Using several classifiers and tuning parameters - Parameters grid

From official `scikit-learn` documentation

Adapted by Claudio Sartori

Example of usage of the ***model selection*** features of `scikit-learn` and comparison of several classification methods.

1. import a sample dataset
2. do the usual preliminary data explorations and separate the predicting attributes from the *target* `'Exited'`
3. define the *models* that will be tested and prepare the *hyperparameter ranges* for the modules
4. set the list of *score functions* to choose from
5. split the dataset into two parts: train and test
6. Loop on score functions and, for each score, loop on the model labels (see details below)
   - optimize with GridSearchCV
   - test
   - store the results
7. for each scoring show the ranking of the models, and the confusion matrix given by the best model
8. for each scoring show the confusion matrix of the prediction given by the best model

In [1]:

```
"""
@author: scikit-learn.org and Claudio Sartori
"""
import warnings
warnings.filterwarnings('ignore') # uncomment this line to suppress warnings

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.metrics import classification_report, ConfusionMatrixDisplay
from sklearn.svm import SVC
from sklearn.linear_model import Perceptron
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier

print(__doc__) # print information included in the triple quotes at the beginning
```

```
@author: scikit-learn.org and Claudio Sartori
```

0. Initial settings

Set the random state and set the seed with `np.random.seed()`

Set the test set size and the number of cross valitadion splits

In [2]:

```
ts = 0.3 # test size
random_state = 42
np.random.seed(random_state) # this sets the random sequence. Setting only this the repeatability is guaranteed
                             # only if we re-execute the entire notebook
cv = 3   # number of cross-validation splits
```

1. Import the dataset

In [3]:

```

```

Out[3]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	0	42	2	0.00	1	1	1	101348.88	True
1	502	0	42	8	159660.80	3	1	0	113931.57	True
2	699	0	39	1	0.00	2	0	0	93826.63	False
3	822	1	50	7	0.00	2	1	1	10062.80	False
4	501	1	44	4	142051.07	2	0	1	74940.50	False

2. Explore the data

The output of exploration is not shown here

In [4]:

```

```

3. Define the *models*

Prepare the *hyperparameter ranges* for the modules

Put everything in a dictionary, for ease of use

In [5]:

```
model_lbls = ['dt' # decision tree
             ,'nb' # gaussian naive bayes
#              ,'lp'   # linear perceptron
#              ,'svc'  # support vector
#              ,'knn'  # k nearest neighbours
#              ,'adb'  # adaboost
#              ,'rf'   # random forest
            ]

models = {
    'dt': {'name': 'Decision Tree       ',
           'estimator': DecisionTreeClassifier(random_state=random_state),
           'param': [{'max_depth': [*range(1,20)],'class_weight':[None,'balanced']}],
          },
    'nb': {'name': 'Gaussian Naive Bayes',
           'estimator': GaussianNB(),
           'param': [{'var_smoothing': [10**exp for exp in range(-3,-12,-1)]}]
          },
    'lp': {'name': 'Linear Perceptron   ',
           'estimator': Perceptron(random_state=random_state),
           'param': [{'early_stopping': [True,False],'class_weight':[None,'balanced']}],
          },
    'svc':{'name': 'Support Vector      ',
           'estimator': SVC(random_state=random_state),
           'param': [{'kernel': ['rbf'],
```

```
                        'gamma': [1e-3, 1e-4],
                        'C': [1, 10, 100],
                        },
                        {'kernel': ['linear'],
                         'C': [1, 10, 100],
                        },
                       ]
            },
      'knn':{'name': 'K Nearest Neighbor ',
             'estimator': KNeighborsClassifier(),
             'param': [{'n_neighbors': list(range(1,7))}]
         },
      'adb':{'name': 'AdaBoost           ',
             'estimator': AdaBoostClassifier(random_state=random_state),
             'param': [{'n_estimators':[20,30,40,50]
                       ,'learning_rate':[0.5,0.75,1,1.25,1.5]}]
            },
      'rf': {'name': 'Random forest       ',
             'estimator': RandomForestClassifier(random_state=random_state),
             'param': [{'max_depth': [*range(4,10)]
                       ,'n_estimators':[*range(10,60,10)]}]
            }

}
```

4. Set the list of *score functions* to choose from

In [6]:

```
scorings = ['accuracy','precision_macro','recall_macro','f1_macro']
```

