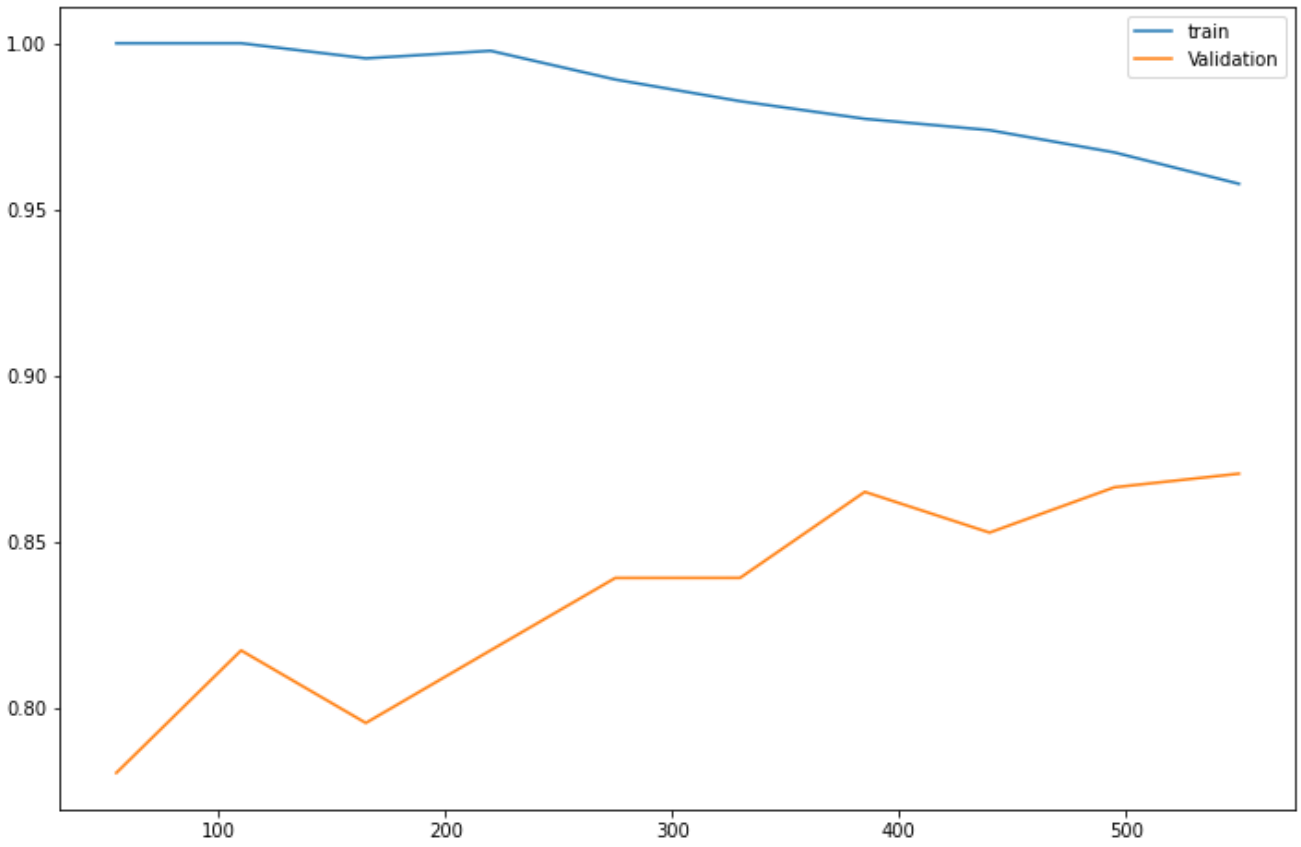

0.975
0.950
0.925
0.900
0.875
0.850
0.825
100
200
300
400
500
train
Validation

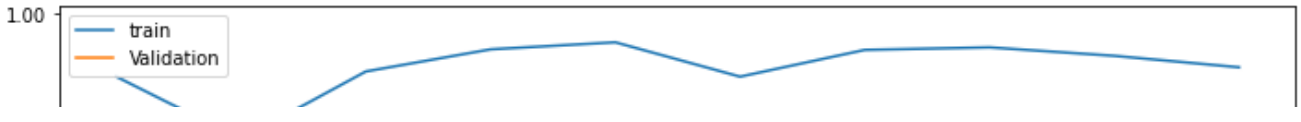

1.00
0.95
0.90
0.85
0.80
0.75
100
200
300
400
500
train
Validation

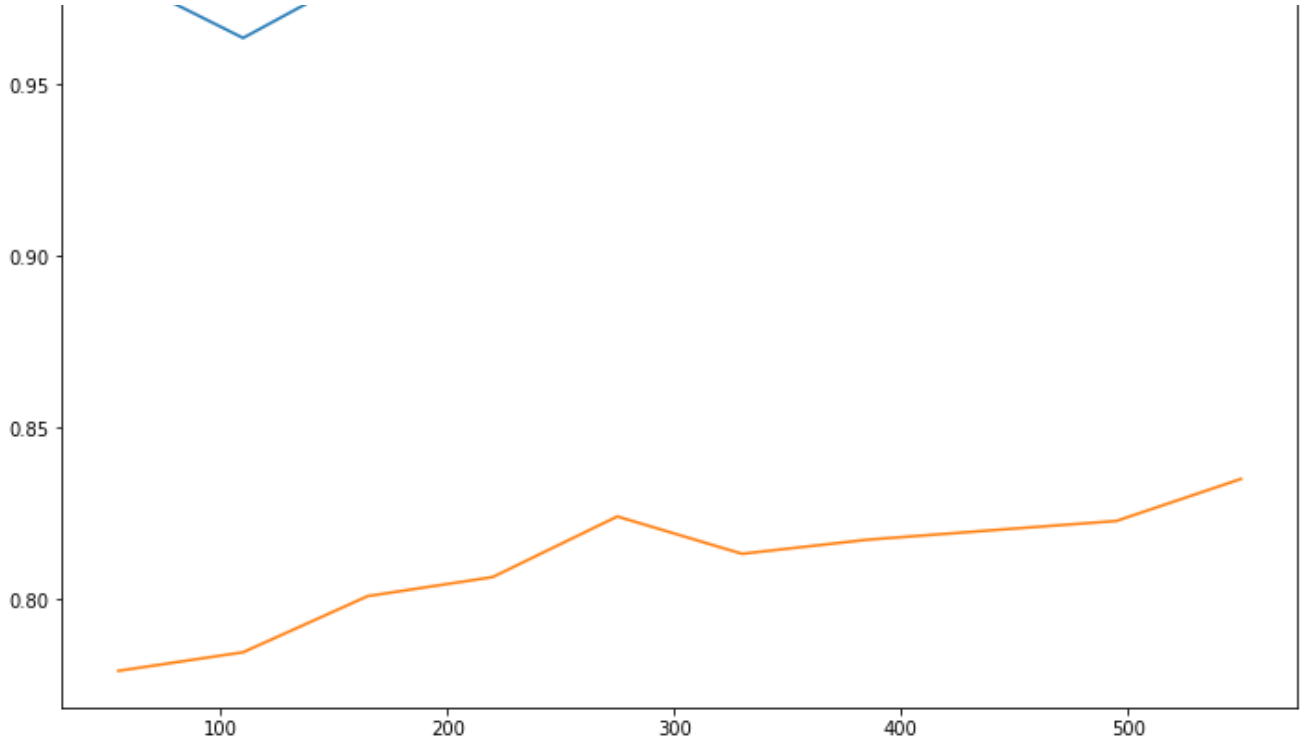

0.94
0.92
0.90
0.88
0.86
train
Validation


0.84
0.82
100
200
300
400
500


train
Validation
0.88
0.87
0.86
0.85
0.84
0.83
0.82
100
200
300
400
500


train
Validation
1.00
0.95
0.90
0.85
0.80
100
200
300
400
500


train
Validation
1.00



- **RandomForest : OverFitting**
- **AdaBoost : A exploiter**
- **SVC : A exploiter**
- **KNN : A oublier**
- **GB : A oublier**
- **BC : A oublier**

OPTIMISATION

In [8]:

```
#AdaBoost

hyper_params = {'n_estimators' : [1,10,50,100,500],
                'learning_rate' : [0.0001,0.001,0.01,0.1,1.0]}


grid = GridSearchCV(AdaBoost,hyper_params,scoring='accuracy',cv=4)

grid.fit(X_train,y_train)
print(grid.best_params_)

y_pred = grid.predict(X_test)
print(classification_report(y_test,y_pred))
```

