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. split the dataset into two parts: train and test
    - the *train* part will be used for training and 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)
3. the function `GridSearchCV` iterates a cross validation experiment to train and test a model with different combinations of paramater values
    - for each parameter we set a list of values to test, the function will generate all the combinations
    - we choose a *score function* which will be used for the optimization
        - e.g. `accuracy_score` , `precision_score` , `cohen_kappa_score` , `f1_score` , see this page for reference
    - the output is a dictionary containing
        - the set of parameters which maximize the score
        - the test scores
4. prepare the parameters for the grid
    - it is a list of dictionaries
5. set the parameters by cross validation and the *score functions* to choose from
6. Loop on scores and, for each score, loop on the model labels (see details below)

In [ ]:

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

from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
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
```

```
# Loading a standard dataset
#dataset = datasets.load_digits()
#dataset = datasets.fetch_olivetti_faces()
#dataset = datasets.fetch_covtype()
dataset = datasets.load_iris()
#dataset = datasets.load_wine()
#dataset = datasets.load_breast_cancer()
```

```
http://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digi
ts.html
@author: scikit-learn.org and Claudio Sartori
```

Prepare the environment

The `dataset` module contains, among others, a few sample datasets.

See this page for reference

Prepare the data and the target in X and y. Set `ts`. Set the random state to 42

In [ ]:

```
X = dataset.data
y = dataset.target
ts = 0.3
random_state = 42
```

Split the dataset into the train and test parts

In [ ]:

```

```

```
Training on 105 examples
```

The code below is intended to ease the remainder of the exercise

In [ ]:

```
model_lbls = [
              'dt',
              'nb',
              'lp',
              'svc',
             'knn',
             'adb',
             'rf',
            ]

# Set the parameters by cross-validation
tuned_param_dt = [{'max_depth': [*range(1,20)]}]
tuned_param_nb = [{'var_smoothing': [10, 1, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e
tuned_param_lp = [{'early_stopping': [True]}]
tuned_param_svc = [{'kernel': ['rbf'],
                    'gamma': [1e-3, 1e-4],
                    'C': [1, 10, 100, 1000],
                    },
                    {'kernel': ['linear'],
                     'C': [1, 10, 100, 1000],
                    },
                   ]
tuned_param_knn =[{'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}]
tuned_param_adb = [{'n_estimators':[20,30,40,50],
                   'learning_rate':[0.5,0.75,1,1.25,1.5]}]
tuned_param_rf = [{'max_depth': [*range(5,15)],
                   'n_estimators':[*range(10,100,10)]}]
```

```
models = {
    'dt': {'name': 'Decision Tree       ',
           'estimator': DecisionTreeClassifier(),
           'param': tuned_param_dt,
          },
    'nb': {'name': 'Gaussian Naive Bayes',
           'estimator': GaussianNB(),
           'param': tuned_param_nb
          },
    'lp': {'name': 'Linear Perceptron   ',
           'estimator': Perceptron(),
           'param': tuned_param_lp,
          },
    'svc':{'name': 'Support Vector      ',
           'estimator': SVC(),
           'param': tuned_param_svc
          },
    'knn':{'name': 'K Nearest Neighbor ',
           'estimator': KNeighborsClassifier(),
           'param': tuned_param_knn
       },
       'adb':{'name': 'AdaBoost           ',
           'estimator': AdaBoostClassifier(),
           'param': tuned_param_adb
          },
    'rf': {'name': 'Random forest       ',
           'estimator': RandomForestClassifier(),
           'param': tuned_param_rf
          }