5. Split the dataset into the train and test parts

```
- the *train* part will be used for training and cross-validation (i.e. for *development*)
- the *test* part will be used for test (i.e. for *evaluation*)
- the fraction of test data will be _ts_ (a value of your choice between 0.2 and 0.5)
```

In [8]:

```
```

```
Training on 3509 examples
```

6. Loop on scores and, for each score, loop on the model labels

The function `GridSearchCV` iterates a cross validation experiment to train and test a model with different combinations of paramater values

- for each parameter we have set before a list of values to test, `ParametersGrid` will be implicitly called to generate all the combinations
- we choose a *score function* which will be used for the optimization
  - e.g. `accuracy_score` , `precision_score` , `recall_score` , `f1_score` , see this page for reference
- the output is a dataframe containing
  - the set of parameters maximising the score
  - the score used for optimisation and all the test scores

Steps

- prepare an empty list `clfs` to store all the fitte models
- prepare an empty DataFrame which will collect the results of the fittings with each combination of parameters
  - dataframe columns are `'scoring','model','best_params','accuracy','precision_macro','recall_macro','f1_macro'`
- loop

In [9]:

```

```

Parameters to collect

`classification_report` produces a dictionary containing some classification performance measures, given the *ground truth* and the *predictions* (use the parameter `output_dict=True`)

The measures are (among others):

- `accuracy`
- `macro avg` a dictionary containing:
  - `precision`

    - `recall`
    - `f1-score`
- ...

Loop

- repeat for all the chosen scorings
    - repeat for all the chosen classification models
        - store in `clf` the initialisation of `GridSearchCV` with the appropriate
            - classification model
            - parameters ranges
            - scoring
            - cross validation method `cv` (the same for all)
        - fit `clf` with the *train* part of `X` and `y`
        - store in `y_pred` the prediction for the *test* part of `X`
        - append `clf` to clfs`
        - append `y_pred` to `y_preds`
        - store in variable `cr` the `classification_report` produced with the test part of `y` and `y_pred`
        - store in the last row of `results` a list containing:
            - the name of the model
            - the `.best_params_` of `clf`
            - a selection of the contents of cr
                - 'accuracy',
                - 'macro avg''precision'
                - 'macro avg''recall'
                - 'macro avg''f1-score'

Hints:

- cr is a multi-level dictionary, second level can be reached with `cr['first level label']['second level label']`
- to append a list as the last row of a dataframe you can use `df.loc[len(df)]=[]`

In [11]:

```

```

7. Display

For each scoring show the ranking of the models, and the confusion matrix given by the best model

For each scoring:

- set a `scoring_filter`
- filter the results of that scoring
- display the filtered dataframe with the `display()` function (it allows several displays of dataframes)

In [12]:

```

```

Results for scoring "accuracy"

	model	best_params	accuracy	precision_macro	recall_macro	f1_macro
0	Decision Tree	{'class_weight': None, 'max_depth': 4}	0.872	0.800	0.700	0.734
1	Gaussian Naive Bayes	{'var_smoothing': 1e-11}	0.847	0.811	0.574	0.588

Results for scoring "precision_macro"

	model	best_params	accuracy	precision_macro	recall_macro	f1_macro
3	Gaussian Naive Bayes	{'var_smoothing': 1e-11}	0.847	0.811	0.574	0.588
2	Decision Tree	{'class_weight': None, 'max_depth': 4}	0.872	0.800	0.700	0.734

Results for scoring "recall_macro"

	model	best_params	accuracy	precision_macro	recall_macro	f1_macro
4	Decision Tree	{'class_weight': 'balanced', 'max_depth': 4}	0.724	0.649	0.744	0.651
5	Gaussian Naive Bayes	{'var_smoothing': 1e-11}	0.847	0.811	0.574	0.588

Results for scoring "f1_macro"

	model	best_params	accuracy	precision_macro	recall_macro	f1_macro

	model	best_params	accuracy	precision_macro	recall_macro	f1_macro
6	Decision Tree	{'class_weight': None, 'max_depth': 4}	0.872	0.800	0.700	0.734
7	Gaussian Naive Bayes	{'var_smoothing': 1e-11}	0.847	0.811	0.574	0.588

