**Heart Disease Data**

1. Import libraries

In [1]:

import numpy as np

import pandas as pd

import math

import random

import seaborn as sns

import pandas\_profiling as pp

import matplotlib.pyplot as plt

%matplotlib inline

*# Preprocessing*

import sklearn

from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler, RobustScaler

from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold, learning\_curve, ShuffleSplit

from sklearn.model\_selection import cross\_val\_predict as cvp

from sklearn import metrics

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, accuracy\_score, confusion\_matrix, explained\_variance\_score

*# Models*

from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn import metrics

import xgboost as xgb

from xgboost import XGBClassifier

import lightgbm as lgb

from lightgbm import LGBMClassifier

import warnings

warnings.filterwarnings("ignore")

2. Download datasets

In [2]:

cv\_n\_split = 5

random\_state = 42

test\_train\_split\_part = 0.25

In [3]:

metrics\_all = {1: 'acc', 2 : 'rmse', 3 : 're'}

metrics\_now = [1, 2, 3] *# you can only select some numbers of metrics from metrics\_all*

In [4]:

data = pd.read\_csv("/heart\_disease\_uci.csv")

data['target'] = data['num']

data = data.drop(columns=['id', 'dataset', 'ca', 'thal', 'num'])

In [5]:

data.head(3)

Out[5]:

|  | age | sex | cp | Trestbps | chol | fbs | restecg | thalch | exang | oldpeak | slope | target |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 63 | Male | typical angina | 145.0 | 233.0 | True | lv hypertrophy | 150.0 | False | 2.3 | downsloping | 0 |
| 1 | 67 | Male | asymptomatic | 160.0 | 286.0 | False | lv hypertrophy | 108.0 | True | 1.5 | flat | 2 |
| 2 | 67 | Male | asymptomatic | 120.0 | 229.0 | False | lv hypertrophy | 129.0 | True | 2.6 | flat | 1 |

In [6]:

data = data[(data['chol'] <= 420) & (data['oldpeak'] >=0) & (data['oldpeak'] <=4)].reset\_index(drop=True)

data = data.dropna().reset\_index(drop=True)

data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 520 entries, 0 to 519

Data columns (total 12 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 age 520 non-null int64

1 sex 520 non-null object

2 cp 520 non-null object

3 trestbps 520 non-null float64

4 chol 520 non-null float64

5 fbs 520 non-null object

6 restecg 520 non-null object

7 thalch 520 non-null float64

8 exang 520 non-null object

9 oldpeak 520 non-null float64

10 slope 520 non-null object

11 target 520 non-null int64

dtypes: float64(4), int64(2), object(6)

memory usage: 48.9+ KB

In [7]:

data.describe()

Out[7]:

|  | age | trestbps | chol | thalch | oldpeak | target |
| --- | --- | --- | --- | --- | --- | --- |
| count | 520.000000 | 520.000000 | 520.000000 | 520.000000 | 520.000000 | 520.000000 |
| mean | 54.780769 | 133.365385 | 215.436538 | 138.744231 | 1.183269 | 1.121154 |
| std | 8.873442 | 18.971664 | 95.672469 | 25.792375 | 1.031624 | 1.175675 |
| min | 29.000000 | 0.000000 | 0.000000 | 60.000000 | 0.000000 | 0.000000 |
| 25% | 48.750000 | 120.000000 | 197.000000 | 120.000000 | 0.100000 | 0.000000 |
| 50% | 56.000000 | 130.000000 | 233.000000 | 140.000000 | 1.000000 | 1.000000 |
| 75% | 61.000000 | 142.000000 | 271.250000 | 159.000000 | 2.000000 | 2.000000 |
| max | 77.000000 | 200.000000 | 417.000000 | 202.000000 | 4.000000 | 4.000000 |

In [8]:

data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 520 entries, 0 to 519

Data columns (total 12 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 age 520 non-null int64

1 sex 520 non-null object

2 cp 520 non-null object

3 trestbps 520 non-null float64

4 chol 520 non-null float64

5 fbs 520 non-null object

6 restecg 520 non-null object

7 thalch 520 non-null float64

8 exang 520 non-null object

9 oldpeak 520 non-null float64

10 slope 520 non-null object

11 target 520 non-null int64

dtypes: float64(4), int64(2), object(6)

memory usage: 48.9+ KB

3. EDA & FE

3.1. FE In [9]:

def str\_features\_to\_numeric(data):

*# Transforms all string features of the df to numeric features*

*# Determination categorical features*

categorical\_columns = []

numerics = ['int8', 'int16', 'int32', 'int64', 'float16', 'float32', 'float64']

features = data.columns.values.tolist()

for col **in** features:

if data[col].dtype **in** numerics: continue

categorical\_columns.append(col)

*# Encoding categorical features*

for col **in** categorical\_columns:

if col **in** data.columns:

le = LabelEncoder()

le.fit(list(data[col].astype(str).values))

data[col] = le.transform(list(data[col].astype(str).values))

return data

In [10]:

*# Transform all string features of the df to numeric features*

data = str\_features\_to\_numeric(data)

data

Out[10]:

|  | age | sex | cp | trestbps | chol | fbs | restecg | thalch | exang | oldpeak | slope | target |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 63 | 1 | 3 | 145.0 | 233.0 | 1 | 0 | 150.0 | 0 | 2.3 | 0 | 0 |
| 1 | 67 | 1 | 0 | 160.0 | 286.0 | 0 | 0 | 108.0 | 1 | 1.5 | 1 | 2 |
| 2 | 67 | 1 | 0 | 120.0 | 229.0 | 0 | 0 | 129.0 | 1 | 2.6 | 1 | 1 |
| 3 | 37 | 1 | 2 | 130.0 | 250.0 | 0 | 1 | 187.0 | 0 | 3.5 | 0 | 0 |
| 4 | 41 | 0 | 1 | 130.0 | 204.0 | 0 | 0 | 172.0 | 0 | 1.4 | 2 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 515 | 57 | 1 | 0 | 130.0 | 207.0 | 0 | 2 | 96.0 | 1 | 1.0 | 1 | 0 |
| 516 | 74 | 1 | 0 | 155.0 | 310.0 | 0 | 1 | 112.0 | 1 | 1.5 | 0 | 2 |
| 517 | 51 | 0 | 0 | 114.0 | 258.0 | 1 | 0 | 96.0 | 0 | 1.0 | 2 | 0 |
| 518 | 62 | 1 | 0 | 160.0 | 254.0 | 1 | 2 | 108.0 | 1 | 3.0 | 1 | 4 |
| 519 | 53 | 1 | 0 | 144.0 | 300.0 | 1 | 2 | 128.0 | 1 | 1.5 | 1 | 3 |

520 rows × 12 columns

In [11]:

data.target.value\_counts()

Out[11]:

0 203

1 159

2 70

3 68

4 20

Name: target, dtype: int64

In [12]:

*# target = 0 or 1 ==> more data*

data = data[data['target'].isin([0, 1])]

data

Out[12]:

|  | age | sex | cp | trestbps | chol | fbs | restecg | thalch | exang | oldpeak | slope | target |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 63 | 1 | 3 | 145.0 | 233.0 | 1 | 0 | 150.0 | 0 | 2.3 | 0 | 0 |
| 2 | 67 | 1 | 0 | 120.0 | 229.0 | 0 | 0 | 129.0 | 1 | 2.6 | 1 | 1 |
| 3 | 37 | 1 | 2 | 130.0 | 250.0 | 0 | 1 | 187.0 | 0 | 3.5 | 0 | 0 |
| 4 | 41 | 0 | 1 | 130.0 | 204.0 | 0 | 0 | 172.0 | 0 | 1.4 | 2 | 0 |
| 5 | 56 | 1 | 1 | 120.0 | 236.0 | 0 | 1 | 178.0 | 0 | 0.8 | 2 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 512 | 60 | 1 | 0 | 130.0 | 186.0 | 1 | 0 | 140.0 | 1 | 0.5 | 1 | 1 |
| 513 | 55 | 1 | 0 | 120.0 | 226.0 | 0 | 0 | 127.0 | 1 | 1.7 | 0 | 1 |
| 514 | 56 | 1 | 0 | 130.0 | 203.0 | 1 | 1 | 98.0 | 0 | 1.5 | 1 | 1 |
| 515 | 57 | 1 | 0 | 130.0 | 207.0 | 0 | 2 | 96.0 | 1 | 1.0 | 1 | 0 |
| 517 | 51 | 0 | 0 | 114.0 | 258.0 | 1 | 0 | 96.0 | 0 | 1.0 | 2 | 0 |

362 rows × 12 columns

In [13]:

def fe\_creation(df):

df['age2'] = df['age']//10

df['trestbps2'] = df['trestbps']//10

df['chol2'] = df['chol']//60

df['thalch2'] = df['thalch']//40

df['oldpeak2'] = df['oldpeak']//0.4

for i **in** ['sex', 'age2', 'fbs', 'restecg', 'exang']:

for j **in** ['cp','trestbps2', 'chol2', 'thalch2', 'oldpeak2', 'slope']:

df[i + "\_" + j] = df[i].astype('str') + "\_" + df[j].astype('str')

return df

data = fe\_creation(data)

In [14]:

*# Transform all string features of the df to numeric features*

pd.set\_option('max\_columns', len(data.columns)+1)

data = str\_features\_to\_numeric(data)

data.head(3)

Out[14]:

|  | age | sex | cp | trestbps | chol | fbs | restecg | thalch | exang | oldpeak | slope | target | age2 | trestbps2 | chol2 | thalch2 | oldpeak2 | sex\_cp | sex\_trestbps2 | sex\_chol2 | sex\_thalch2 | sex\_oldpeak2 | sex\_slope | age2\_cp | age2\_trestbps2 | age2\_chol2 | age2\_thalch2 | age2\_oldpeak2 | age2\_slope | fbs\_cp | fbs\_trestbps2 | fbs\_chol2 | fbs\_thalch2 | fbs\_oldpeak2 | fbs\_slope | restecg\_cp | restecg\_trestbps2 | restecg\_chol2 | restecg\_thalch2 | restecg\_oldpeak2 | restecg\_slope | exang\_cp | exang\_trestbps2 | exang\_chol2 | exang\_thalch2 | exang\_oldpeak2 | exang\_slope |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 63 | 1 | 3 | 145.0 | 233.0 | 1 | 0 | 150.0 | 0 | 2.3 | 0 | 0 | 6 | 14.0 | 3.0 | 3.0 | 5.0 | 7 | 14 | 9 | 4 | 11 | 3 | 16 | 30 | 22 | 11 | 30 | 10 | 7 | 16 | 9 | 5 | 15 | 3 | 3 | 4 | 2 | 1 | 5 | 0 | 3 | 4 | 2 | 1 | 5 | 0 |
| 2 | 67 | 1 | 0 | 120.0 | 229.0 | 0 | 0 | 129.0 | 1 | 2.6 | 1 | 1 | 6 | 12.0 | 3.0 | 3.0 | 6.0 | 4 | 12 | 9 | 4 | 12 | 4 | 13 | 28 | 22 | 11 | 31 | 11 | 0 | 2 | 3 | 1 | 6 | 1 | 0 | 2 | 2 | 1 | 6 | 1 | 4 | 12 | 9 | 5 | 15 | 4 |
| 3 | 37 | 1 | 2 | 130.0 | 250.0 | 0 | 1 | 187.0 | 0 | 3.5 | 0 | 0 | 3 | 13.0 | 4.0 | 4.0 | 8.0 | 6 | 13 | 10 | 5 | 14 | 3 | 3 | 4 | 5 | 3 | 8 | 1 | 2 | 3 | 4 | 2 | 8 | 0 | 6 | 12 | 10 | 6 | 15 | 3 | 2 | 3 | 3 | 2 | 8 | 0 |

In [15]:

*# features\_best = ['trestbps', 'thalch', 'sex\_oldpeak2', 'age', 'cp', 'chol', 'sex\_chol2',*

*# 'exang\_chol2', 'exang\_oldpeak2', 'slope', 'sex\_trestbps2', 'age2\_cp',*

*# 'exang', 'oldpeak', 'age2\_trestbps2', 'fbs\_cp', 'fbs\_oldpeak2',*

*# 'fbs\_slope', 'restecg\_oldpeak2', 'exang\_cp', 'sex', 'thalch2',*

*# 'sex\_cp', 'fbs\_trestbps2', 'fbs\_thalch2', 'target']*

*# data = data[features\_best]*

*# data*

In [16]:

data.shape

Out[16]:

(362, 47)

3.2. EDA

In [17]:

features\_best = data.columns.tolist()

pd.set\_option('max\_columns', len(features\_best)+1)

Pandas Profiling

In [18]:

*#pp.ProfileReport(data[features\_best])*

Pandas Describe

In [19]:

data.describe()

Out[19]:

|  | age | sex | cp | trestbps | chol | fbs | restecg | thalch | exang | oldpeak | slope | target | age2 | trestbps2 | chol2 | thalch2 | oldpeak2 | sex\_cp | sex\_trestbps2 | sex\_chol2 | sex\_thalch2 | sex\_oldpeak2 | sex\_slope | age2\_cp | age2\_trestbps2 | age2\_chol2 | age2\_thalch2 | age2\_oldpeak2 | age2\_slope | fbs\_cp | fbs\_trestbps2 | fbs\_chol2 | fbs\_thalch2 | fbs\_oldpeak2 | fbs\_slope | restecg\_cp | restecg\_trestbps2 | restecg\_chol2 | restecg\_thalch2 | restecg\_oldpeak2 | restecg\_slope | exang\_cp | exang\_trestbps2 | exang\_chol2 | exang\_thalch2 | exang\_oldpeak2 | exang\_slope |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| count | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 | 362.000000 |
| mean | 53.049724 | 0.698895 | 0.895028 | 132.085635 | 230.179558 | 0.132597 | 0.812155 | 143.469613 | 0.419890 | 0.984254 | 1.356354 | 0.439227 | 4.850829 | 13.016575 | 3.381215 | 3.140884 | 2.058011 | 3.690608 | 10.132597 | 7.279006 | 3.237569 | 6.248619 | 3.453039 | 9.303867 | 17.342541 | 15.262431 | 7.690608 | 17.397790 | 7.908840 | 1.425414 | 4.740331 | 4.185083 | 1.671271 | 3.372928 | 1.754144 | 4.143646 | 10.541436 | 7.991713 | 4.287293 | 8.497238 | 3.792818 | 2.574586 | 7.328729 | 5.345304 | 2.820442 | 5.834254 | 2.616022 |
| std | 8.803976 | 0.459373 | 1.001399 | 17.740539 | 78.592559 | 0.339608 | 0.598304 | 25.301187 | 0.494224 | 0.898136 | 0.574216 | 0.496980 | 0.917580 | 1.835251 | 1.247038 | 0.693889 | 2.005382 | 1.998163 | 5.054724 | 3.206132 | 1.478909 | 3.745218 | 1.423683 | 3.787490 | 8.655539 | 5.932228 | 2.660550 | 7.583829 | 2.803832 | 1.777426 | 4.612278 | 2.408779 | 1.512534 | 4.006454 | 1.117871 | 2.402982 | 6.168879 | 3.991325 | 2.176506 | 5.433174 | 1.799863 | 1.764914 | 5.621457 | 3.605873 | 1.765575 | 5.475722 | 1.376119 |
| min | 29.000000 | 0.000000 | 0.000000 | 92.000000 | 0.000000 | 0.000000 | 0.000000 | 82.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 2.000000 | 9.000000 | 0.000000 | 2.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 47.000000 | 0.000000 | 0.000000 | 120.000000 | 204.000000 | 0.000000 | 0.000000 | 124.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 4.000000 | 12.000000 | 3.000000 | 3.000000 | 0.000000 | 2.000000 | 5.000000 | 4.000000 | 2.000000 | 3.000000 | 2.000000 | 6.000000 | 10.000000 | 10.000000 | 6.000000 | 11.250000 | 6.000000 | 0.000000 | 2.000000 | 3.000000 | 1.000000 | 0.000000 | 1.000000 | 2.000000 | 4.250000 | 3.250000 | 2.000000 | 4.000000 | 2.000000 | 1.000000 | 2.000000 | 2.000000 | 1.000000 | 0.000000 | 2.000000 |
| 50% | 54.000000 | 1.000000 | 1.000000 | 130.000000 | 235.500000 | 0.000000 | 1.000000 | 147.000000 | 0.000000 | 1.000000 | 1.000000 | 0.000000 | 5.000000 | 13.000000 | 3.000000 | 3.000000 | 2.000000 | 4.000000 | 12.000000 | 9.000000 | 4.000000 | 6.000000 | 4.000000 | 9.000000 | 17.000000 | 16.000000 | 8.000000 | 17.000000 | 8.000000 | 1.000000 | 3.000000 | 4.000000 | 1.000000 | 2.000000 | 1.000000 | 4.000000 | 11.000000 | 9.000000 | 5.000000 | 9.000000 | 4.000000 | 2.000000 | 5.000000 | 3.500000 | 2.000000 | 4.000000 | 2.000000 |
| 75% | 59.000000 | 1.000000 | 2.000000 | 140.000000 | 273.000000 | 0.000000 | 1.000000 | 162.000000 | 1.000000 | 1.500000 | 2.000000 | 1.000000 | 5.000000 | 14.000000 | 4.000000 | 4.000000 | 3.000000 | 5.000000 | 14.000000 | 10.000000 | 4.000000 | 9.000000 | 5.000000 | 12.000000 | 21.000000 | 18.000000 | 9.000000 | 22.000000 | 9.000000 | 2.000000 | 5.000000 | 5.000000 | 2.000000 | 4.000000 | 2.000000 | 6.000000 | 13.000000 | 10.000000 | 6.000000 | 12.000000 | 5.000000 | 4.000000 | 13.000000 | 9.000000 | 5.000000 | 11.000000 | 4.000000 |
| max | 76.000000 | 1.000000 | 3.000000 | 200.000000 | 417.000000 | 1.000000 | 2.000000 | 202.000000 | 1.000000 | 4.000000 | 2.000000 | 1.000000 | 7.000000 | 20.000000 | 6.000000 | 5.000000 | 9.000000 | 7.000000 | 21.000000 | 12.000000 | 6.000000 | 15.000000 | 5.000000 | 19.000000 | 41.000000 | 30.000000 | 15.000000 | 37.000000 | 15.000000 | 7.000000 | 19.000000 | 12.000000 | 6.000000 | 16.000000 | 5.000000 | 10.000000 | 28.000000 | 18.000000 | 9.000000 | 23.000000 | 8.000000 | 7.000000 | 20.000000 | 12.000000 | 6.000000 | 17.000000 | 5.000000 |

The analysis shows different patterns, but most importantly, it confirms that the features are quite diverse, there are no too strongly correlated. Some features clustering target values quite well, but there are none that do it with 100% accuracy. Those, a good dataset has been formed, but it is impossible to unambiguously choose the optimal model. There is little data, so any model can overfit.

4. Preparing to modeling

In [20]:

*# Target*

dataset = data.copy() *# original data*

target\_name = 'target'

target = data.pop(target\_name)

In [21]:

*# Model standartization*

scaler = StandardScaler()

*#scaler = RobustScaler()*

data = pd.DataFrame(scaler.fit\_transform(data), columns = data.columns)

In [22]:

*# Synthesis valid as test for selection models*

train, valid, train\_target, valid\_target = train\_test\_split(data, target, test\_size=test\_train\_split\_part, random\_state=random\_state)

In [23]:

train

Out[23]:

|  | age | sex | cp | trestbps | chol | fbs | restecg | thalch | exang | oldpeak | slope | age2 | trestbps2 | chol2 | thalch2 | oldpeak2 | sex\_cp | sex\_trestbps2 | sex\_chol2 | sex\_thalch2 | sex\_oldpeak2 | sex\_slope | age2\_cp | age2\_trestbps2 | age2\_chol2 | age2\_thalch2 | age2\_oldpeak2 | age2\_slope | fbs\_cp | fbs\_trestbps2 | fbs\_chol2 | fbs\_thalch2 | fbs\_oldpeak2 | fbs\_slope | restecg\_cp | restecg\_trestbps2 | restecg\_chol2 | restecg\_thalch2 | restecg\_oldpeak2 | restecg\_slope | exang\_cp | exang\_trestbps2 | exang\_chol2 | exang\_thalch2 | exang\_oldpeak2 | exang\_slope |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 60 | -0.915594 | 0.656376 | -0.895014 | -1.585324 | -0.282600 | -0.390981 | -1.359306 | 0.179306 | 1.175406 | 2.247471 | -0.621450 | -0.928537 | -1.645960 | -0.30612 | -0.203316 | 2.467774 | 0.155053 | -0.026269 | 0.537525 | 0.51625 | 1.805162 | 0.384719 | -1.137910 | -1.196558 | -1.057124 | -1.012697 | -0.448651 | -1.038888 | -0.803064 | -1.029186 | -0.492666 | -0.444420 | 0.906560 | -0.675558 | -1.726763 | -1.711174 | -1.503261 | -1.512444 | -0.275955 | -1.553832 | 0.808758 | 0.475850 | 1.014943 | 1.236184 | 1.859082 | 1.007103 |
| 195 | 0.904282 | 0.656376 | -0.895014 | 0.446735 | -0.295341 | -0.390981 | -1.359306 | -0.216479 | 1.175406 | 1.021018 | 1.122465 | 1.254128 | 0.536595 | -0.30612 | -0.203316 | 0.969729 | 0.155053 | 0.766166 | 0.537525 | 0.51625 | 1.003032 | 1.088095 | 0.977230 | 1.464378 | 1.137329 | 1.245597 | 1.531979 | 1.461151 | -0.803064 | -0.160735 | -0.492666 | -0.444420 | 0.156732 | 0.220237 | -1.726763 | -1.061861 | -1.503261 | -1.512444 | -0.828882 | -0.997465 | 0.808758 | 1.188394 | 1.014943 | 1.236184 | 1.310451 | 1.734790 |
| 302 | -0.688109 | 0.656376 | -0.895014 | 1.575657 | 0.774941 | -0.390981 | 1.988101 | 0.575092 | 1.175406 | 2.247471 | -0.621450 | -0.928537 | 1.627873 | 0.49689 | -0.203316 | 2.467774 | 0.155053 | 1.162383 | 0.849859 | 0.51625 | 1.805162 | 0.384719 | -1.137910 | -0.502401 | -0.888320 | -1.012697 | -0.448651 | -1.038888 | -0.803064 | 0.273490 | -0.076943 | -0.444420 | 0.906560 | -0.675558 | 1.607041 | 2.347035 | 2.009200 | 1.708172 | 2.488684 | 1.784370 | 0.808758 | 1.544666 | 1.292652 | 1.236184 | 1.859082 | 1.007103 |
| 132 | -0.005656 | 0.656376 | 1.104956 | -0.117726 | 0.201576 | 2.557668 | -1.359306 | 1.168770 | -0.850770 | -1.097402 | 1.122465 | 0.162795 | -0.009044 | 0.49689 | 1.239831 | -1.027664 | 1.157358 | 0.568057 | 0.849859 | 1.19336 | -0.066475 | 1.088095 | 0.448445 | -0.039630 | 0.293309 | 0.492832 | -0.184567 | 0.389706 | 2.577277 | 2.227504 | 2.417393 | 2.865866 | 1.656388 | 2.907623 | -0.893312 | -1.224189 | -1.252371 | -1.052356 | -1.566119 | -0.997465 | -0.326011 | -0.771103 | -0.651312 | -0.465332 | -1.066951 | -0.448271 |
| 220 | -1.029336 | 0.656376 | 0.104971 | -0.682187 | -0.588395 | -0.390981 | 0.314397 | -0.058165 | -0.850770 | 0.017556 | -0.621450 | -0.928537 | -0.554683 | -0.30612 | -0.203316 | -0.028968 | 0.656205 | 0.369949 | 0.537525 | 0.51625 | 0.468279 | 0.384719 | -0.873518 | -0.965173 | -1.057124 | -1.012697 | -0.844777 | -1.038888 | -0.239674 | -0.594961 | -0.492666 | -0.444420 | -0.343153 | -0.675558 | 0.356864 | 0.074438 | 0.252969 | 0.327908 | 0.276973 | 0.115269 | -0.893395 | -0.949239 | -0.929022 | -1.032503 | -0.701197 | -1.175958 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 71 | -1.029336 | -1.523516 | 1.104956 | -1.359540 | -1.136278 | -0.390981 | 0.314397 | 1.247927 | -0.850770 | -0.428427 | -0.621450 | -0.928537 | -1.645960 | -1.10913 | 1.239831 | -0.528316 | -0.847252 | -2.007354 | -1.961147 | -0.83797 | -1.403358 | -1.725407 | -0.609125 | -1.196558 | -1.225928 | -0.636315 | -0.976819 | -1.038888 | 0.323716 | -1.029186 | -0.908389 | 0.217638 | -0.593096 | -0.675558 | 0.773590 | -0.250219 | 0.002079 | 0.787996 | 0.092664 | 0.115269 | -0.326011 | -1.305511 | -1.206731 | -0.465332 | -0.884074 | -1.175958 |
| 106 | -0.688109 | 0.656376 | 1.104956 | -1.359540 | 0.163351 | -0.390981 | 0.314397 | 0.337620 | -0.850770 | -1.097402 | 1.122465 | -0.928537 | -1.645960 | 0.49689 | -0.203316 | -1.027664 | 1.157358 | -0.026269 | 0.849859 | 0.51625 | -0.066475 | 1.088095 | -0.609125 | -1.196558 | -0.888320 | -1.012697 | -1.108861 | -0.681739 | 0.323716 | -1.029186 | -0.076943 | -0.444420 | -0.843039 | 0.220237 | 0.773590 | -0.250219 | 0.503859 | 0.327908 | -0.091645 | 0.671636 | -0.326011 | -1.305511 | -0.651312 | -1.032503 | -1.066951 | -0.448271 |
| 270 | -0.346882 | 0.656376 | -0.895014 | -0.117726 | 0.035937 | -0.390981 | 0.314397 | -0.889315 | 1.175406 | 1.132514 | -0.621450 | 0.162795 | -0.009044 | -0.30612 | -0.203316 | 0.969729 | 0.155053 | 0.568057 | 0.537525 | 0.51625 | 1.003032 | 0.384719 | -0.080340 | -0.039630 | 0.124505 | 0.116450 | 0.343601 | 0.032558 | -0.803064 | -0.377848 | -0.492666 | -0.444420 | 0.156732 | -0.675558 | -0.059861 | 0.236766 | 0.252969 | 0.327908 | 0.645591 | 0.115269 | 0.808758 | 1.010258 | 1.014943 | 1.236184 | 1.310451 | 1.007103 |
| 348 | 0.108087 | -1.523516 | -0.895014 | 0.333843 | 0.558336 | -0.390981 | 0.314397 | -1.522571 | 1.175406 | 0.575035 | -0.621450 | 0.162795 | -0.009044 | 0.49689 | -1.646464 | 0.470381 | -1.849557 | -1.413028 | -1.336479 | -2.19219 | -0.868605 | -1.725407 | -0.080340 | -0.039630 | 0.293309 | -0.259933 | 0.211559 | 0.032558 | -0.803064 | -0.377848 | -0.076943 | -1.106477 | -0.093211 | -0.675558 | -0.059861 | 0.236766 | 0.503859 | -0.132180 | 0.461282 | 0.115269 | 0.808758 | 1.010258 | 1.292652 | 0.669012 | 1.127574 | 1.007103 |
| 102 | 0.676798 | 0.656376 | 2.104940 | 2.140118 | 0.736717 | -0.390981 | -1.359306 | 0.614670 | -0.850770 | -0.874410 | -0.621450 | 0.162795 | 2.173512 | 0.49689 | -0.203316 | -1.027664 | 1.658510 | 1.360491 | 0.849859 | 0.51625 | -0.066475 | 0.384719 | 0.712837 | 0.423142 | 0.293309 | 0.116450 | -0.184567 | 0.032558 | 0.887106 | 0.490603 | -0.076943 | -0.444420 | -0.843039 | -0.675558 | -0.476587 | -0.574876 | -1.252371 | -1.512444 | -1.566119 | -1.553832 | 0.241373 | -0.058559 | -0.651312 | -1.032503 | -1.066951 | -1.175958 |