}

scores = ['precision', 'recall']
```

The function below groups all the outputs

Write a function which has as parameter the fitted model and uses the components of the fitted model to inspect the results of the search with the parameters grid.

The components are:
`model.best_params_`
`model.cv_results_['mean_test_score']`
`model.cv_results_['std_test_score']`
`model.cv_results_['params']`

The classification report is generated by the function imported above from sklearn.metrics, which takes as argument the true and the predicted test labels.

The +/- in the results is obtained doubling the `std_test_score`

The function will be used to print the results for each set of parameters

In [ ]:
```
def print_results(model):
    print("Best parameters set found on train set:")
    print()
    # if best is linear there is no gamma parameter
    print(model.best_params_)
    print()
    print("Grid scores on train set:")
    print()
    means = model.cv_results_['mean_test_score']
    stds = model.cv_results_['std_test_score']
    params = model.cv_results_['params']
    for mean, std, params_tuple in zip(means, stds, params):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params_tuple))
    print()
    print("Detailed classification report for the best parameter set:")
    print()
    print("The model is trained on the full train set.")
    print("The scores are computed on the full test set.")
    print()
    y_true, y_pred = y_test, model.predict(X_test)
    print(classification_report(y_true, y_pred))
    print()
```

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

- iterate varying the score function
    1. iterate varying the classification model among Decision Tree, Naive Bayes, Linear Perceptron, Support Vector, AdaBoost, Random Forest and KNN
        - activate the *grid search*
            A. the resulting model will be the best one according to the current score function
        - print the best parameter set and the results for each set of parameters using the above defined function
        - print the classification report
        - store the `.best score_` in a dictionary for a final report
    2. print the final report for the current *score funtion*

In [ ]:

```
========================================
# Tuning hyper-parameters for precision

----------------------------------------
Trying model Decision Tree
Best parameters set found on train set:

{'max_depth': 14}

Grid scores on train set:

0.491 (+/-0.009) for {'max_depth': 1}
0.924 (+/-0.070) for {'max_depth': 2}
0.941 (+/-0.072) for {'max_depth': 3}
0.943 (+/-0.042) for {'max_depth': 4}
0.933 (+/-0.048) for {'max_depth': 5}
0.943 (+/-0.042) for {'max_depth': 6}
0.943 (+/-0.042) for {'max_depth': 7}
0.943 (+/-0.042) for {'max_depth': 8}
0.943 (+/-0.042) for {'max_depth': 9}
0.943 (+/-0.042) for {'max_depth': 10}
0.943 (+/-0.042) for {'max_depth': 11}
0.943 (+/-0.042) for {'max_depth': 12}
0.943 (+/-0.042) for {'max_depth': 13}
0.951 (+/-0.062) for {'max_depth': 14}
0.943 (+/-0.042) for {'max_depth': 15}
0.943 (+/-0.042) for {'max_depth': 16}
0.943 (+/-0.042) for {'max_depth': 17}
0.943 (+/-0.042) for {'max_depth': 18}
0.943 (+/-0.042) for {'max_depth': 19}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


----------------------------------------
Trying model Gaussian Naive Bayes
Best parameters set found on train set:

{'var_smoothing': 0.001}

Grid scores on train set:

0.762 (+/-0.352) for {'var_smoothing': 10}
0.920 (+/-0.131) for {'var_smoothing': 1}
0.911 (+/-0.150) for {'var_smoothing': 0.1}
0.933 (+/-0.090) for {'var_smoothing': 0.01}
0.941 (+/-0.072) for {'var_smoothing': 0.001}
0.941 (+/-0.072) for {'var_smoothing': 0.0001}
0.941 (+/-0.072) for {'var_smoothing': 1e-05}
0.941 (+/-0.072) for {'var_smoothing': 1e-06}
0.941 (+/-0.072) for {'var_smoothing': 1e-07}
```

```
0.941 (+/-0.072) for {'var_smoothing': 1e-08}
0.941 (+/-0.072) for {'var_smoothing': 1e-09}
0.941 (+/-0.072) for {'var_smoothing': 1e-10}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45


----------------------------------------
Trying model Linear Perceptron
Best parameters set found on train set:

{'early_stopping': True}

Grid scores on train set:

0.609 (+/-0.358) for {'early_stopping': True}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       0.00      0.00      0.00        19
           1       0.29      1.00      0.45        13
           2       0.00      0.00      0.00        13

    accuracy                           0.29        45
   macro avg       0.10      0.33      0.15        45
weighted avg       0.08      0.29      0.13        45


----------------------------------------
Trying model Support Vector
Best parameters set found on train set:

{'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}

Grid scores on train set:

0.322 (+/-0.349) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.171 (+/-0.240) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.940 (+/-0.108) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.322 (+/-0.349) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.964 (+/-0.063) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.940 (+/-0.108) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.978 (+/-0.055) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.964 (+/-0.063) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.970 (+/-0.052) for {'C': 1, 'kernel': 'linear'}
0.964 (+/-0.063) for {'C': 10, 'kernel': 'linear'}
```

```
0.964 (+/-0.063) for {'C': 100, 'kernel': 'linear'}
0.964 (+/-0.063) for {'C': 1000, 'kernel': 'linear'}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