8. Confusion matrices

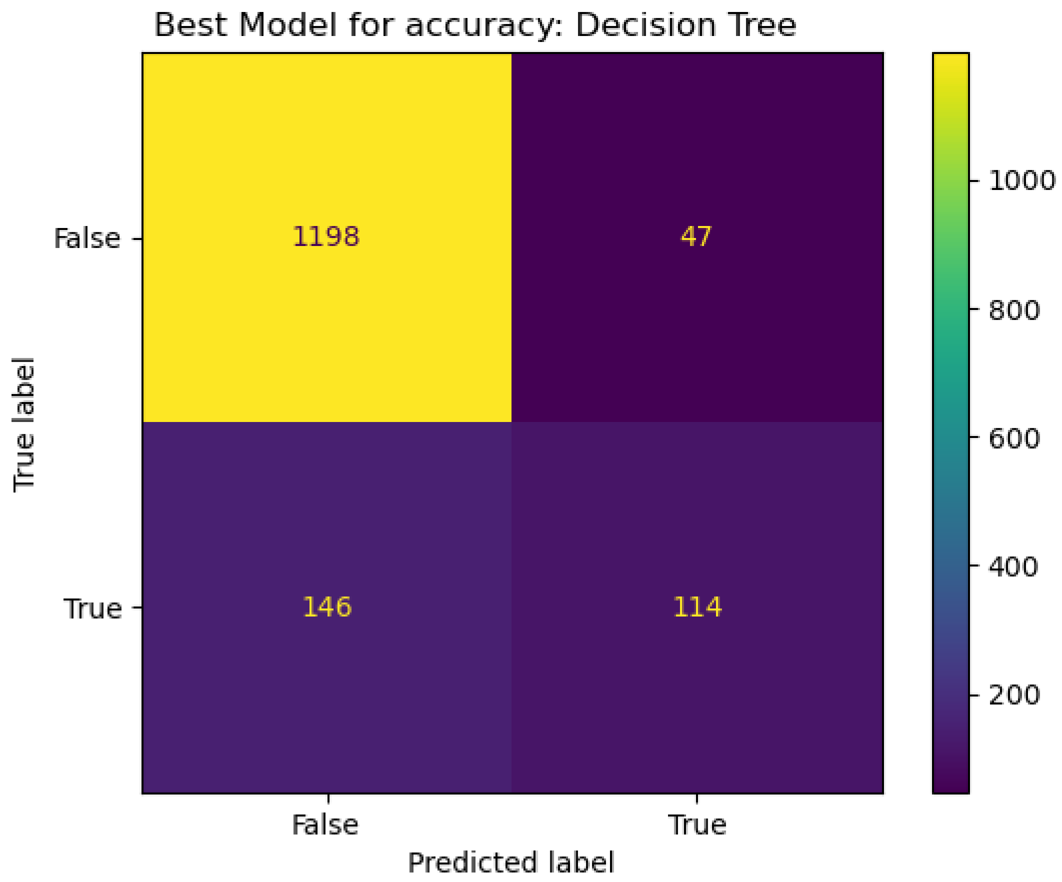
Use the `ConfusionMatrixDisplay` with the best model of each scoring to compare the predictions