271 rows × 46 columns

In [24]:

valid

Out[24]:

|  | age | sex | cp | trestbps | chol | fbs | restecg | thalch | exang | oldpeak | slope | age2 | trestbps2 | chol2 | thalch2 | oldpeak2 | sex\_cp | sex\_trestbps2 | sex\_chol2 | sex\_thalch2 | sex\_oldpeak2 | sex\_slope | age2\_cp | age2\_trestbps2 | age2\_chol2 | age2\_thalch2 | age2\_oldpeak2 | age2\_slope | fbs\_cp | fbs\_trestbps2 | fbs\_chol2 | fbs\_thalch2 | fbs\_oldpeak2 | fbs\_slope | restecg\_cp | restecg\_trestbps2 | restecg\_chol2 | restecg\_thalch2 | restecg\_oldpeak2 | restecg\_slope | exang\_cp | exang\_trestbps2 | exang\_chol2 | exang\_thalch2 | exang\_oldpeak2 | exang\_slope |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 163 | 1.131767 | -1.523516 | -0.895014 | -1.359540 | 0.494629 | -0.390981 | 0.314397 | 1.010455 | 1.175406 | 0.909522 | -0.621450 | 1.254128 | -1.645960 | 0.496890 | 1.239831 | 0.969729 | -1.849557 | -2.007354 | -1.336479 | -0.83797 | -0.601228 | -1.725407 | 0.977230 | 1.001606 | 1.306133 | 1.621979 | 1.531979 | 1.104003 | -0.803064 | -1.029186 | -0.076943 | 0.217638 | 0.156732 | -0.675558 | -0.059861 | -0.250219 | 0.503859 | 0.787996 | 0.645591 | 0.115269 | 0.808758 | 0.475850 | 1.292652 | 1.803355 | 1.310451 | 1.007103 |
| 33 | 0.904282 | -1.523516 | -0.895014 | -0.117726 | 1.271858 | -0.390981 | -1.359306 | 1.010455 | -0.850770 | -1.097402 | 1.122465 | 1.254128 | -0.009044 | 1.299901 | 1.239831 | -1.027664 | -1.849557 | -1.413028 | -1.024145 | -0.83797 | -1.670735 | -1.022032 | 0.977230 | 1.348685 | 1.474937 | 1.621979 | 1.003811 | 1.461151 | -0.803064 | -0.377848 | 0.338780 | 0.217638 | -0.843039 | 0.220237 | -1.726763 | -1.224189 | -1.001481 | -1.052356 | -1.566119 | -0.997465 | -1.460779 | -0.771103 | -0.373603 | -0.465332 | -1.066951 | -0.448271 |
| 15 | -0.460625 | 0.656376 | 0.104971 | -0.117726 | 0.456405 | -0.390981 | 0.314397 | 1.089613 | -0.850770 | -0.428427 | 1.122465 | -0.928537 | -0.009044 | 0.496890 | 1.239831 | -0.528316 | 0.656205 | 0.568057 | 0.849859 | 1.19336 | 0.200902 | 1.088095 | -0.873518 | -0.849480 | -0.888320 | -0.636315 | -0.976819 | -0.681739 | -0.239674 | -0.377848 | -0.076943 | 0.217638 | -0.593096 | 0.220237 | 0.356864 | 0.236766 | 0.503859 | 0.787996 | 0.092664 | 0.671636 | -0.893395 | -0.771103 | -0.651312 | -0.465332 | -0.884074 | -0.448271 |
| 322 | 2.269190 | -1.523516 | 1.104956 | 1.575657 | -2.932824 | -0.390981 | 1.988101 | -0.889315 | -0.850770 | -1.097402 | 1.122465 | 2.345460 | 1.627873 | -2.715151 | -0.203316 | -1.027664 | -0.847252 | -0.818703 | -2.273481 | -1.51508 | -1.670735 | -1.022032 | 2.563585 | 2.736999 | 1.812545 | 2.374744 | 2.060147 | 2.532597 | 0.323716 | 0.273490 | -1.739834 | -0.444420 | -0.843039 | 0.220237 | 2.440492 | 2.347035 | 1.256530 | 1.708172 | 1.382828 | 2.340737 | -0.326011 | -0.236695 | -1.484440 | -1.032503 | -1.066951 | -0.448271 |
| 57 | 0.790540 | 0.656376 | -0.895014 | -0.399956 | 0.354473 | -0.390981 | -1.359306 | -0.097744 | 1.175406 | 2.024480 | -0.621450 | 1.254128 | -0.554683 | 0.496890 | -0.203316 | 1.968426 | 0.155053 | 0.369949 | 0.849859 | 0.51625 | 1.537785 | 0.384719 | 0.977230 | 1.232992 | 1.306133 | 1.245597 | 1.796063 | 1.104003 | -0.803064 | -0.594961 | -0.076943 | -0.444420 | 0.656617 | -0.675558 | -1.726763 | -1.386518 | -1.252371 | -1.512444 | -0.460264 | -1.553832 | 0.808758 | 0.832122 | 1.292652 | 1.236184 | 1.676205 | 1.007103 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 316 | 0.676798 | 0.656376 | -0.895014 | 0.164504 | -2.932824 | -0.390981 | 0.314397 | -1.126786 | 1.175406 | 0.017556 | -0.621450 | 0.162795 | -0.009044 | -2.715151 | -1.646464 | -0.028968 | 0.155053 | 0.568057 | -0.399477 | -0.16086 | 0.468279 | 0.384719 | -0.080340 | -0.039630 | -0.381908 | -0.259933 | 0.079517 | 0.032558 | -0.803064 | -0.377848 | -1.739834 | -1.106477 | -0.343153 | -0.675558 | -0.059861 | 0.236766 | -0.499701 | -0.132180 | 0.276973 | 0.115269 | 0.808758 | 1.010258 | 0.181815 | 0.669012 | 0.944697 | 1.007103 |
| 310 | -1.711790 | -1.523516 | -0.895014 | -1.246648 | -2.932824 | -0.390981 | 0.314397 | 0.495934 | -0.850770 | -1.097402 | -0.621450 | -2.019869 | -1.100321 | -2.715151 | -0.203316 | -1.027664 | -1.849557 | -1.809246 | -2.273481 | -1.51508 | -1.670735 | -1.725407 | -2.195480 | -1.775023 | -2.407556 | -2.141844 | -2.165197 | -2.110333 | -0.803064 | -0.812073 | -1.739834 | -0.444420 | -0.843039 | -0.675558 | -0.059861 | -0.087891 | -0.499701 | 0.327908 | -0.091645 | 0.115269 | -1.460779 | -1.127375 | -1.484440 | -1.032503 | -1.066951 | -1.175958 |
| 339 | 0.563056 | 0.656376 | -0.895014 | -0.682187 | -2.932824 | -0.390981 | -1.359306 | -1.482993 | 1.175406 | 0.575035 | -2.365366 | 0.162795 | -0.554683 | -2.715151 | -1.646464 | 0.470381 | 0.155053 | 0.369949 | -0.399477 | -0.16086 | 0.735655 | -0.318656 | -0.080340 | -0.155322 | -0.381908 | -0.259933 | 0.211559 | -0.324591 | -0.803064 | -0.594961 | -1.739834 | -1.106477 | -0.093211 | -1.571354 | -1.726763 | -1.386518 | -2.005042 | -1.972531 | -1.013192 | -2.110199 | 0.808758 | 0.832122 | 0.181815 | 0.669012 | 1.127574 | 0.279416 |
| 110 | -0.119398 | 0.656376 | 2.104940 | 1.124088 | 0.864132 | 2.557668 | 0.314397 | 1.366662 | -0.850770 | 0.240548 | -0.621450 | 0.162795 | 1.082234 | 0.496890 | 1.239831 | -0.028968 | 1.658510 | 0.964274 | 0.849859 | 1.19336 | 0.468279 | 0.384719 | 0.712837 | 0.191756 | 0.293309 | 0.492832 | 0.079517 | 0.032558 | 3.140667 | 2.661730 | 2.417393 | 2.865866 | 2.156274 | 2.011828 | 1.190316 | 0.561423 | 0.503859 | 0.787996 | 0.276973 | 0.115269 | 0.241373 | -0.414831 | -0.651312 | -0.465332 | -0.701197 | -1.175958 |
| 295 | 0.335571 | 0.656376 | -0.895014 | 1.011196 | -0.002288 | -0.390981 | 1.988101 | -0.770579 | 1.175406 | 0.575035 | -0.621450 | 0.162795 | 1.082234 | -0.306120 | -0.203316 | 0.470381 | 0.155053 | 0.964274 | 0.537525 | 0.51625 | 0.735655 | 0.384719 | -0.080340 | 0.191756 | 0.124505 | 0.116450 | 0.211559 | 0.032558 | -0.803064 | 0.056377 | -0.492666 | -0.444420 | -0.093211 | -0.675558 | 1.607041 | 2.184707 | 1.758310 | 1.708172 | 1.935756 | 1.784370 | 0.808758 | 1.366530 | 1.014943 | 1.236184 | 1.127574 | 1.007103 |