----------------------------------------
Trying model K Nearest Neighbor
Best parameters set found on train set:

{'n_neighbors': 1}

Grid scores on train set:

0.963 (+/-0.043) for {'n_neighbors': 1}
0.947 (+/-0.061) for {'n_neighbors': 2}
0.950 (+/-0.054) for {'n_neighbors': 3}
0.950 (+/-0.054) for {'n_neighbors': 4}
0.956 (+/-0.052) for {'n_neighbors': 5}
0.953 (+/-0.060) for {'n_neighbors': 6}
0.963 (+/-0.043) for {'n_neighbors': 7}
0.963 (+/-0.043) for {'n_neighbors': 8}
0.957 (+/-0.049) for {'n_neighbors': 9}
0.947 (+/-0.061) for {'n_neighbors': 10}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


----------------------------------------
Trying model AdaBoost
Best parameters set found on train set:

{'learning_rate': 1.5, 'n_estimators': 20}

Grid scores on train set:

0.928 (+/-0.067) for {'learning_rate': 0.5, 'n_estimators': 20}
0.923 (+/-0.059) for {'learning_rate': 0.5, 'n_estimators': 30}
```

```
0.923 (+/-0.059) for {'learning_rate': 0.5, 'n_estimators': 40}
0.923 (+/-0.059) for {'learning_rate': 0.5, 'n_estimators': 50}
0.937 (+/-0.041) for {'learning_rate': 0.75, 'n_estimators': 20}
0.937 (+/-0.041) for {'learning_rate': 0.75, 'n_estimators': 30}
0.928 (+/-0.067) for {'learning_rate': 0.75, 'n_estimators': 40}
0.937 (+/-0.041) for {'learning_rate': 0.75, 'n_estimators': 50}
0.942 (+/-0.087) for {'learning_rate': 1, 'n_estimators': 20}
0.937 (+/-0.041) for {'learning_rate': 1, 'n_estimators': 30}
0.942 (+/-0.087) for {'learning_rate': 1, 'n_estimators': 40}
0.937 (+/-0.041) for {'learning_rate': 1, 'n_estimators': 50}
0.942 (+/-0.087) for {'learning_rate': 1.25, 'n_estimators': 20}
0.935 (+/-0.071) for {'learning_rate': 1.25, 'n_estimators': 30}
0.942 (+/-0.087) for {'learning_rate': 1.25, 'n_estimators': 40}
0.935 (+/-0.071) for {'learning_rate': 1.25, 'n_estimators': 50}
0.943 (+/-0.088) for {'learning_rate': 1.5, 'n_estimators': 20}
0.937 (+/-0.086) for {'learning_rate': 1.5, 'n_estimators': 30}
0.937 (+/-0.086) for {'learning_rate': 1.5, 'n_estimators': 40}
0.937 (+/-0.086) for {'learning_rate': 1.5, 'n_estimators': 50}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


----------------------------------------
Trying model Random forest
Best parameters set found on train set:

{'max_depth': 10, 'n_estimators': 30}

Grid scores on train set:

0.951 (+/-0.062) for {'max_depth': 5, 'n_estimators': 10}
0.952 (+/-0.062) for {'max_depth': 5, 'n_estimators': 20}
0.943 (+/-0.088) for {'max_depth': 5, 'n_estimators': 30}
0.943 (+/-0.088) for {'max_depth': 5, 'n_estimators': 40}
0.944 (+/-0.087) for {'max_depth': 5, 'n_estimators': 50}
0.952 (+/-0.062) for {'max_depth': 5, 'n_estimators': 60}
0.943 (+/-0.088) for {'max_depth': 5, 'n_estimators': 70}
0.943 (+/-0.088) for {'max_depth': 5, 'n_estimators': 80}
0.952 (+/-0.062) for {'max_depth': 5, 'n_estimators': 90}
0.952 (+/-0.062) for {'max_depth': 6, 'n_estimators': 10}
0.943 (+/-0.042) for {'max_depth': 6, 'n_estimators': 20}
0.943 (+/-0.088) for {'max_depth': 6, 'n_estimators': 30}
0.952 (+/-0.062) for {'max_depth': 6, 'n_estimators': 40}
0.952 (+/-0.062) for {'max_depth': 6, 'n_estimators': 50}
0.952 (+/-0.062) for {'max_depth': 6, 'n_estimators': 60}
0.952 (+/-0.062) for {'max_depth': 6, 'n_estimators': 70}
0.952 (+/-0.062) for {'max_depth': 6, 'n_estimators': 80}
0.952 (+/-0.062) for {'max_depth': 6, 'n_estimators': 90}
0.952 (+/-0.062) for {'max_depth': 7, 'n_estimators': 10}
0.944 (+/-0.087) for {'max_depth': 7, 'n_estimators': 20}
0.952 (+/-0.062) for {'max_depth': 7, 'n_estimators': 30}
```