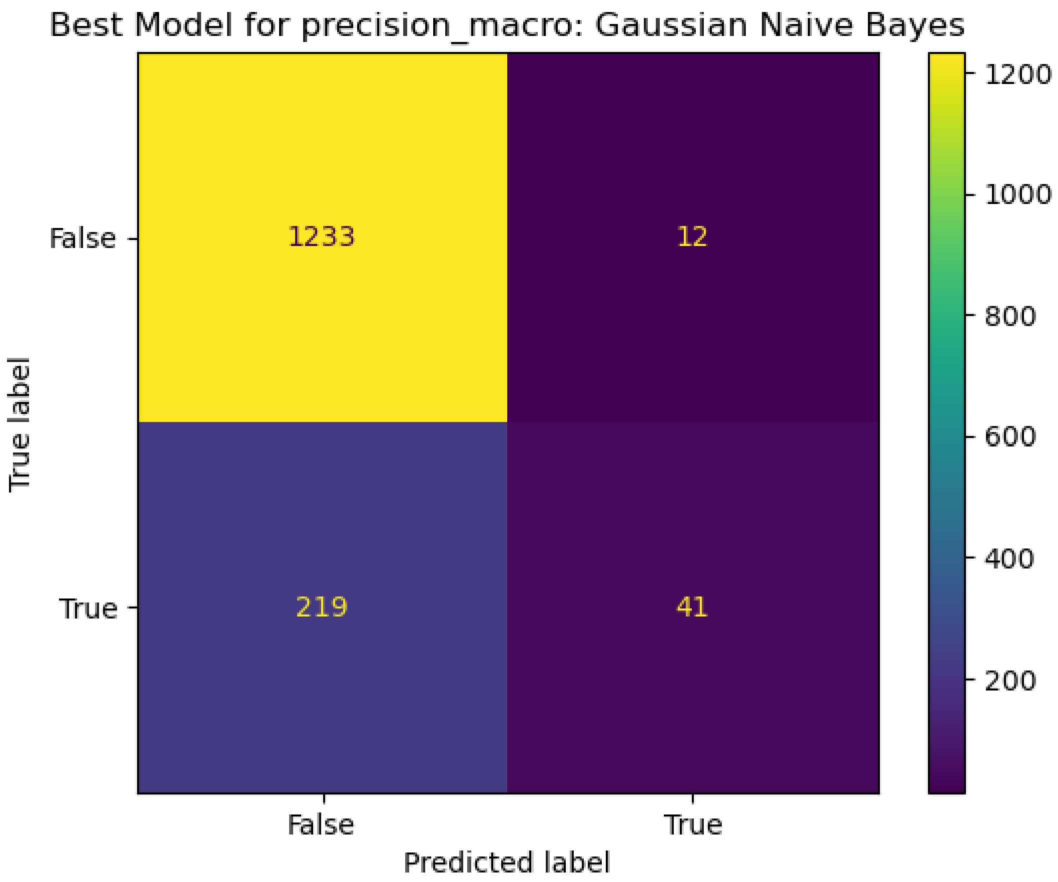
Repeat for every scoring:

- filter the results for the current scoring
- find the row with the best value of the scoring; this row is also the index of the corresponding
-

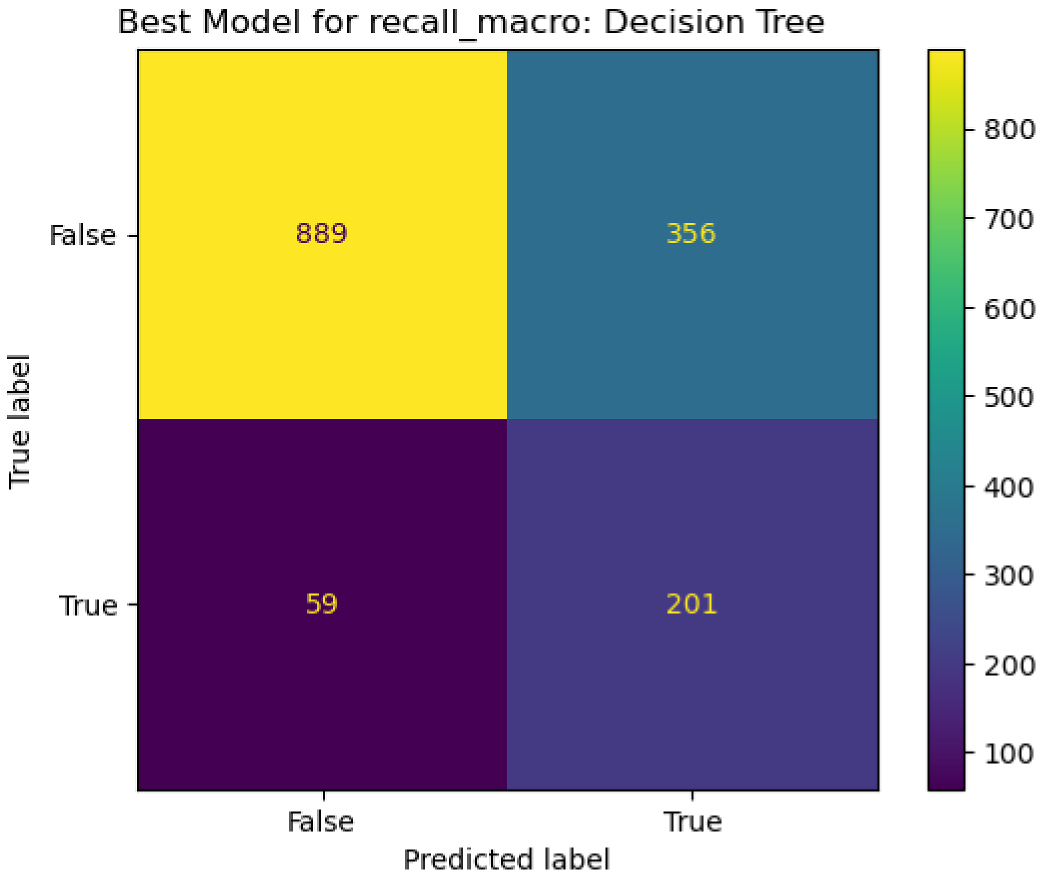
In [13]:

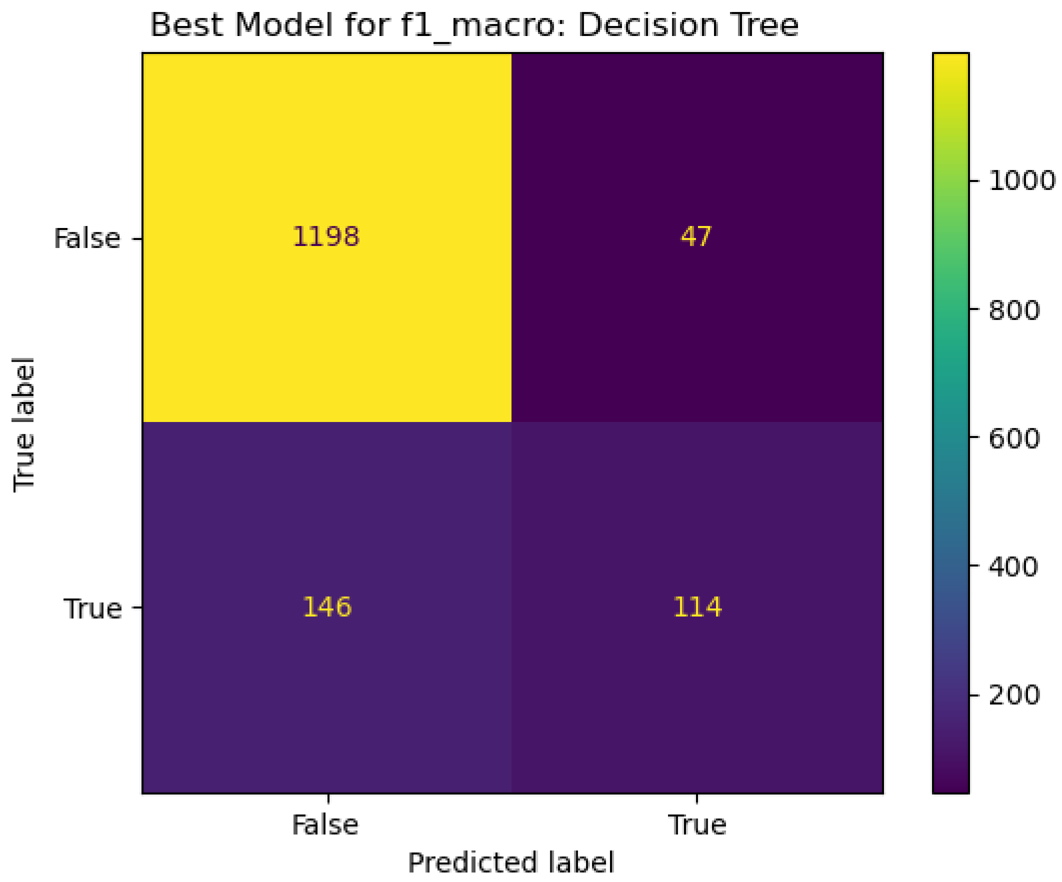
```

```


Best Model for accuracy: Decision Tree
True label
False
True
1198
47
146
114
False
True
Predicted label
1000
800
600
400
200



Best Model for precision_macro: Gaussian Naive Bayes
True label
False
True
1233
12
219
41
False
True
Predicted label
1200
1000
800
600
400
200


Best Model for recall_macro: Decision Tree
889
356
59
201
False
True
True label
False
True
Predicted label
800
700
600
500
400
300
200
100


Best Model for f1_macro: Decision Tree
True label
False
True
1198
47
146
114
Predicted label
False
True
1000
800
600
400
200