91 rows × 46 columns

In [25]:

*# list of accuracy of all model - amount of metrics\_now \* 2 (train & valid datasets)*

num\_models = 6

acc\_train = []

acc\_valid = []

acc\_all = np.empty((len(metrics\_now)\*2, 0)).tolist()

acc\_all

Out[25]:

[[], [], [], [], [], []]

In [26]:

acc\_all\_pred = np.empty((len(metrics\_now), 0)).tolist()

acc\_all\_pred

Out[26]:

[[], [], []]

In [27]:

*# Splitting train data for model tuning with cross-validation*

cv\_train = ShuffleSplit(n\_splits=cv\_n\_split, test\_size=test\_train\_split\_part, random\_state=random\_state)

In [28]:

def acc\_d(y\_meas, y\_pred):

*# Relative error between predicted y\_pred and measured y\_meas values*

return mean\_absolute\_error(y\_meas, y\_pred)\*len(y\_meas)/sum(abs(y\_meas))

def acc\_rmse(y\_meas, y\_pred):

*# RMSE between predicted y\_pred and measured y\_meas values*

return (mean\_squared\_error(y\_meas, y\_pred))\*\*0.5

In [29]:

def plot\_cm(train\_target, train\_target\_pred, valid\_target, valid\_target\_pred):

*# Building the confusion matrices*

def cm\_calc(y\_true, y\_pred):

cm = confusion\_matrix(y\_true, y\_pred, labels=np.unique(y\_true))

cm\_sum = np.sum(cm, axis=1, keepdims=True)

cm\_perc = cm / cm\_sum.astype(float) \* 100

annot = np.empty\_like(cm).astype(str)

nrows, ncols = cm.shape

for i **in** range(nrows):

for j **in** range(ncols):

c = cm[i, j]

p = cm\_perc[i, j]

if i == j:

s = cm\_sum[i]

annot[i, j] = '**%.1f%%\n%d**/**%d**' % (p, c, s)

elif c == 0:

annot[i, j] = ''

else:

annot[i, j] = '**%.1f%%\n%d**' % (p, c)

cm = pd.DataFrame(cm, index=np.unique(y\_true), columns=np.unique(y\_true))

cm.index.name = 'Actual'

cm.columns.name = 'Predicted'

return cm, annot

*# Building the confusion matrices*

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16, 6), sharex=True)

*# Training data*

ax = axes[0]

ax.set\_title("for training data")

cm0, annot0 = cm\_calc(train\_target, train\_target\_pred)

sns.heatmap(cm0, cmap= "YlGnBu", annot=annot0, fmt='', ax=ax)

*# Test data*

ax = axes[1]

ax.set\_title("for test (validation) data")

cm1, annot1 = cm\_calc(valid\_target, valid\_target\_pred)

sns.heatmap(cm1, cmap= "YlGnBu", annot=annot1, fmt='', ax=ax)

fig.suptitle('CONFUSION MATRICES')

plt.show()

In [30]:

def acc\_metrics\_calc(num, acc\_all, model, train, valid, train\_target, valid\_target):

*# The models selection stage*

*# Calculation of accuracy of model by different metrics*

ytrain = model.predict(train).astype(int)

yvalid = model.predict(valid).astype(int)

print('train\_target = ', train\_target[:5].values)

print('ytrain = ', ytrain[:5])

print('valid\_target =', valid\_target[:5].values)

print('yvalid =', yvalid[:5])

num\_acc = 0

for x **in** metrics\_now:

if x == 1:

*#accuracy\_score criterion*

acc\_train = round(metrics.accuracy\_score(train\_target, ytrain), 2)

acc\_valid = round(metrics.accuracy\_score(valid\_target, yvalid), 2)

elif x == 2:

*#rmse criterion*

acc\_train = round(acc\_rmse(train\_target, ytrain), 2)

acc\_valid = round(acc\_rmse(valid\_target, yvalid), 2)

elif x == 3:

*#relative error criterion*

acc\_train = round(acc\_d(train\_target, ytrain) \* 100, 2)

acc\_valid = round(acc\_d(valid\_target, yvalid) \* 100, 2)

print('acc of', metrics\_all[x], 'for train =', acc\_train)

print('acc of', metrics\_all[x], 'for valid =', acc\_valid)

acc\_all[num\_acc].append(acc\_train) *#train*

acc\_all[num\_acc+1].append(acc\_valid) *#valid*

num\_acc += 2

*# Building the confusion matrices*

plot\_cm(train\_target, ytrain, valid\_target, yvalid)

return acc\_all

5. Tuning models

*ng\_curve.html#sphx-glr-auto-examples-model-selection-plot-learning-curve-py*

def plot\_learning\_curve(estimator, title, X, y, cv=None, axes=None, ylim=None,

n\_jobs=None, train\_sizes=np.linspace(.1, 1.0, 5), random\_state=0):

*"""*

*Generate 2 plots:*

*- the test and training learning curve,*

*- the training samples vs fit times curve.*

*Parameters*

*----------*

*estimator : object type that implements the "fit" and "predict" methods*

*An object of that type which is cloned for each validation.*

*title : string*

*Title for the chart.*

*X : array-like, shape (n\_samples, n\_features)*

*Training vector, where n\_samples is the number of samples and*

*n\_features is the number of features.*

*y : array-like, shape (n\_samples) or (n\_samples, n\_features), optional*

*Target relative to X for classification or regression;*

*None for unsupervised learning.*

*axes : array of 3 axes, optional (default=None)*

*Axes to use for plotting the curves.*

*ylim : tuple, shape (ymin, ymax), optional*

*Defines minimum and maximum yvalues plotted.*

*cv : int, cross-validation generator or an iterable, optional*

*Determines the cross-validation splitting strategy.*

*Possible inputs for cv are:*

*- None, to use the default 5-fold cross-validation,*

*- integer, to specify the number of folds.*

*- :term:`CV splitter`,*

*- An iterable yielding (train, test) splits as arrays of indices.*

*For integer/None inputs, if ``y`` is binary or multiclass,*

*:class:`StratifiedKFold` used. If the estimator is not a classifier*

*or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.*

*Refer :ref:`User Guide <cross\_validation>` for the various*

*cross-validators that can be used here.*

*train\_sizes : array-like, shape (n\_ticks,), dtype float or int*

*Relative or absolute numbers of training examples that will be used to*

*generate the learning curve. If the dtype is float, it is regarded as a*

*fraction of the maximum size of the training set (that is determined*

*by the selected validation method), i.e. it has to be within (0, 1].*

*Otherwise it is interpreted as absolute sizes of the training sets.*

*Note that for classification the number of samples usually have to*

*be big enough to contain at least one sample from each class.*

*(default: np.linspace(0.1, 1.0, 5))*

*random\_state : random\_state*

*"""*

fig, axes = plt.subplots(2, 1, figsize=(20, 10))

if axes **is** None:

\_, axes = plt.subplots(1, 2, figsize=(20, 5))

axes[0].set\_title(title)

if ylim **is** **not** None:

axes[0].set\_ylim(\*ylim)

axes[0].set\_xlabel("Training examples")

axes[0].set\_ylabel("Score")

cv\_train = ShuffleSplit(n\_splits=cv\_n\_split, test\_size=test\_train\_split\_part, random\_state=random\_state)

train\_sizes, train\_scores, test\_scores, fit\_times, \_ = \

learning\_curve(estimator=estimator, X=X, y=y, cv=cv,

train\_sizes=train\_sizes,

return\_times=True)

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1)

test\_scores\_std = np.std(test\_scores, axis=1)

fit\_times\_mean = np.mean(fit\_times, axis=1)

fit\_times\_std = np.std(fit\_times, axis=1)

*# Plot learning curve*

axes[0].grid()

axes[0].fill\_between(train\_sizes, train\_scores\_mean - train\_scores\_std,

train\_scores\_mean + train\_scores\_std, alpha=0.1,

color="r")

axes[0].fill\_between(train\_sizes, test\_scores\_mean - test\_scores\_std,

test\_scores\_mean + test\_scores\_std, alpha=0.1,

color="g")

axes[0].plot(train\_sizes, train\_scores\_mean, 'o-', color="r",

label="Training score")

axes[0].plot(train\_sizes, test\_scores\_mean, 'o-', color="g",

label="Cross-validation score")

axes[0].legend(loc="best")

*# Plot n\_samples vs fit\_times*

axes[1].grid()

axes[1].plot(train\_sizes, fit\_times\_mean, 'o-')

axes[1].fill\_between(train\_sizes, fit\_times\_mean - fit\_times\_std,

fit\_times\_mean + fit\_times\_std, alpha=0.1)

axes[1].set\_xlabel("Training examples")

axes[1].set\_ylabel("fit\_times")

axes[1].set\_title("Scalability of the model")

plt.show()

return

5.1 Decision Tree Classifier

 [32]:

*# Decision Tree Classifier*

decision\_tree = DecisionTreeClassifier()

param\_grid = {'min\_samples\_leaf': [i for i **in** range(2,12)]}

decision\_tree\_CV = GridSearchCV(decision\_tree, param\_grid=param\_grid, cv=cv\_train, verbose=False)

decision\_tree\_CV.fit(train, train\_target)

print(decision\_tree\_CV.best\_params\_)

acc\_all = acc\_metrics\_calc(0, acc\_all, decision\_tree\_CV, train, valid, train\_target, valid\_target)

{'min\_samples\_leaf': 11}

train\_target = [0 1 1 0 0]

ytrain = [0 1 0 0 0]

valid\_target = [1 1 0 1 1]

yvalid = [0 0 0 0 1]

acc of acc for train = 0.82

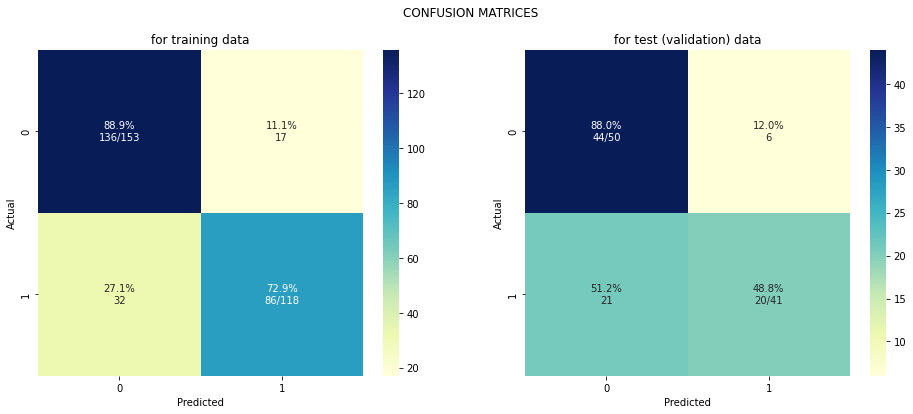
acc of acc for valid = 0.7

acc of rmse for train = 0.43

acc of rmse for valid = 0.54

acc of re for train = 41.53

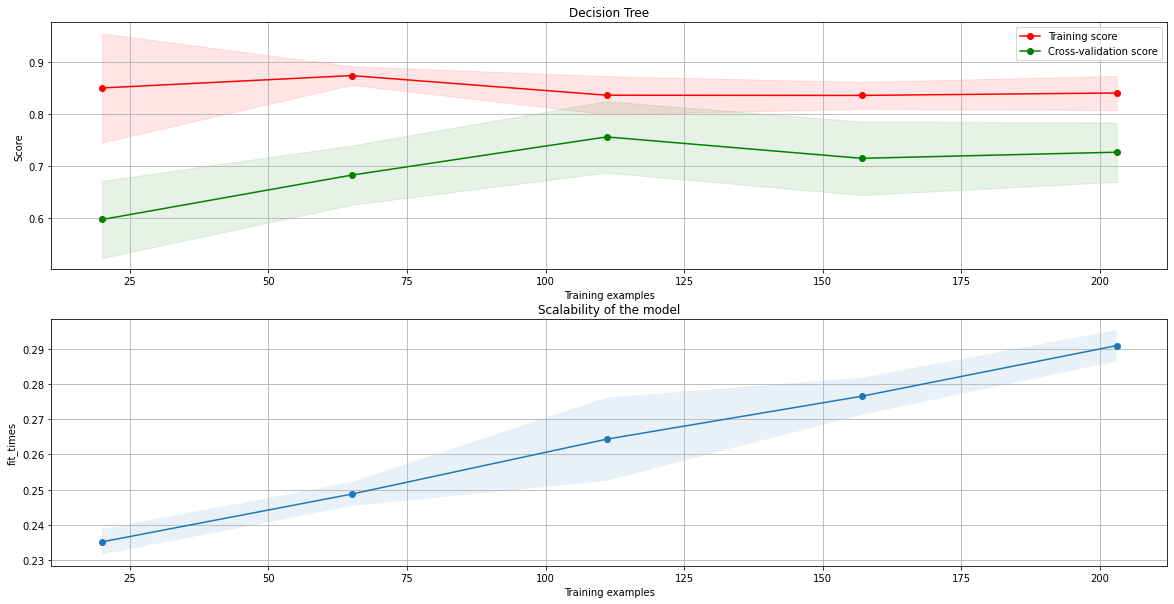
acc of re for valid = 65.85



In [33]:

*# Building learning curve of model*

plot\_learning\_curve(decision\_tree\_CV, "Decision Tree", train, train\_target, cv=cv\_train)



5.2 Random Forest Classifier

In [34]:

%%time

*# Random Forest*

random\_forest = RandomForestClassifier()

param\_grid = {'n\_estimators': [50, 60, 80, 100], 'min\_samples\_leaf': [12, 13, 14, 15, 16, 17],

'max\_features': ['auto'], 'max\_depth': [3, 4, 5, 6], 'criterion': ['gini'], 'bootstrap': [False]}

random\_forest\_CV = GridSearchCV(estimator=random\_forest, param\_grid=param\_grid,

cv=cv\_train, verbose=False)

random\_forest\_CV.fit(train, train\_target)

print(random\_forest\_CV.best\_params\_)

acc\_all = acc\_metrics\_calc(1, acc\_all, random\_forest\_CV, train, valid, train\_target, valid\_target)

{'bootstrap': False, 'criterion': 'gini', 'max\_depth': 6, 'max\_features': 'auto', 'min\_samples\_leaf': 15, 'n\_estimators': 50}

train\_target = [0 1 1 0 0]

ytrain = [1 1 1 0 0]

valid\_target = [1 1 0 1 1]

yvalid = [0 0 0 0 1]

acc of acc for train = 0.82

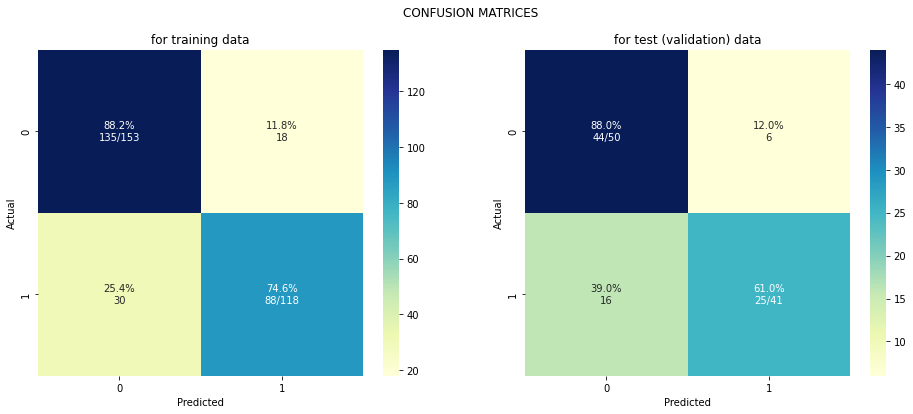
acc of acc for valid = 0.76

acc of rmse for train = 0.42

acc of rmse for valid = 0.49

acc of re for train = 40.68

acc of re for valid = 53.66



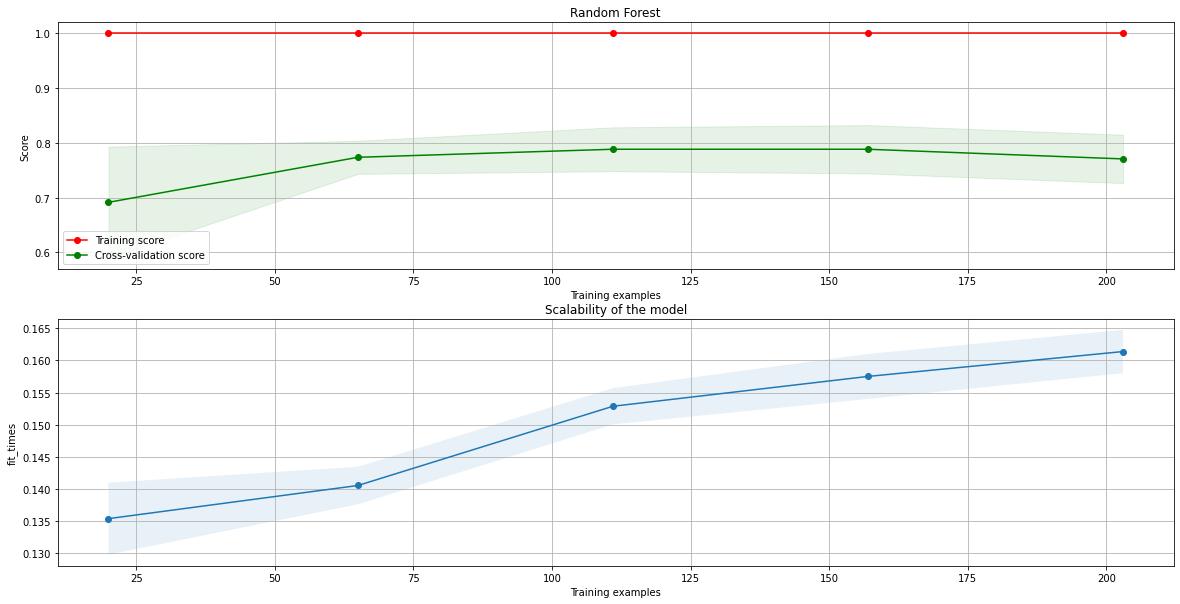
CPU times: user 47.1 s, sys: 407 ms, total: 47.5 s

Wall time: 47.2 s

In [35]:

*# Building learning curve of model*

plot\_learning\_curve(random\_forest, "Random Forest", train, train\_target, cv=cv\_train)



5.3 XGB Classifier

In [36]:

%%time

*# XGBoost Classifier*

xgb\_clf = xgb.XGBClassifier(objective='reg:squarederror')

parameters = {'n\_estimators': [30, 40, 50, 60, 75, 100],

'learning\_rate': [0.01, 0.03, 0.05, 0.1],

'max\_depth': [3, 4, 5]}

xgb\_reg = GridSearchCV(estimator=xgb\_clf, param\_grid=parameters, cv=cv\_train).fit(train, train\_target)

print("Best score: **%0.3f**" % xgb\_reg.best\_score\_)

print("Best parameters set:", xgb\_reg.best\_params\_)

acc\_all = acc\_metrics\_calc(2, acc\_all, xgb\_reg, train, valid, train\_target, valid\_target)

Best score: 0.768

Best parameters set: {'learning\_rate': 0.03, 'max\_depth': 3, 'n\_estimators': 60}

train\_target = [0 1 1 0 0]

ytrain = [1 1 1 0 0]

valid\_target = [1 1 0 1 1]

yvalid = [0 0 0 0 1]

acc of acc for train = 0.88

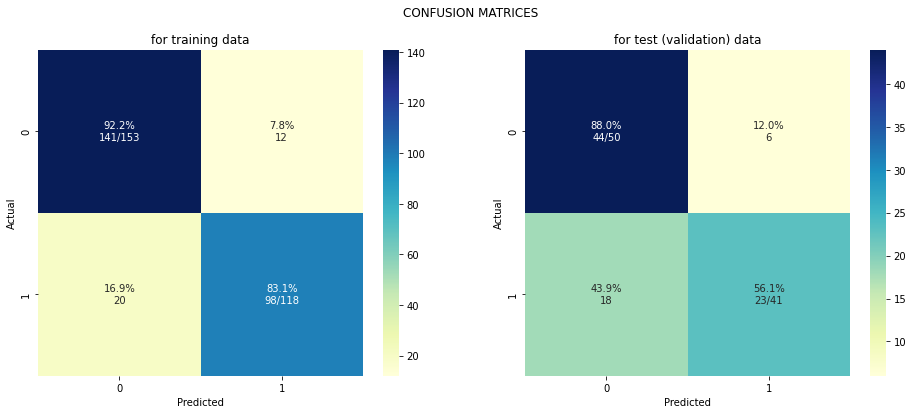
acc of acc for valid = 0.74

acc of rmse for train = 0.34

acc of rmse for valid = 0.51

acc of re for train = 27.12

acc of re for valid = 58.54



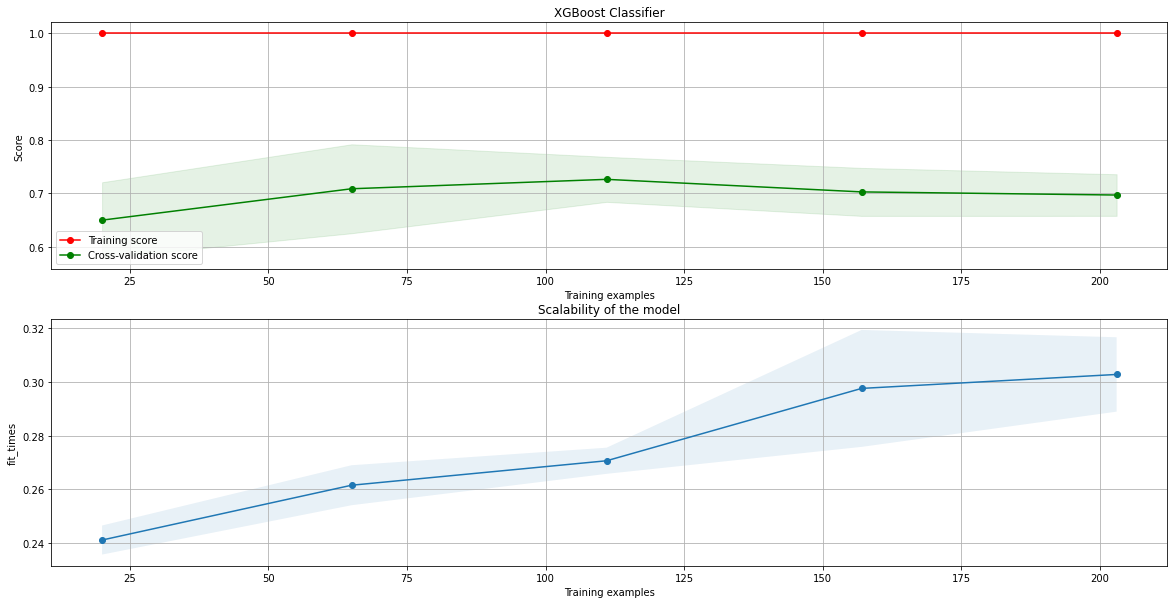
CPU times: user 4min 37s, sys: 2.8 s, total: 4min 39s

Wall time: 1min 11s

In [37]:

*# Building learning curve of model*

plot\_learning\_curve(xgb\_clf, "XGBoost Classifier", train, train\_target, cv=cv\_train)



5.4 LGBM Classifier

In [38]:

*#%% split training set to validation set*

Xtrain, Xval, Ztrain, Zval = train\_test\_split(train, train\_target, test\_size=test\_train\_split\_part, random\_state=random\_state)

modelL = lgb.LGBMClassifier(n\_estimators=10000, num\_leaves=40)

modelL.fit(Xtrain, Ztrain, eval\_set=[(Xval, Zval)], early\_stopping\_rounds=200, verbose=True)