```
0.943 (+/-0.088) for {'max_depth': 7, 'n_estimators': 40}
0.952 (+/-0.062) for {'max_depth': 7, 'n_estimators': 50}
0.952 (+/-0.062) for {'max_depth': 7, 'n_estimators': 60}
0.952 (+/-0.062) for {'max_depth': 7, 'n_estimators': 70}
0.952 (+/-0.062) for {'max_depth': 7, 'n_estimators': 80}
0.952 (+/-0.062) for {'max_depth': 7, 'n_estimators': 90}
0.924 (+/-0.089) for {'max_depth': 8, 'n_estimators': 10}
0.935 (+/-0.071) for {'max_depth': 8, 'n_estimators': 20}
0.952 (+/-0.062) for {'max_depth': 8, 'n_estimators': 30}
0.952 (+/-0.062) for {'max_depth': 8, 'n_estimators': 40}
0.952 (+/-0.062) for {'max_depth': 8, 'n_estimators': 50}
0.952 (+/-0.062) for {'max_depth': 8, 'n_estimators': 60}
0.952 (+/-0.062) for {'max_depth': 8, 'n_estimators': 70}
0.952 (+/-0.062) for {'max_depth': 8, 'n_estimators': 80}
0.952 (+/-0.062) for {'max_depth': 8, 'n_estimators': 90}
0.954 (+/-0.028) for {'max_depth': 9, 'n_estimators': 10}
0.952 (+/-0.062) for {'max_depth': 9, 'n_estimators': 20}
0.952 (+/-0.062) for {'max_depth': 9, 'n_estimators': 30}
0.952 (+/-0.062) for {'max_depth': 9, 'n_estimators': 40}
0.934 (+/-0.122) for {'max_depth': 9, 'n_estimators': 50}
0.952 (+/-0.062) for {'max_depth': 9, 'n_estimators': 60}
0.943 (+/-0.042) for {'max_depth': 9, 'n_estimators': 70}
0.952 (+/-0.062) for {'max_depth': 9, 'n_estimators': 80}
0.952 (+/-0.062) for {'max_depth': 9, 'n_estimators': 90}
0.937 (+/-0.087) for {'max_depth': 10, 'n_estimators': 10}
0.943 (+/-0.042) for {'max_depth': 10, 'n_estimators': 20}
0.962 (+/-0.047) for {'max_depth': 10, 'n_estimators': 30}
0.952 (+/-0.062) for {'max_depth': 10, 'n_estimators': 40}
0.952 (+/-0.062) for {'max_depth': 10, 'n_estimators': 50}
0.952 (+/-0.062) for {'max_depth': 10, 'n_estimators': 60}
0.952 (+/-0.062) for {'max_depth': 10, 'n_estimators': 70}
0.952 (+/-0.062) for {'max_depth': 10, 'n_estimators': 80}
0.952 (+/-0.062) for {'max_depth': 10, 'n_estimators': 90}
0.948 (+/-0.030) for {'max_depth': 11, 'n_estimators': 10}
0.956 (+/-0.052) for {'max_depth': 11, 'n_estimators': 20}
0.943 (+/-0.042) for {'max_depth': 11, 'n_estimators': 30}
0.952 (+/-0.062) for {'max_depth': 11, 'n_estimators': 40}
0.943 (+/-0.042) for {'max_depth': 11, 'n_estimators': 50}
0.952 (+/-0.062) for {'max_depth': 11, 'n_estimators': 60}
0.943 (+/-0.088) for {'max_depth': 11, 'n_estimators': 70}
0.952 (+/-0.062) for {'max_depth': 11, 'n_estimators': 80}
0.952 (+/-0.062) for {'max_depth': 11, 'n_estimators': 90}
0.943 (+/-0.042) for {'max_depth': 12, 'n_estimators': 10}
0.952 (+/-0.062) for {'max_depth': 12, 'n_estimators': 20}
0.952 (+/-0.062) for {'max_depth': 12, 'n_estimators': 30}
0.943 (+/-0.088) for {'max_depth': 12, 'n_estimators': 40}
0.952 (+/-0.062) for {'max_depth': 12, 'n_estimators': 50}
0.952 (+/-0.062) for {'max_depth': 12, 'n_estimators': 60}
0.952 (+/-0.062) for {'max_depth': 12, 'n_estimators': 70}
0.952 (+/-0.062) for {'max_depth': 12, 'n_estimators': 80}
0.952 (+/-0.062) for {'max_depth': 12, 'n_estimators': 90}
0.943 (+/-0.088) for {'max_depth': 13, 'n_estimators': 10}
0.935 (+/-0.071) for {'max_depth': 13, 'n_estimators': 20}
0.952 (+/-0.062) for {'max_depth': 13, 'n_estimators': 30}
0.952 (+/-0.062) for {'max_depth': 13, 'n_estimators': 40}
0.952 (+/-0.062) for {'max_depth': 13, 'n_estimators': 50}
0.952 (+/-0.062) for {'max_depth': 13, 'n_estimators': 60}
0.952 (+/-0.062) for {'max_depth': 13, 'n_estimators': 70}
0.952 (+/-0.062) for {'max_depth': 13, 'n_estimators': 80}
0.952 (+/-0.062) for {'max_depth': 13, 'n_estimators': 90}
0.952 (+/-0.062) for {'max_depth': 14, 'n_estimators': 10}
0.952 (+/-0.062) for {'max_depth': 14, 'n_estimators': 20}
0.948 (+/-0.030) for {'max_depth': 14, 'n_estimators': 30}
0.943 (+/-0.088) for {'max_depth': 14, 'n_estimators': 40}
```