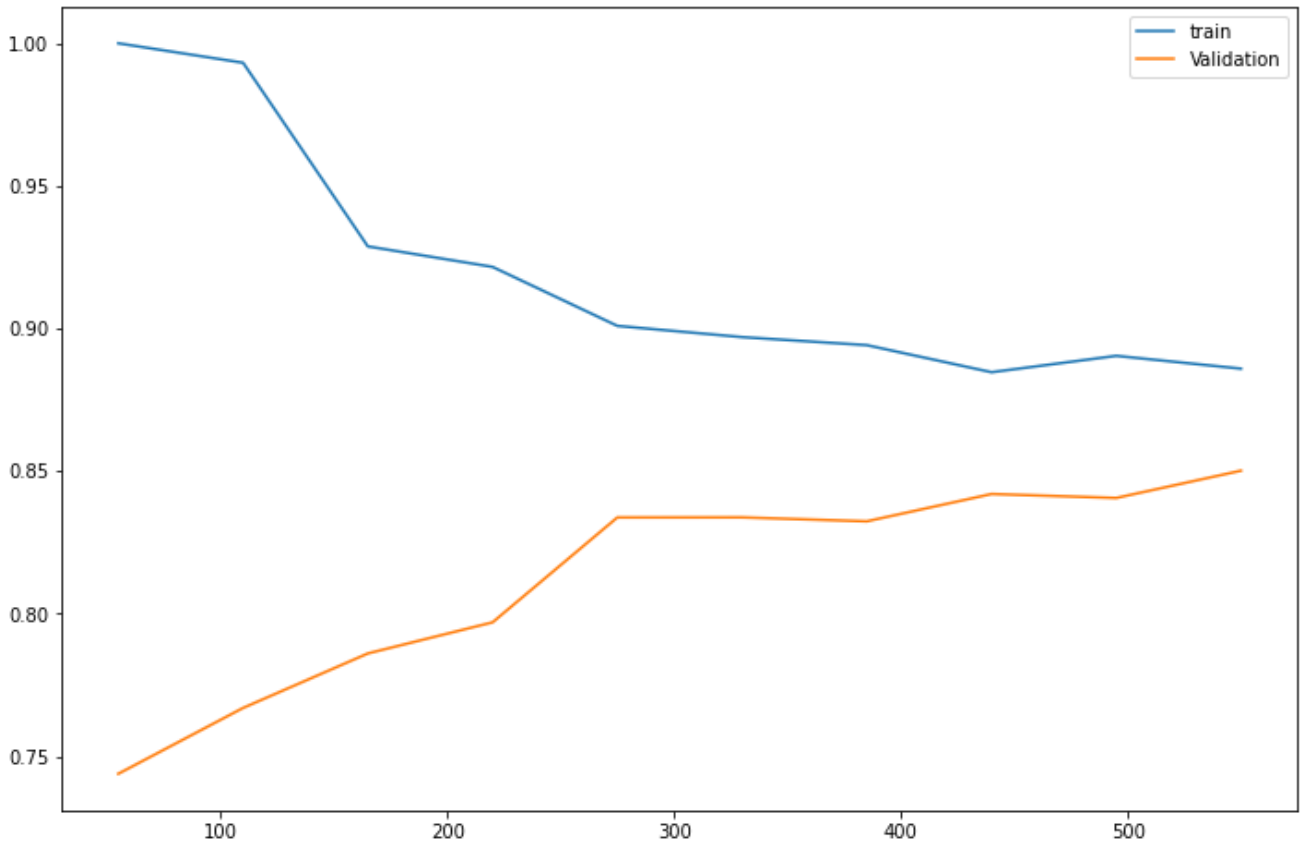
```
{'learning_rate': 0.1, 'n_estimators': 500}
              precision    recall  f1-score   support

           0       0.87      0.92      0.89        73
           1       0.94      0.91      0.93       111

    accuracy                           0.91       184
   macro avg       0.91      0.91      0.91       184
weighted avg       0.91      0.91      0.91       184
```

In [9]:

```
evaluation(grid.best_estimator_)
```

```
[[ 67   6]
 [ 10 101]]
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

           0       0.87      0.92      0.89        73
           1       0.94      0.91      0.93       111

    accuracy                           0.91       184
   macro avg       0.91      0.91      0.91       184
weighted avg       0.91      0.91      0.91       184
```



In [10]:

```
model = grid.best_estimator_

#On applique l'entrainement aux tests
print("Meilleur score TEST : ",model.score(X_test,y_test)," avec les paramètres : ", gri
d.best_estimator_)
```

```
Meilleur score TEST :  0.9130434782608695  avec les paramètres :  AdaBoostClassifier(lear
ning_rate=0.1, n_estimators=500, random_state=0)
```