[1] valid\_0's binary\_logloss: 0.645049

[2] valid\_0's binary\_logloss: 0.628612

[3] valid\_0's binary\_logloss: 0.615584

[4] valid\_0's binary\_logloss: 0.605008

[5] valid\_0's binary\_logloss: 0.592086

[6] valid\_0's binary\_logloss: 0.583354

[7] valid\_0's binary\_logloss: 0.56706

[8] valid\_0's binary\_logloss: 0.551119

[9] valid\_0's binary\_logloss: 0.543756

[10] valid\_0's binary\_logloss: 0.539334

[11] valid\_0's binary\_logloss: 0.538419

[12] valid\_0's binary\_logloss: 0.533309

[13] valid\_0's binary\_logloss: 0.526364

[14] valid\_0's binary\_logloss: 0.518809

[15] valid\_0's binary\_logloss: 0.516534

[16] valid\_0's binary\_logloss: 0.515419

[17] valid\_0's binary\_logloss: 0.518452

[18] valid\_0's binary\_logloss: 0.519006

[19] valid\_0's binary\_logloss: 0.520657

[20] valid\_0's binary\_logloss: 0.517584

[21] valid\_0's binary\_logloss: 0.521726

[22] valid\_0's binary\_logloss: 0.523686

[23] valid\_0's binary\_logloss: 0.522808

[24] valid\_0's binary\_logloss: 0.519967

[25] valid\_0's binary\_logloss: 0.517494

[26] valid\_0's binary\_logloss: 0.520464

[27] valid\_0's binary\_logloss: 0.519673

[28] valid\_0's binary\_logloss: 0.5179

[29] valid\_0's binary\_logloss: 0.516924

[30] valid\_0's binary\_logloss: 0.517781

[31] valid\_0's binary\_logloss: 0.52524

[32] valid\_0's binary\_logloss: 0.525103

[33] valid\_0's binary\_logloss: 0.523768

[34] valid\_0's binary\_logloss: 0.523536

[35] valid\_0's binary\_logloss: 0.521316

[36] valid\_0's binary\_logloss: 0.523444

[37] valid\_0's binary\_logloss: 0.525737

[38] valid\_0's binary\_logloss: 0.524573

[39] valid\_0's binary\_logloss: 0.525842

[40] valid\_0's binary\_logloss: 0.530485

[41] valid\_0's binary\_logloss: 0.532843

[42] valid\_0's binary\_logloss: 0.532148

[43] valid\_0's binary\_logloss: 0.53408

[44] valid\_0's binary\_logloss: 0.532897

[45] valid\_0's binary\_logloss: 0.532142

[46] valid\_0's binary\_logloss: 0.534791

[47] valid\_0's binary\_logloss: 0.536851

[48] valid\_0's binary\_logloss: 0.537858

[49] valid\_0's binary\_logloss: 0.534851

[50] valid\_0's binary\_logloss: 0.535245

[51] valid\_0's binary\_logloss: 0.534687

[52] valid\_0's binary\_logloss: 0.53576

[53] valid\_0's binary\_logloss: 0.539094

[54] valid\_0's binary\_logloss: 0.546328

[55] valid\_0's binary\_logloss: 0.550474

[56] valid\_0's binary\_logloss: 0.556925

[57] valid\_0's binary\_logloss: 0.559429

[58] valid\_0's binary\_logloss: 0.559634

[59] valid\_0's binary\_logloss: 0.562981

[60] valid\_0's binary\_logloss: 0.560878

[61] valid\_0's binary\_logloss: 0.564784

[62] valid\_0's binary\_logloss: 0.569616

[63] valid\_0's binary\_logloss: 0.569773

[64] valid\_0's binary\_logloss: 0.577556

[65] valid\_0's binary\_logloss: 0.58179

[66] valid\_0's binary\_logloss: 0.58049

[67] valid\_0's binary\_logloss: 0.584825

[68] valid\_0's binary\_logloss: 0.58934

[69] valid\_0's binary\_logloss: 0.59192

[70] valid\_0's binary\_logloss: 0.595457

[71] valid\_0's binary\_logloss: 0.596458

[72] valid\_0's binary\_logloss: 0.599729

[73] valid\_0's binary\_logloss: 0.603368

[74] valid\_0's binary\_logloss: 0.606888

[75] valid\_0's binary\_logloss: 0.608506

[76] valid\_0's binary\_logloss: 0.609108

[77] valid\_0's binary\_logloss: 0.610135

[78] valid\_0's binary\_logloss: 0.612933

[79] valid\_0's binary\_logloss: 0.615142

[80] valid\_0's binary\_logloss: 0.611084

[81] valid\_0's binary\_logloss: 0.613973

[82] valid\_0's binary\_logloss: 0.615439

[83] valid\_0's binary\_logloss: 0.614837

[84] valid\_0's binary\_logloss: 0.617275

[85] valid\_0's binary\_logloss: 0.621032

[86] valid\_0's binary\_logloss: 0.620922

[87] valid\_0's binary\_logloss: 0.623474

[88] valid\_0's binary\_logloss: 0.624458

[89] valid\_0's binary\_logloss: 0.626076

[90] valid\_0's binary\_logloss: 0.628252

[91] valid\_0's binary\_logloss: 0.631448

[92] valid\_0's binary\_logloss: 0.633773

[93] valid\_0's binary\_logloss: 0.636139

[94] valid\_0's binary\_logloss: 0.638078

[95] valid\_0's binary\_logloss: 0.639798

[96] valid\_0's binary\_logloss: 0.640261

[97] valid\_0's binary\_logloss: 0.643042

[98] valid\_0's binary\_logloss: 0.644675

[99] valid\_0's binary\_logloss: 0.644155

[100] valid\_0's binary\_logloss: 0.64477

[101] valid\_0's binary\_logloss: 0.652303

[102] valid\_0's binary\_logloss: 0.657836

[103] valid\_0's binary\_logloss: 0.659591

[104] valid\_0's binary\_logloss: 0.664124

[105] valid\_0's binary\_logloss: 0.662096

[106] valid\_0's binary\_logloss: 0.671568

[107] valid\_0's binary\_logloss: 0.671461

[108] valid\_0's binary\_logloss: 0.674958

[109] valid\_0's binary\_logloss: 0.679175

[110] valid\_0's binary\_logloss: 0.679528

[111] valid\_0's binary\_logloss: 0.675579

[112] valid\_0's binary\_logloss: 0.674403

[113] valid\_0's binary\_logloss: 0.675727

[114] valid\_0's binary\_logloss: 0.677759

[115] valid\_0's binary\_logloss: 0.677179

[116] valid\_0's binary\_logloss: 0.679071

[117] valid\_0's binary\_logloss: 0.680812

[118] valid\_0's binary\_logloss: 0.683245

[119] valid\_0's binary\_logloss: 0.685444

[120] valid\_0's binary\_logloss: 0.686744

[121] valid\_0's binary\_logloss: 0.688631

[122] valid\_0's binary\_logloss: 0.687349

[123] valid\_0's binary\_logloss: 0.692559

[124] valid\_0's binary\_logloss: 0.69508

[125] valid\_0's binary\_logloss: 0.696107

[126] valid\_0's binary\_logloss: 0.697425

[127] valid\_0's binary\_logloss: 0.699265

[128] valid\_0's binary\_logloss: 0.700754

[129] valid\_0's binary\_logloss: 0.701056

[130] valid\_0's binary\_logloss: 0.70129

[131] valid\_0's binary\_logloss: 0.699014

[132] valid\_0's binary\_logloss: 0.698319

[133] valid\_0's binary\_logloss: 0.704376

[134] valid\_0's binary\_logloss: 0.708412

[135] valid\_0's binary\_logloss: 0.707813

[136] valid\_0's binary\_logloss: 0.705777

[137] valid\_0's binary\_logloss: 0.710161

[138] valid\_0's binary\_logloss: 0.711195

[139] valid\_0's binary\_logloss: 0.713255

[140] valid\_0's binary\_logloss: 0.713907

[141] valid\_0's binary\_logloss: 0.716568

[142] valid\_0's binary\_logloss: 0.720011

[143] valid\_0's binary\_logloss: 0.724665

[144] valid\_0's binary\_logloss: 0.727715

[145] valid\_0's binary\_logloss: 0.730276

[146] valid\_0's binary\_logloss: 0.728785

[147] valid\_0's binary\_logloss: 0.72979

[148] valid\_0's binary\_logloss: 0.727819

[149] valid\_0's binary\_logloss: 0.732754

[150] valid\_0's binary\_logloss: 0.732808

[151] valid\_0's binary\_logloss: 0.741104

[152] valid\_0's binary\_logloss: 0.745773

[153] valid\_0's binary\_logloss: 0.745311

[154] valid\_0's binary\_logloss: 0.746491

[155] valid\_0's binary\_logloss: 0.744933

[156] valid\_0's binary\_logloss: 0.748493

[157] valid\_0's binary\_logloss: 0.753112

[158] valid\_0's binary\_logloss: 0.757454

[159] valid\_0's binary\_logloss: 0.759284

[160] valid\_0's binary\_logloss: 0.761331

[161] valid\_0's binary\_logloss: 0.762443

[162] valid\_0's binary\_logloss: 0.764603

[163] valid\_0's binary\_logloss: 0.763676

[164] valid\_0's binary\_logloss: 0.76864

[165] valid\_0's binary\_logloss: 0.771445

[166] valid\_0's binary\_logloss: 0.776363

[167] valid\_0's binary\_logloss: 0.78474

[168] valid\_0's binary\_logloss: 0.787232

[169] valid\_0's binary\_logloss: 0.78554

[170] valid\_0's binary\_logloss: 0.789005

[171] valid\_0's binary\_logloss: 0.791013

[172] valid\_0's binary\_logloss: 0.795299

[173] valid\_0's binary\_logloss: 0.796209

[174] valid\_0's binary\_logloss: 0.80211

[175] valid\_0's binary\_logloss: 0.805768

[176] valid\_0's binary\_logloss: 0.808432

[177] valid\_0's binary\_logloss: 0.810293

[178] valid\_0's binary\_logloss: 0.807251

[179] valid\_0's binary\_logloss: 0.808082

[180] valid\_0's binary\_logloss: 0.810712

[181] valid\_0's binary\_logloss: 0.811596

[182] valid\_0's binary\_logloss: 0.812437

[183] valid\_0's binary\_logloss: 0.811823

[184] valid\_0's binary\_logloss: 0.814289

[185] valid\_0's binary\_logloss: 0.812455

[186] valid\_0's binary\_logloss: 0.811661

[187] valid\_0's binary\_logloss: 0.820199

[188] valid\_0's binary\_logloss: 0.821706

[189] valid\_0's binary\_logloss: 0.822795

[190] valid\_0's binary\_logloss: 0.820028

[191] valid\_0's binary\_logloss: 0.822607

[192] valid\_0's binary\_logloss: 0.825762

[193] valid\_0's binary\_logloss: 0.826985

[194] valid\_0's binary\_logloss: 0.830846

[195] valid\_0's binary\_logloss: 0.831152

[196] valid\_0's binary\_logloss: 0.836777

[197] valid\_0's binary\_logloss: 0.836897

[198] valid\_0's binary\_logloss: 0.837874

[199] valid\_0's binary\_logloss: 0.842823

[200] valid\_0's binary\_logloss: 0.84631

[201] valid\_0's binary\_logloss: 0.848535

[202] valid\_0's binary\_logloss: 0.849492

[203] valid\_0's binary\_logloss: 0.848605

[204] valid\_0's binary\_logloss: 0.846468

[205] valid\_0's binary\_logloss: 0.848918

[206] valid\_0's binary\_logloss: 0.84861

[207] valid\_0's binary\_logloss: 0.855545

[208] valid\_0's binary\_logloss: 0.861277

[209] valid\_0's binary\_logloss: 0.863138

[210] valid\_0's binary\_logloss: 0.861779

[211] valid\_0's binary\_logloss: 0.868609

[212] valid\_0's binary\_logloss: 0.871435

[213] valid\_0's binary\_logloss: 0.872785

[214] valid\_0's binary\_logloss: 0.876514

[215] valid\_0's binary\_logloss: 0.874297

[216] valid\_0's binary\_logloss: 0.87993

Out[38]:

LGBMClassifier(n\_estimators=10000, num\_leaves=40)

In [39]:

acc\_all = acc\_metrics\_calc(3, acc\_all, modelL, train, valid, train\_target, valid\_target)

train\_target = [0 1 1 0 0]

ytrain = [1 1 1 0 0]

valid\_target = [1 1 0 1 1]

yvalid = [1 0 0 0 1]

acc of acc for train = 0.85

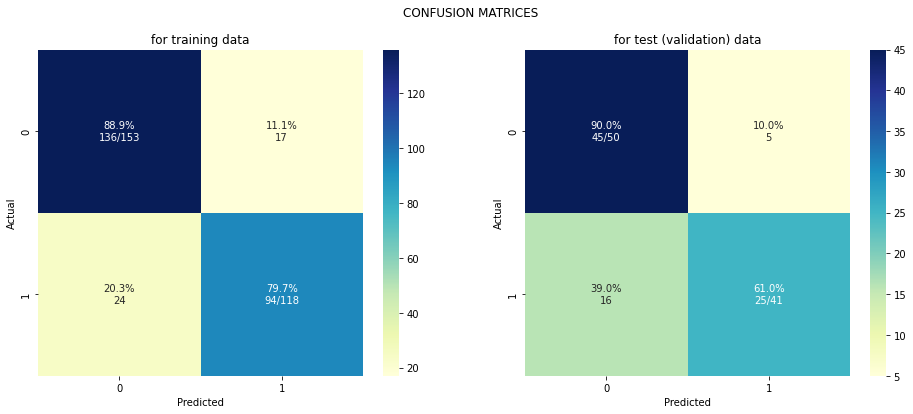
acc of acc for valid = 0.77

acc of rmse for train = 0.39

acc of rmse for valid = 0.48

acc of re for train = 34.75

acc of re for valid = 51.22



In [40]:

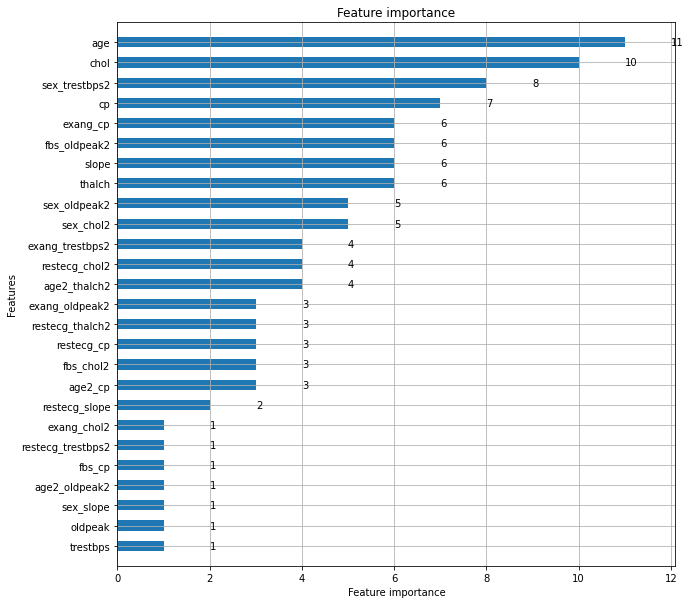
fig = plt.figure(figsize = (10,10))

axes = fig.add\_subplot(111)

lgb.plot\_importance(modelL,ax = axes,height = 0.5)

plt.show();

plt.close()



5.5 Extra Trees Classifier

In [41]:

*# Extra Trees Classifier*

etr = ExtraTreesClassifier()

etr\_CV = GridSearchCV(estimator=etr, param\_grid={'min\_samples\_leaf' : [20, 25, 30]}, cv=cv\_train, verbose=False)

etr\_CV.fit(train, train\_target)

print(etr\_CV.best\_params\_)

acc\_all = acc\_metrics\_calc(4, acc\_all, etr\_CV, train, valid, train\_target, valid\_target)

{'min\_samples\_leaf': 25}

train\_target = [0 1 1 0 0]

ytrain = [1 1 1 0 0]

valid\_target = [1 1 0 1 1]

yvalid = [1 0 0 0 1]

acc of acc for train = 0.77

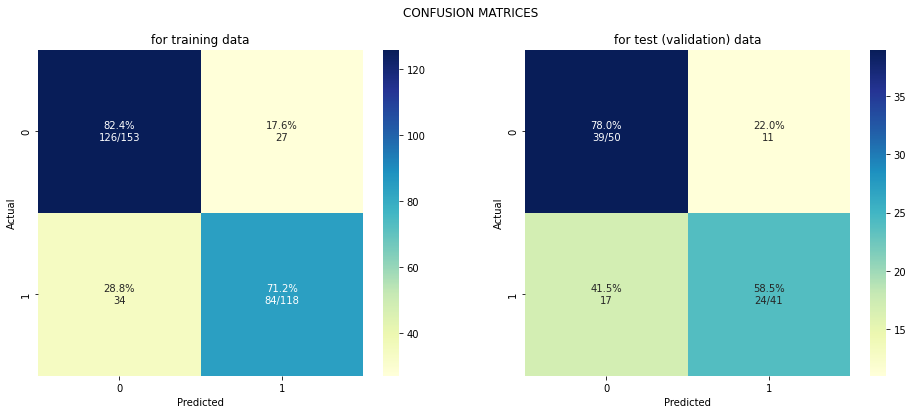
acc of acc for valid = 0.69

acc of rmse for train = 0.47

acc of rmse for valid = 0.55

acc of re for train = 51.69

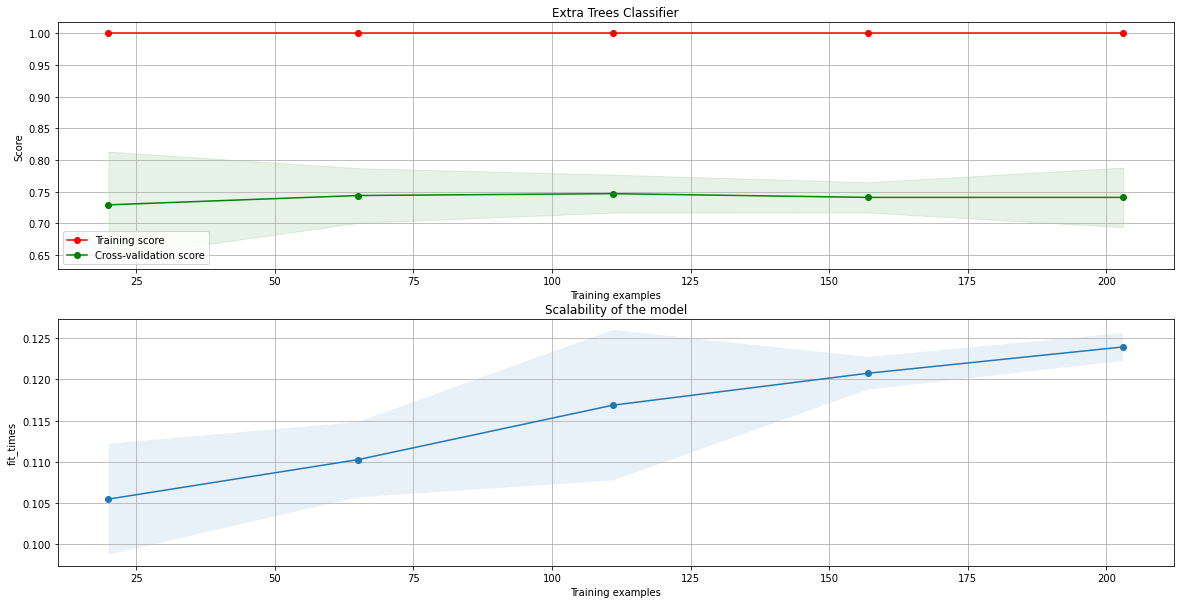
acc of re for valid = 68.29



In [42]:

*# Building learning curve of model*

plot\_learning\_curve(etr, "Extra Trees Classifier", train, train\_target, cv=cv\_train)



5.6 AdaBoost Classifier

In [43]:

*# AdaBoost Classifier*

Ada\_Boost = AdaBoostClassifier()

Ada\_Boost\_CV = GridSearchCV(estimator=Ada\_Boost, param\_grid={'learning\_rate' : [0.09, 0.1, 0.2]}, cv=cv\_train, verbose=False)

Ada\_Boost\_CV.fit(train, train\_target)

print(Ada\_Boost\_CV.best\_params\_)

acc\_all = acc\_metrics\_calc(5, acc\_all, Ada\_Boost\_CV, train, valid, train\_target, valid\_target)

{'learning\_rate': 0.2}

train\_target = [0 1 1 0 0]

ytrain = [0 1 1 0 0]

valid\_target = [1 1 0 1 1]

yvalid = [0 0 0 0 1]

acc of acc for train = 0.86

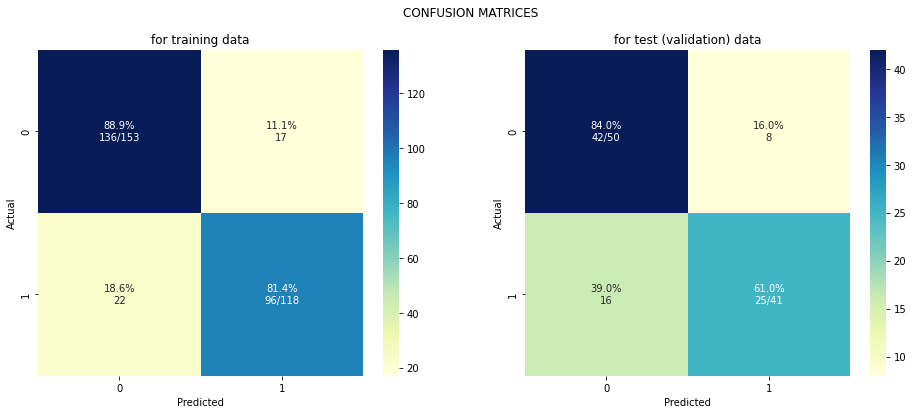
acc of acc for valid = 0.74

acc of rmse for train = 0.38

acc of rmse for valid = 0.51

acc of re for train = 33.05

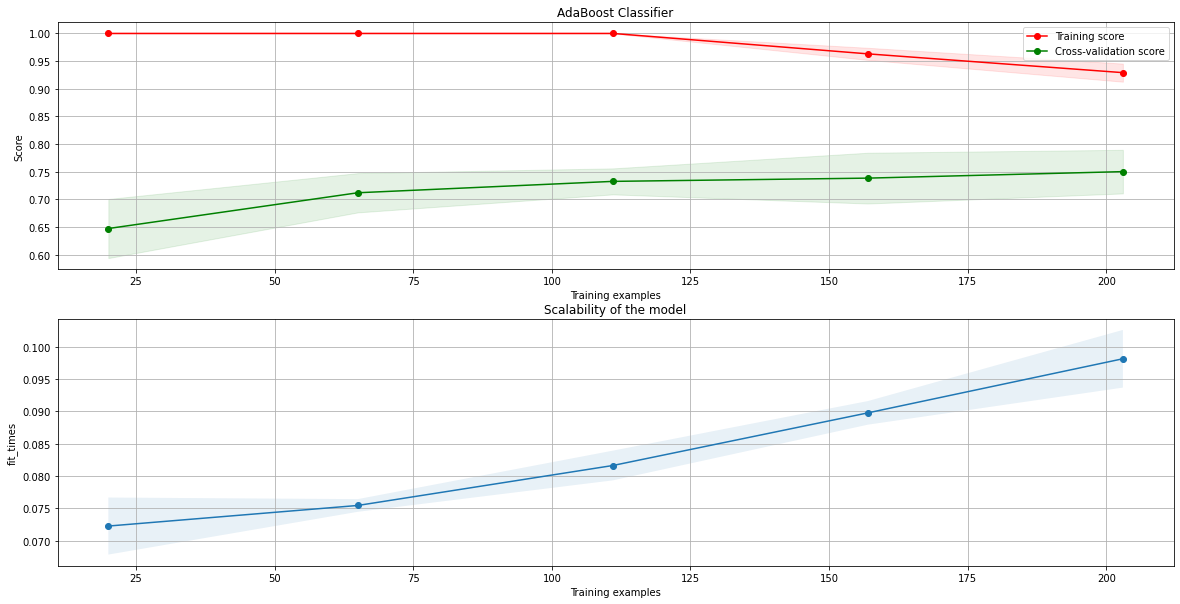
acc of re for valid = 58.54



In [44]:

*# Building learning curve of model*

plot\_learning\_curve(Ada\_Boost, "AdaBoost Classifier", train, train\_target, cv=cv\_train)



6. Models evaluation

In [45]:

models = pd.DataFrame({

'Model': ['Decision Tree Classifier',

'Random Forest Classifier',

'XGB Classifier',

'LGBM Classifier',

'ExtraTrees Classifier',

'AdaBoost Classifier',

]})

In [46]:

for x **in** metrics\_now:

xs = metrics\_all[x]

models[xs + '\_train'] = acc\_all[(x-1)\*2]

models[xs + '\_valid'] = acc\_all[(x-1)\*2+1]

if xs == "acc":

models[xs + '\_diff'] = models[xs + '\_train'] - models[xs + '\_valid']

*#models*

In [47]:

print('Prediction accuracy for models')

ms = metrics\_all[metrics\_now[0]] *# the first from metrics*

models[['Model', ms + '\_train', ms + '\_valid', 'acc\_diff']].sort\_values(by=[(ms + '\_valid'), (ms + '\_train')], ascending=False)

Prediction accuracy for models

Out[47]:

|  | Model | acc\_train | acc\_valid | acc\_diff |
| --- | --- | --- | --- | --- |
| 3 | LGBM Classifier | 0.85 | 0.77 | 0.08 |
| 1 | Random Forest Classifier | 0.82 | 0.76 | 0.06 |
| 2 | XGB Classifier | 0.88 | 0.74 | 0.14 |
| 5 | AdaBoost Classifier | 0.86 | 0.74 | 0.12 |
| 0 | Decision Tree Classifier | 0.82 | 0.70 | 0.12 |
| 4 | ExtraTrees Classifier | 0.77 | 0.69 | 0.08 |

In [48]:

pd.options.display.float\_format = '**{:,.2f}**'.format

In [49]:

for x **in** metrics\_now:

*# Plot*

xs = metrics\_all[x]

xs\_train = metrics\_all[x] + '\_train'

xs\_test = metrics\_all[x] + '\_valid'

plt.figure(figsize=[15,6])

xx = models['Model']

plt.tick\_params(labelsize=14)

plt.plot(xx, models[xs\_train], label = xs\_train)

plt.plot(xx, models[xs\_test], label = xs\_test)

plt.legend()

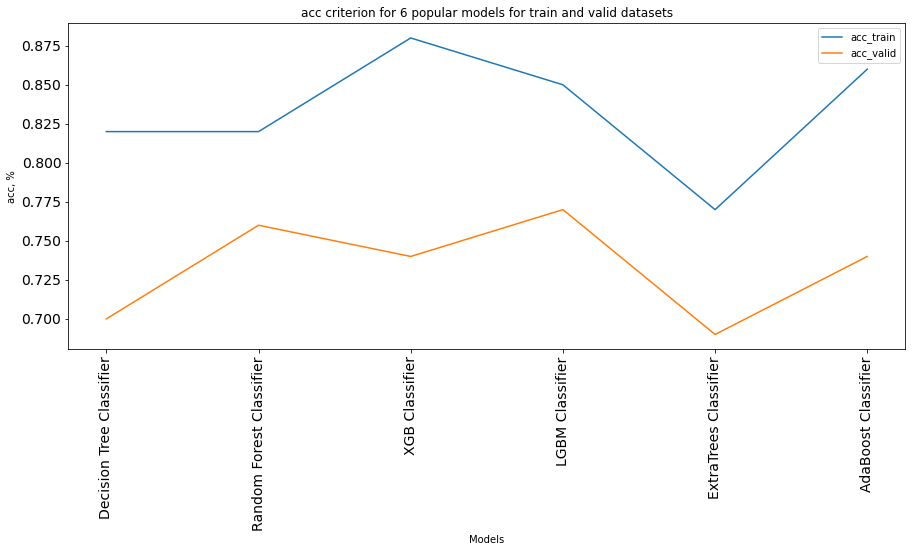
plt.title(str(xs) + ' criterion for ' + str(num\_models) + ' popular models for train and valid datasets')

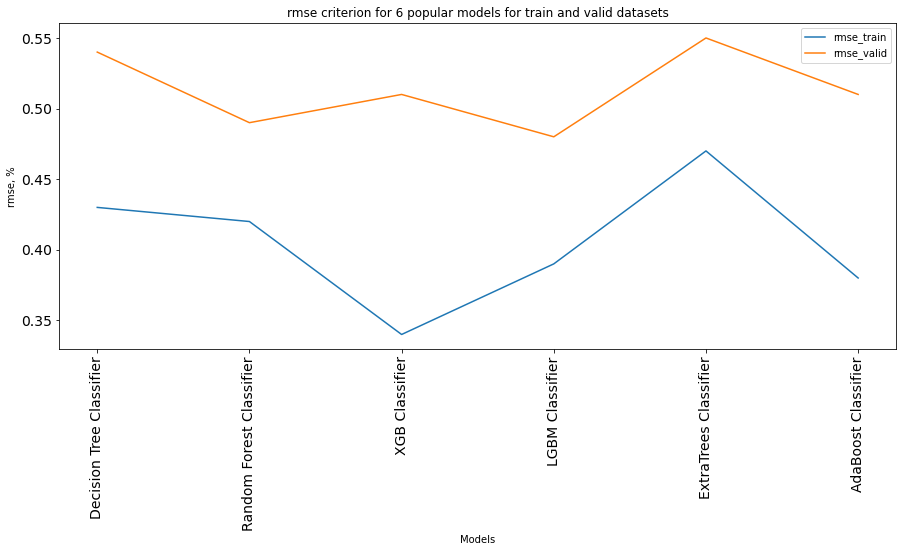
plt.xlabel('Models')

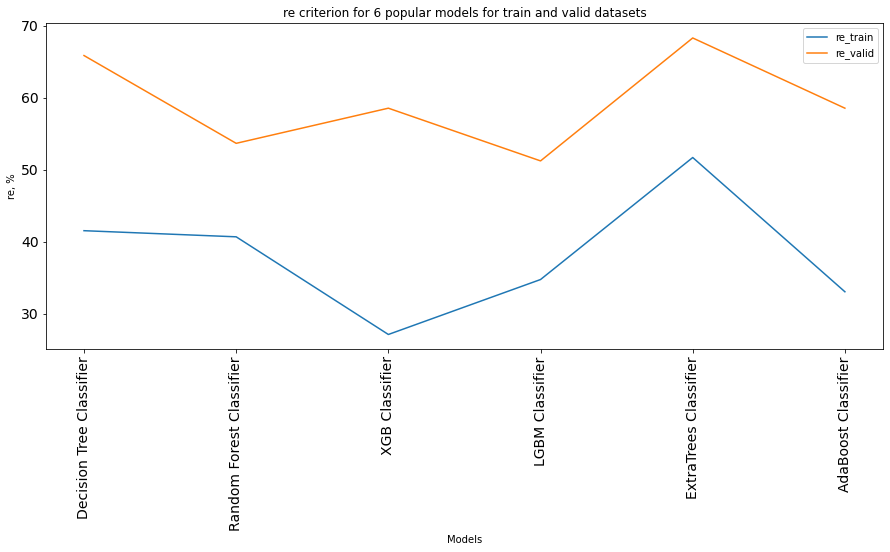
plt.ylabel(xs + ', %')

plt.xticks(xx, rotation='vertical')

plt.show()







Larger values **r2\_score\_diff** mean overfitting.

7. Conclusion

In [50]:

*# Choose the number of metric by which the best models will be determined => {1: 'accuracy\_score', 2 : 'relative\_error', 3 : 'rmse'}*

metrics\_main = 1

xs = metrics\_all[metrics\_main]

xs\_train = metrics\_all[metrics\_main] + '\_train'

xs\_test = metrics\_all[metrics\_main] + '\_valid'

print('The best models by the',xs,'criterion:')

direct\_sort = False if (metrics\_main==1) else True

models.sort\_values(by=[xs\_test, xs\_train], ascending=direct\_sort)

The best models by the acc criterion:

Out[50]:

|  | Model | acc\_train | acc\_valid | acc\_diff | rmse\_train | rmse\_valid | re\_train | re\_valid |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | LGBM Classifier | 0.85 | 0.77 | 0.08 | 0.39 | 0.48 | 34.75 | 51.22 |
| 1 | Random Forest Classifier | 0.82 | 0.76 | 0.06 | 0.42 | 0.49 | 40.68 | 53.66 |
| 2 | XGB Classifier | 0.88 | 0.74 | 0.14 | 0.34 | 0.51 | 27.12 | 58.54 |
| 5 | AdaBoost Classifier | 0.86 | 0.74 | 0.12 | 0.38 | 0.51 | 33.05 | 58.54 |
| 0 | Decision Tree Classifier | 0.82 | 0.70 | 0.12 | 0.43 | 0.54 | 41.53 | 65.85 |
| 4 | ExtraTrees Classifier | 0.77 | 0.69 | 0.08 | 0.47 | 0.55 | 51.69 | 68.29 |

The best models by accuracy\_score:

In [51]:

*# Selection the best models*

models\_best = models[(models.acc\_diff < 0.1) & (models.acc\_valid > 0.7)]

if len(models\_best)>0:

print('The best models:')

display(models\_best[['Model', 'acc\_train', 'acc\_valid']].sort\_values(by=['acc\_valid'], ascending=False))

*# Selection the best models from the best*

models\_best\_best = models\_best[(models\_best.acc\_valid > 0.9)]

if len(models\_best\_best)>0:

print('Optimal model:')

display(models\_best\_best[['Model', 'acc\_train', 'acc\_valid']].sort\_values(by=['acc\_valid'], ascending=False))

else: print('But no model provides good accuracy at least above 0.9')

else:

print('There are no good models - either they have not learned enough, or they have overfit!')

The best models:

|  | Model | acc\_train | acc\_valid |
| --- | --- | --- | --- |
| 3 | LGBM Classifier | 0.85 | 0.77 |
| 1 | Random Forest Classifier | 0.82 | 0.76 |

But no model provides good accuracy at least above 0.9

I hope you find this notebook useful and enjoyable.

Your comments and feedback are most welcome.

[Go to Top](https://www.kaggleusercontent.com/kf/113409560/eyJhbGciOiJkaXIiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0..EF8L1AJ9gLPNnlenYQlJ7w.6JMuD0YYZlEN1BRC0tqUK-EHcgpeJgwYjOJDq2bGne87Z-izY_Sw0aQm5gmjwMuSQCMcN5YtwjgWTgEtIq_6WYlHn2DP9R0kSTvTDHBwguGUuqtcL5zFsO8h1z8RT8f_K9H9O5COsc01YuPNEOty-Qge7OsPLXElnMoHoP4pKsrWcWkkeAmEdsVLek-00X-FKNZglBB7dXIm7OIFW04UN5RzkIcfOqsncLFwdhBfQbZ1gm4VHFiRIXCNdYTyKZ-MZ3ka68gOAmoJ6i9HV4mEY3cHhqJEmH46u202JQ8PZDOLsAEul8-z6Jr24D8rySCdukGxy9RM-ji8flmv818Fm07crZTMfe5yMyTMRrYWEoZzMTRuNwmK89Si6988DtzPECJ55fbvuRPr3WJNesDTv06ofPknAxPmDXYTGSNbbDjpDe8HvKgnikgE1DUnm0kldPWP2DFpFE1CqH64a-1QsTjLs_FHuBXPvW-uoW_I9doQIeQU0fxyDCHbunfncn6oWdIezLalb8o16nLuUAN8SDf6ZDBWGNZIBNL1q4_w0u--Ao2b8PqafjHWzHQ3g8B5eDMls3nqRQ_pxxHZne7gOdBgAMoaUiskOGzuYnEBF5Qa65MPYGArgddoaMgrvV-zCV3rH-b61V2cyhsk9kXgo1RarGYwGnIoAVs-yPWzM18mxsbKwPhlLg-rOTMMUAvT.Yqx6BWtGYjX6N273f1Bo8g/__results__.html?sharingControls=true#0)