```
0.943 (+/-0.088) for {'max_depth': 14, 'n_estimators': 50}
0.952 (+/-0.062) for {'max_depth': 14, 'n_estimators': 60}
0.943 (+/-0.088) for {'max_depth': 14, 'n_estimators': 70}
0.952 (+/-0.062) for {'max_depth': 14, 'n_estimators': 80}
0.952 (+/-0.062) for {'max_depth': 14, 'n_estimators': 90}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


Summary of results for precision
Estimator
Decision Tree            - score: 0.95%
Gaussian Naive Bayes     - score: 0.94%
Linear Perceptron        - score: 0.61%
Support Vector           - score: 0.98%
K Nearest Neighbor       - score: 0.96%
AdaBoost                 - score: 0.94%
Random forest            - score: 0.96%
========================================
# Tuning hyper-parameters for recall

----------------------------------------
Trying model Decision Tree
Best parameters set found on train set:

{'max_depth': 4}

Grid scores on train set:

0.667 (+/-0.000) for {'max_depth': 1}
0.920 (+/-0.065) for {'max_depth': 2}
0.937 (+/-0.071) for {'max_depth': 3}
0.938 (+/-0.040) for {'max_depth': 4}
0.929 (+/-0.045) for {'max_depth': 5}
0.938 (+/-0.040) for {'max_depth': 6}
0.938 (+/-0.040) for {'max_depth': 7}
0.938 (+/-0.040) for {'max_depth': 8}
0.938 (+/-0.040) for {'max_depth': 9}
0.938 (+/-0.040) for {'max_depth': 10}
0.938 (+/-0.040) for {'max_depth': 11}
0.938 (+/-0.040) for {'max_depth': 12}
0.938 (+/-0.040) for {'max_depth': 13}
0.938 (+/-0.040) for {'max_depth': 14}
0.938 (+/-0.040) for {'max_depth': 15}
0.938 (+/-0.040) for {'max_depth': 16}
0.938 (+/-0.040) for {'max_depth': 17}
0.938 (+/-0.040) for {'max_depth': 18}
0.938 (+/-0.040) for {'max_depth': 19}

Detailed classification report for the best parameter set:
```

```
The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


----------------------------------------
Trying model Gaussian Naive Bayes
Best parameters set found on train set:

{'var_smoothing': 0.001}

Grid scores on train set:

0.716 (+/-0.224) for {'var_smoothing': 10}
0.918 (+/-0.134) for {'var_smoothing': 1}
0.908 (+/-0.152) for {'var_smoothing': 0.1}
0.927 (+/-0.092) for {'var_smoothing': 0.01}
0.937 (+/-0.071) for {'var_smoothing': 0.001}
0.937 (+/-0.071) for {'var_smoothing': 0.0001}
0.937 (+/-0.071) for {'var_smoothing': 1e-05}
0.937 (+/-0.071) for {'var_smoothing': 1e-06}
0.937 (+/-0.071) for {'var_smoothing': 1e-07}
0.937 (+/-0.071) for {'var_smoothing': 1e-08}
0.937 (+/-0.071) for {'var_smoothing': 1e-09}
0.937 (+/-0.071) for {'var_smoothing': 1e-10}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.92      0.96        13
           2       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45


----------------------------------------
Trying model Linear Perceptron
Best parameters set found on train set:

{'early_stopping': True}

Grid scores on train set:

0.656 (+/-0.193) for {'early_stopping': True}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       0.00      0.00      0.00        19
           1       0.29      1.00      0.45        13
           2       0.00      0.00      0.00        13

    accuracy                           0.29        45
   macro avg       0.10      0.33      0.15        45
weighted avg       0.08      0.29      0.13        45


----------------------------------------
Trying model Support Vector
Best parameters set found on train set:

{'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}

Grid scores on train set:

0.505 (+/-0.299) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.371 (+/-0.152) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.937 (+/-0.111) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.505 (+/-0.299) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.955 (+/-0.080) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.937 (+/-0.111) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.973 (+/-0.075) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.955 (+/-0.080) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.963 (+/-0.071) for {'C': 1, 'kernel': 'linear'}
0.955 (+/-0.080) for {'C': 10, 'kernel': 'linear'}
0.955 (+/-0.080) for {'C': 100, 'kernel': 'linear'}
0.955 (+/-0.080) for {'C': 1000, 'kernel': 'linear'}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


----------------------------------------
Trying model K Nearest Neighbor
Best parameters set found on train set:

{'n_neighbors': 1}

Grid scores on train set:

0.954 (+/-0.060) for {'n_neighbors': 1}
0.935 (+/-0.075) for {'n_neighbors': 2}
0.936 (+/-0.073) for {'n_neighbors': 3}
0.935 (+/-0.078) for {'n_neighbors': 4}
0.945 (+/-0.067) for {'n_neighbors': 5}
0.944 (+/-0.069) for {'n_neighbors': 6}
```

```
0.954 (+/-0.060) for {'n_neighbors': 7}
0.954 (+/-0.060) for {'n_neighbors': 8}
0.944 (+/-0.072) for {'n_neighbors': 9}
0.935 (+/-0.075) for {'n_neighbors': 10}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


----------------------------------------
Trying model AdaBoost
Best parameters set found on train set:

{'learning_rate': 1, 'n_estimators': 20}

Grid scores on train set:

0.920 (+/-0.065) for {'learning_rate': 0.5, 'n_estimators': 20}
0.911 (+/-0.057) for {'learning_rate': 0.5, 'n_estimators': 30}
0.911 (+/-0.057) for {'learning_rate': 0.5, 'n_estimators': 40}
0.911 (+/-0.057) for {'learning_rate': 0.5, 'n_estimators': 50}
0.929 (+/-0.045) for {'learning_rate': 0.75, 'n_estimators': 20}
0.929 (+/-0.045) for {'learning_rate': 0.75, 'n_estimators': 30}
0.920 (+/-0.065) for {'learning_rate': 0.75, 'n_estimators': 40}
0.929 (+/-0.045) for {'learning_rate': 0.75, 'n_estimators': 50}
0.939 (+/-0.088) for {'learning_rate': 1, 'n_estimators': 20}
0.929 (+/-0.045) for {'learning_rate': 1, 'n_estimators': 30}
0.939 (+/-0.088) for {'learning_rate': 1, 'n_estimators': 40}
0.929 (+/-0.045) for {'learning_rate': 1, 'n_estimators': 50}
0.939 (+/-0.088) for {'learning_rate': 1.25, 'n_estimators': 20}
0.930 (+/-0.068) for {'learning_rate': 1.25, 'n_estimators': 30}
0.939 (+/-0.088) for {'learning_rate': 1.25, 'n_estimators': 40}
0.930 (+/-0.068) for {'learning_rate': 1.25, 'n_estimators': 50}
0.938 (+/-0.087) for {'learning_rate': 1.5, 'n_estimators': 20}
0.929 (+/-0.089) for {'learning_rate': 1.5, 'n_estimators': 30}
0.929 (+/-0.089) for {'learning_rate': 1.5, 'n_estimators': 40}
0.929 (+/-0.089) for {'learning_rate': 1.5, 'n_estimators': 50}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

```
----------------------------------------
Trying model Random forest
Best parameters set found on train set:

{'max_depth': 7, 'n_estimators': 70}

Grid scores on train set:

0.946 (+/-0.064) for {'max_depth': 5, 'n_estimators': 10}
0.946 (+/-0.064) for {'max_depth': 5, 'n_estimators': 20}
0.938 (+/-0.087) for {'max_depth': 5, 'n_estimators': 30}
0.938 (+/-0.040) for {'max_depth': 5, 'n_estimators': 40}
0.938 (+/-0.040) for {'max_depth': 5, 'n_estimators': 50}
0.946 (+/-0.064) for {'max_depth': 5, 'n_estimators': 60}
0.946 (+/-0.064) for {'max_depth': 5, 'n_estimators': 70}
0.946 (+/-0.064) for {'max_depth': 5, 'n_estimators': 80}
0.938 (+/-0.087) for {'max_depth': 5, 'n_estimators': 90}
0.946 (+/-0.064) for {'max_depth': 6, 'n_estimators': 10}
0.946 (+/-0.064) for {'max_depth': 6, 'n_estimators': 20}
0.946 (+/-0.064) for {'max_depth': 6, 'n_estimators': 30}
0.946 (+/-0.064) for {'max_depth': 6, 'n_estimators': 40}
0.946 (+/-0.064) for {'max_depth': 6, 'n_estimators': 50}
0.946 (+/-0.064) for {'max_depth': 6, 'n_estimators': 60}
0.946 (+/-0.064) for {'max_depth': 6, 'n_estimators': 70}
0.946 (+/-0.064) for {'max_depth': 6, 'n_estimators': 80}
0.946 (+/-0.064) for {'max_depth': 6, 'n_estimators': 90}
0.937 (+/-0.071) for {'max_depth': 7, 'n_estimators': 10}
0.938 (+/-0.040) for {'max_depth': 7, 'n_estimators': 20}
0.946 (+/-0.064) for {'max_depth': 7, 'n_estimators': 30}
0.946 (+/-0.064) for {'max_depth': 7, 'n_estimators': 40}
0.946 (+/-0.064) for {'max_depth': 7, 'n_estimators': 50}
0.946 (+/-0.064) for {'max_depth': 7, 'n_estimators': 60}
0.956 (+/-0.077) for {'max_depth': 7, 'n_estimators': 70}
0.946 (+/-0.064) for {'max_depth': 7, 'n_estimators': 80}
0.946 (+/-0.064) for {'max_depth': 7, 'n_estimators': 90}
0.955 (+/-0.053) for {'max_depth': 8, 'n_estimators': 10}
0.946 (+/-0.064) for {'max_depth': 8, 'n_estimators': 20}
0.956 (+/-0.077) for {'max_depth': 8, 'n_estimators': 30}
0.946 (+/-0.064) for {'max_depth': 8, 'n_estimators': 40}
0.938 (+/-0.040) for {'max_depth': 8, 'n_estimators': 50}
0.946 (+/-0.064) for {'max_depth': 8, 'n_estimators': 60}
0.946 (+/-0.064) for {'max_depth': 8, 'n_estimators': 70}
0.946 (+/-0.064) for {'max_depth': 8, 'n_estimators': 80}
0.946 (+/-0.064) for {'max_depth': 8, 'n_estimators': 90}
0.946 (+/-0.064) for {'max_depth': 9, 'n_estimators': 10}
0.955 (+/-0.080) for {'max_depth': 9, 'n_estimators': 20}
0.938 (+/-0.040) for {'max_depth': 9, 'n_estimators': 30}
0.938 (+/-0.087) for {'max_depth': 9, 'n_estimators': 40}
0.938 (+/-0.087) for {'max_depth': 9, 'n_estimators': 50}
0.946 (+/-0.064) for {'max_depth': 9, 'n_estimators': 60}
0.946 (+/-0.064) for {'max_depth': 9, 'n_estimators': 70}
0.946 (+/-0.064) for {'max_depth': 9, 'n_estimators': 80}
0.938 (+/-0.040) for {'max_depth': 9, 'n_estimators': 90}
0.946 (+/-0.064) for {'max_depth': 10, 'n_estimators': 10}
0.930 (+/-0.068) for {'max_depth': 10, 'n_estimators': 20}
0.955 (+/-0.053) for {'max_depth': 10, 'n_estimators': 30}
0.946 (+/-0.064) for {'max_depth': 10, 'n_estimators': 40}
0.946 (+/-0.064) for {'max_depth': 10, 'n_estimators': 50}
0.946 (+/-0.064) for {'max_depth': 10, 'n_estimators': 60}
0.938 (+/-0.087) for {'max_depth': 10, 'n_estimators': 70}
0.938 (+/-0.087) for {'max_depth': 10, 'n_estimators': 80}
0.946 (+/-0.064) for {'max_depth': 10, 'n_estimators': 90}
```

```
0.945 (+/-0.067) for {'max_depth': 11, 'n_estimators': 10}
0.946 (+/-0.067) for {'max_depth': 11, 'n_estimators': 20}
0.946 (+/-0.064) for {'max_depth': 11, 'n_estimators': 30}
0.946 (+/-0.064) for {'max_depth': 11, 'n_estimators': 40}
0.938 (+/-0.040) for {'max_depth': 11, 'n_estimators': 50}
0.948 (+/-0.101) for {'max_depth': 11, 'n_estimators': 60}
0.938 (+/-0.087) for {'max_depth': 11, 'n_estimators': 70}
0.938 (+/-0.040) for {'max_depth': 11, 'n_estimators': 80}
0.938 (+/-0.040) for {'max_depth': 11, 'n_estimators': 90}
0.946 (+/-0.064) for {'max_depth': 12, 'n_estimators': 10}
0.945 (+/-0.067) for {'max_depth': 12, 'n_estimators': 20}
0.938 (+/-0.087) for {'max_depth': 12, 'n_estimators': 30}
0.937 (+/-0.044) for {'max_depth': 12, 'n_estimators': 40}
0.938 (+/-0.040) for {'max_depth': 12, 'n_estimators': 50}
0.946 (+/-0.064) for {'max_depth': 12, 'n_estimators': 60}
0.938 (+/-0.087) for {'max_depth': 12, 'n_estimators': 70}
0.946 (+/-0.064) for {'max_depth': 12, 'n_estimators': 80}
0.946 (+/-0.064) for {'max_depth': 12, 'n_estimators': 90}
0.938 (+/-0.040) for {'max_depth': 13, 'n_estimators': 10}
0.946 (+/-0.064) for {'max_depth': 13, 'n_estimators': 20}
0.946 (+/-0.030) for {'max_depth': 13, 'n_estimators': 30}
0.946 (+/-0.064) for {'max_depth': 13, 'n_estimators': 40}
0.946 (+/-0.064) for {'max_depth': 13, 'n_estimators': 50}
0.938 (+/-0.087) for {'max_depth': 13, 'n_estimators': 60}
0.946 (+/-0.064) for {'max_depth': 13, 'n_estimators': 70}
0.946 (+/-0.064) for {'max_depth': 13, 'n_estimators': 80}
0.946 (+/-0.064) for {'max_depth': 13, 'n_estimators': 90}
0.946 (+/-0.064) for {'max_depth': 14, 'n_estimators': 10}
0.955 (+/-0.053) for {'max_depth': 14, 'n_estimators': 20}
0.946 (+/-0.064) for {'max_depth': 14, 'n_estimators': 30}
0.946 (+/-0.064) for {'max_depth': 14, 'n_estimators': 40}
0.956 (+/-0.077) for {'max_depth': 14, 'n_estimators': 50}
0.946 (+/-0.064) for {'max_depth': 14, 'n_estimators': 60}
0.946 (+/-0.064) for {'max_depth': 14, 'n_estimators': 70}
0.946 (+/-0.064) for {'max_depth': 14, 'n_estimators': 80}
0.946 (+/-0.064) for {'max_depth': 14, 'n_estimators': 90}

Detailed classification report for the best parameter set:

The model is trained on the full train set.
The scores are computed on the full test set.

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45


Summary of results for recall
Estimator
Decision Tree            - score: 0.94%
Gaussian Naive Bayes     - score: 0.94%
Linear Perceptron        - score: 0.66%
Support Vector           - score: 0.97%
K Nearest Neighbor       - score: 0.95%
AdaBoost                 - score: 0.94%
Random forest            - score: 0.96%
```

In [ ]: