sklearn-evaluation Documentation

Release 0.5.9dev

Eduardo Blancas Reyes

Apr 16, 2022

CONTENTS

scikit-learn model evaluation made easy: plots, tables and markdown reports.

CHAPTER

ONE

INSTALLATION

```
pip install sklearn-evaluation
```

CHAPTER

TWO

TABLE OF CONTENTS

2.1 User Guide

2.1.1 Classifier evaluation

sklearn-evluation has two main modules for evaluating classifiers: *sklearn_evaluation.plot* and *sklearn_evaluation.table*, let's see an example of how to use them.

First, let's load some data and split it in training and test set.

```
In [1]: data = datasets.make_classification(200, 10, n_informative=5,
   ...:                                     class_sep=0.65)
   ...:

In [2]: X = data[0]

In [3]: y = data[1]

# shuffle and split training and test sets
In [4]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Now, we are going to train the data using one of the scikit-learn classifiers.

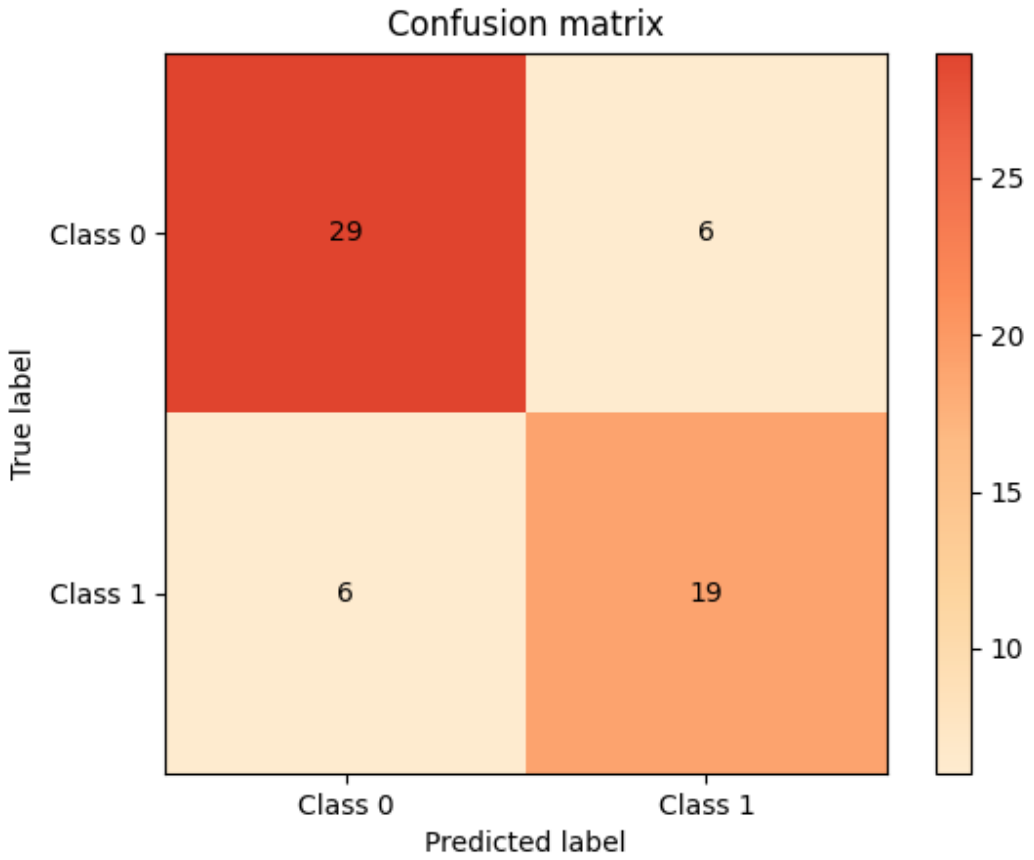
```
In [5]: est = RandomForestClassifier(n_estimators=5)

In [6]: est.fit(X_train, y_train)
Out[6]: RandomForestClassifier(n_estimators=5)
```

Most of the functions require us to pass the class predictions for the test set (`y_pred`), the scores assigned (`y_score`) and the ground truth classes (`y_true`), let's define such variables.

```
In [7]: y_pred = est.predict(X_test)

In [8]: y_score = est.predict_proba(X_test)

In [9]: y_true = y_test
```

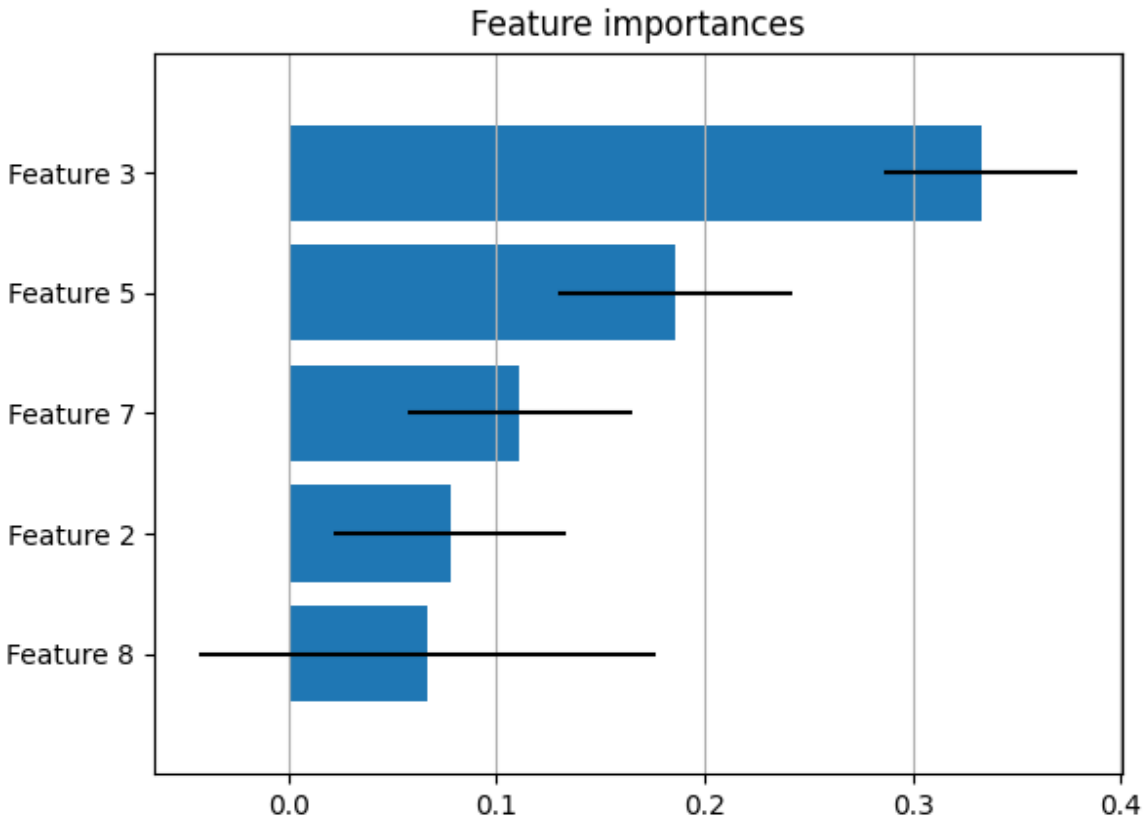
We can start evaluating our model, the following example shows how to plot a confusion matrix.

```
In [10]: plot.confusion_matrix(y_true, y_pred)
Out[10]: <AxesSubplot:title={'center':'Confusion matrix'}, xlabel='Predicted label',␣
↪ylabel='True label'>
```



Some classifiers (such as `sklearn.ensemble.RandomForestClassifier`) have feature importances, we can plot then passing the estimator object to the feature_importances function.

```
In [11]: plot.feature_importances(est, top_n=5)
Out[11]: <AxesSubplot:title={'center':'Feature importances'}>
```



A feature importances function is also available in the table module.

```
In [12]: print(table.feature_importances(est))
+----------------+--------------+-----------+
| feature_name   |   importance |      std_ |
+================+==============+===========+
| Feature 3      |    0.332923  | 0.109856  |
+----------------+--------------+-----------+
| Feature 5      |    0.186159  | 0.0561236 |
+----------------+--------------+-----------+
| Feature 7      |    0.111431  | 0.0536574 |
+----------------+--------------+-----------+
| Feature 2      |    0.0777974 | 0.0560804 |
+----------------+--------------+-----------+
| Feature 8      |    0.0666351 | 0.0466724 |
+----------------+--------------+-----------+
| Feature 1      |    0.0643182 | 0.0435577 |
+----------------+--------------+-----------+
| Feature 6      |    0.0542521 | 0.0639939 |
+----------------+--------------+-----------+
| Feature 9      |    0.0508333 | 0.0597737 |
+----------------+--------------+-----------+
| Feature 4      |    0.0338024 | 0.0443134 |
```

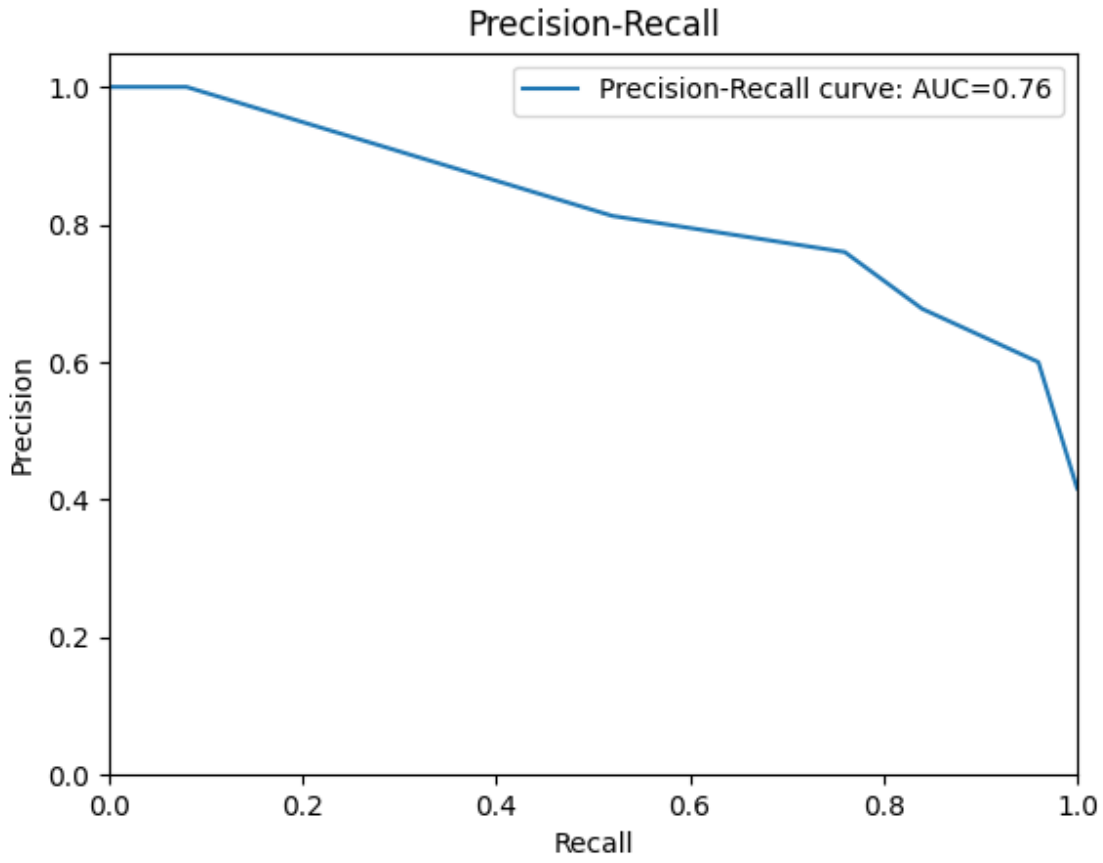
(continues on next page)

(continued from previous page)

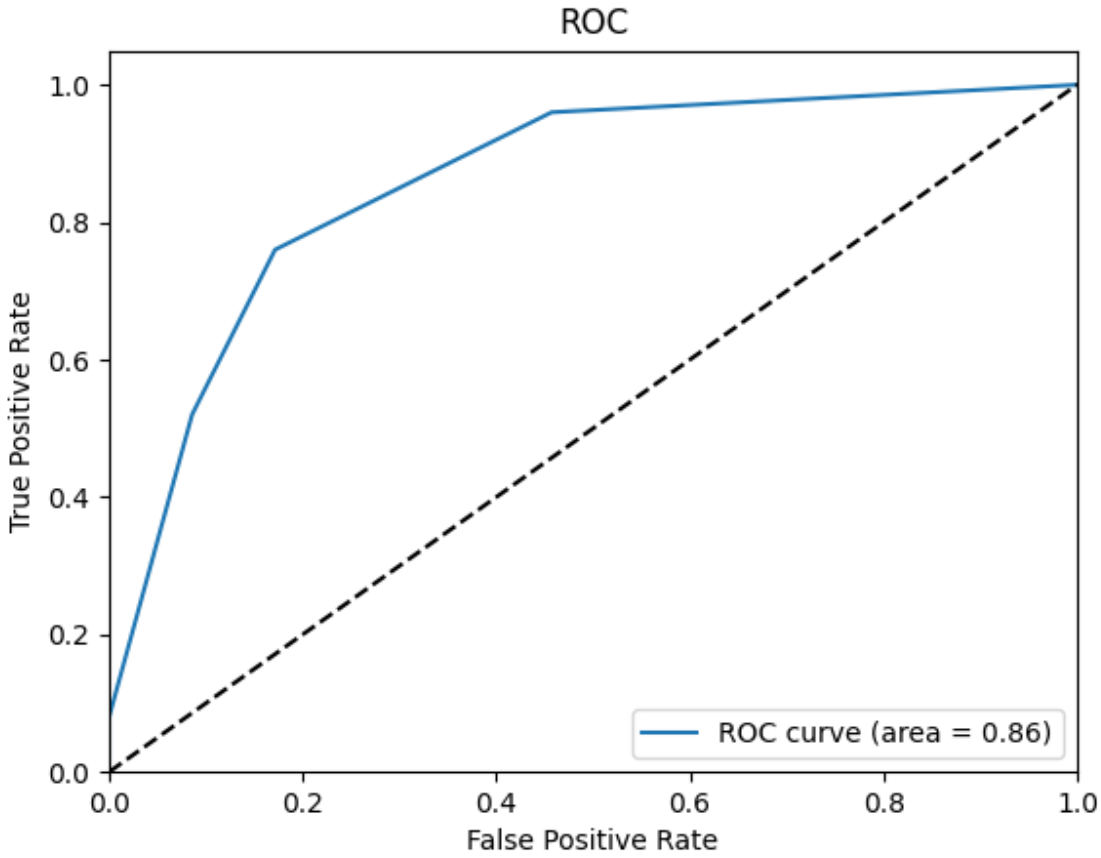
```
+----------------+--------------+-----------+
| Feature 10     |    0.0218483 | 0.0242025 |
+----------------+--------------+-----------+
```

Now, let's see how to generate two of the most common plots for evaluating classifiers: Precision-Recall and ROC.

```
In [13]: plot.precision_recall(y_true, y_score)
Out[13]: <AxesSubplot:title={'center':'Precision-Recall'}, xlabel='Recall', ylabel=
↪'Precision'>
```




```
In [14]: plot.roc(y_true, y_score)
Out[14]: <AxesSubplot:title={'center':'ROC'}, xlabel='False Positive Rate', ylabel='True␣
↪Positive Rate'>
```



2.1.2 Classifier evaluation using the OOP interface (report generation)

We can also use the *sklearn_evaluation.ClassifierEvaluator* class to pack the results from our estimator. This way we can generate plots and tables without having to pass the parameters over and over again. If we are evaluating more than one model at a time this also gives us a way to keep it organized. Furthermore, the ClassifierEvaluator class offers a way to create HTML reports from our model results.

First, let's load some data and split for training and testing.

```
In [1]: iris = datasets.load_iris()

In [2]: X = iris.data

In [3]: y = iris.target

In [4]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Let's now train a classifier and predict on the test set.

```
In [5]: est = RandomForestClassifier(n_estimators=5)

In [6]: est.fit(X_train, y_train)
Out[6]: RandomForestClassifier(n_estimators=5)
```

(continues on next page)

(continued from previous page)

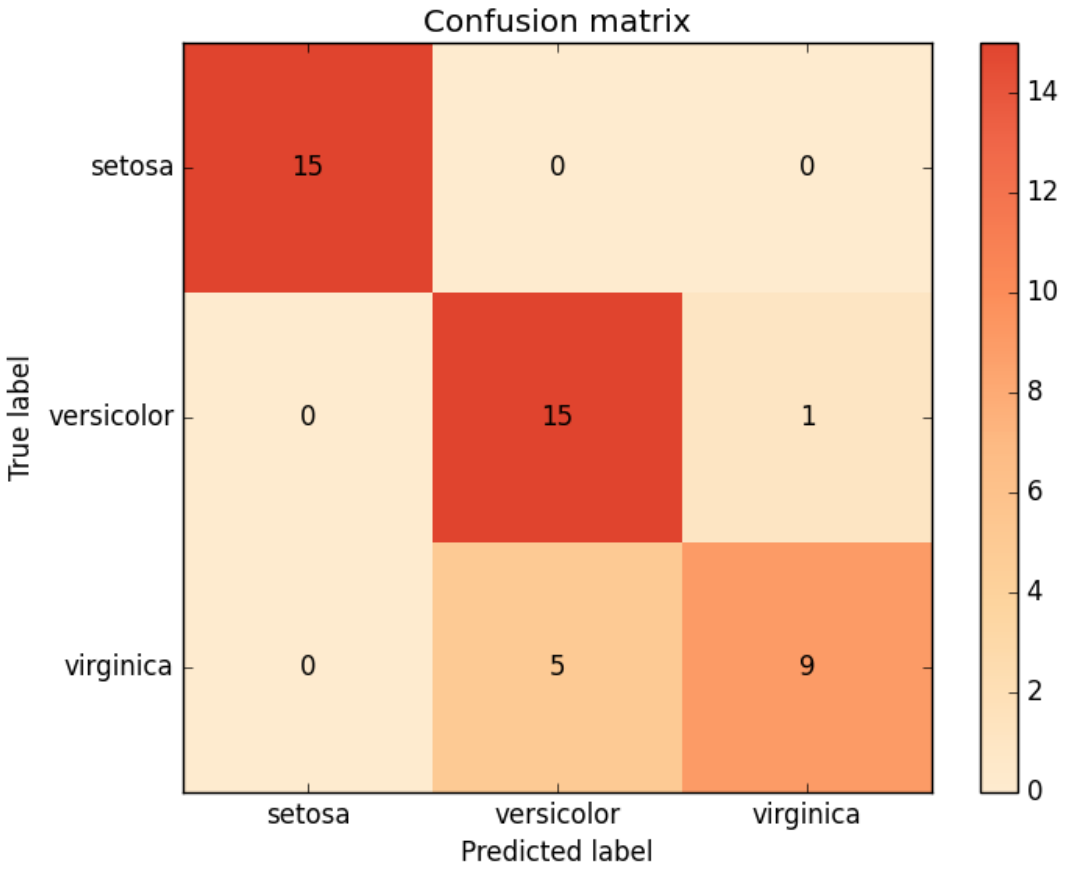
```
In [7]: y_pred = est.predict(X_test)

In [8]: y_score = est.predict_proba(X_test)

In [9]: feature_list = range(4)

In [10]: target_names = ['setosa', 'versicolor', 'virginica']
```

Now that we have everything we need, let's pack our results using ClassifierEvaluator, every parameter is optional.

```
In [11]: ce = ClassifierEvaluator(est, y_test, y_pred, y_score,
   ....:                          feature_list, target_names,
   ....:                          estimator_name='RF')
   ....:
```

We can use most of the functions in plot and table directly from the ClassifierEvaluator object, let's see how to plot a confusion matrix.

```
In [12]: ce.confusion_matrix()
Out[12]: <AxesSubplot:title={'center':'Confusion matrix'}, xlabel='Predicted label',␣
↪ylabel='True label'>
```



We can also generate HTML reports from our models by using the make_report function. The first parameter is a HTML or Markdown template with jinja2 format. If a pathlib.Path object is passed, the content of the file is read. Within the template, the evaluator is passed as "e", so you can use things like {{e.confusion_matrix()}} or any other attribute/method. If None, a default template is used

```
In [13]: report = ce.make_report()
```

The function returns a Report object, which will automatically render in a Jupyter notebook, *report.save('/path/to/report.html')* will save the report and *report.rendered* will return a string with the HTML report.

2.1.3 Functional vs Object Oriented interface

Why having two different ways of doing the same? While the plot/table module can be accessed directly or via *sklearn_evaluation.ClassifierEvaluator*, they serve slighly different purposes.

The purpose of *sklearn_evaluation.ClassifierEvaluator* is to provide a simpler API where you can quickly plot and evaluate a model(s) and generate reports from them (right now the only way of generating reports is to use the OOP interface). Since the OOP is simpler, it also has some constraints. When plotting a confusion matrix from the plot module, you can pass a `matplotlib.axes.Axes` object which gives you great flexibility, you can use this to plot a 2 x 2 grid with 4 confusion matrices for different models for example, or to customize the style and elements in the plot.

2.1.4 Evaluating Grid Search Results

A common practice in Machine Learning is to train several models with different hyperparameters and compare the performance across hyperparameter sets. scikit-learn provides a tool to do it: `sklearn.grid_search.GridSearchCV`, which trains the same model with different parameters. When doing grid search, it is tempting to just take the 'best model' and carry on, but analizing the results can give us some interesting information, so it's worth taking a look at the results.

sklearn-evaluation includes a plotting function to evaluate grid search results, this way we can see how the model performs when changing one (or two) hyperparameter(s) by keeping the rest constant.

First, let's load some data.

```
In [1]: data = datasets.make_classification(n_samples=200, n_features=10,
   ...:                                     n_informative=4, class_sep=0.5)
   ...:

In [2]: X = data[0]

In [3]: y = data[1]
```

Now, we need to define which hyperparameter sets we wan to include in the grid search, we do so by defining a dictionary with hyperparameter-values pairs and scikit-learn will automatically generate all possible combinations. For the dictionary below, we can generate 16 combinations (4*2*2).

```
In [4]: hyperparameters = {
   ...:     'n_estimators': [1, 10, 50, 100],
   ...:     'criterion': ['gini', 'entropy'],
   ...:     'max_features': ['sqrt', 'log2'],
   ...: }
   ...:
```

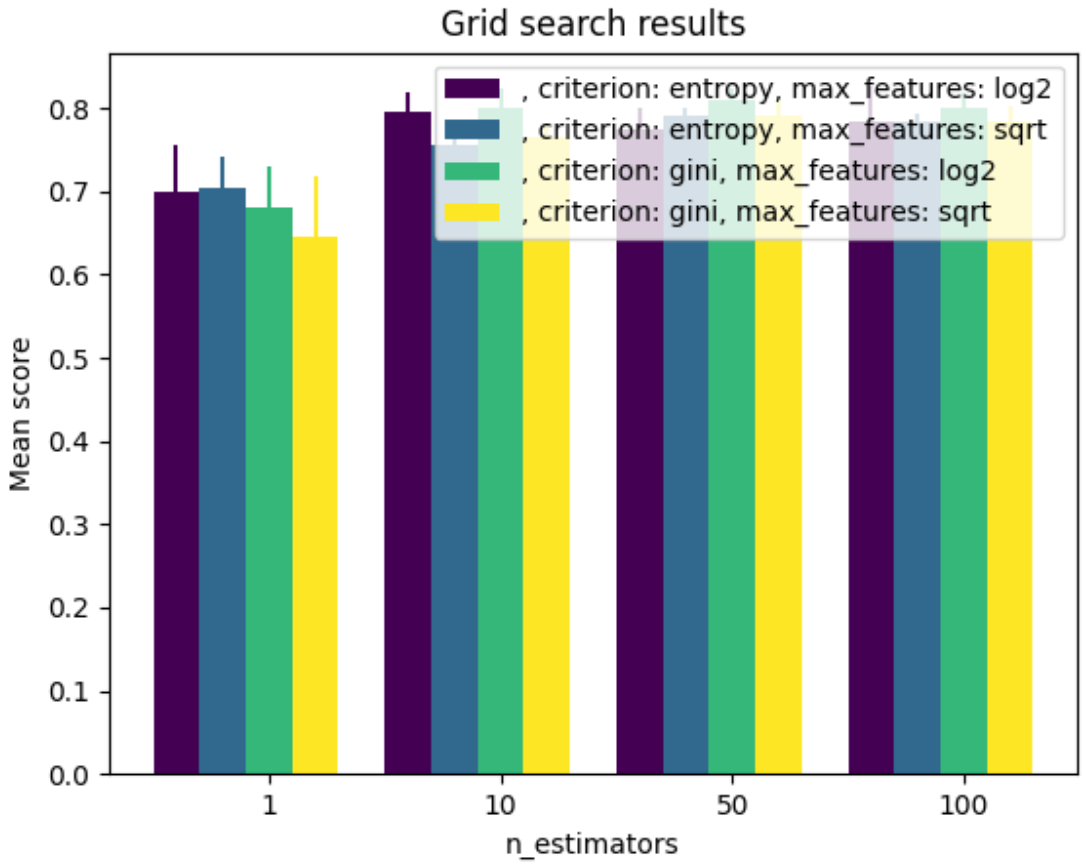
To perform a grid search we first need to select an estimator, in this case a Random Forest, then use the GridSearchCV class to pass the estimator, the hypeparameter dictionary and the number of folds for cross-validation.

After fitting the models (note that we call fit on the GridSearchCV instead of the estimator itself) we can get the results using the `sklearn.grid_search.GridSearchCV.cv_results_` attribute.

```
In [5]: est = RandomForestClassifier(n_estimators=5)

In [6]: clf = GridSearchCV(est, hyperparameters, cv=3)

In [7]: clf.fit(X, y)
Out[7]:
GridSearchCV(cv=3, estimator=RandomForestClassifier(n_estimators=5),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_features': ['sqrt', 'log2'],
                         'n_estimators': [1, 10, 50, 100]})

In [8]: grid_scores = clf.cv_results_
```

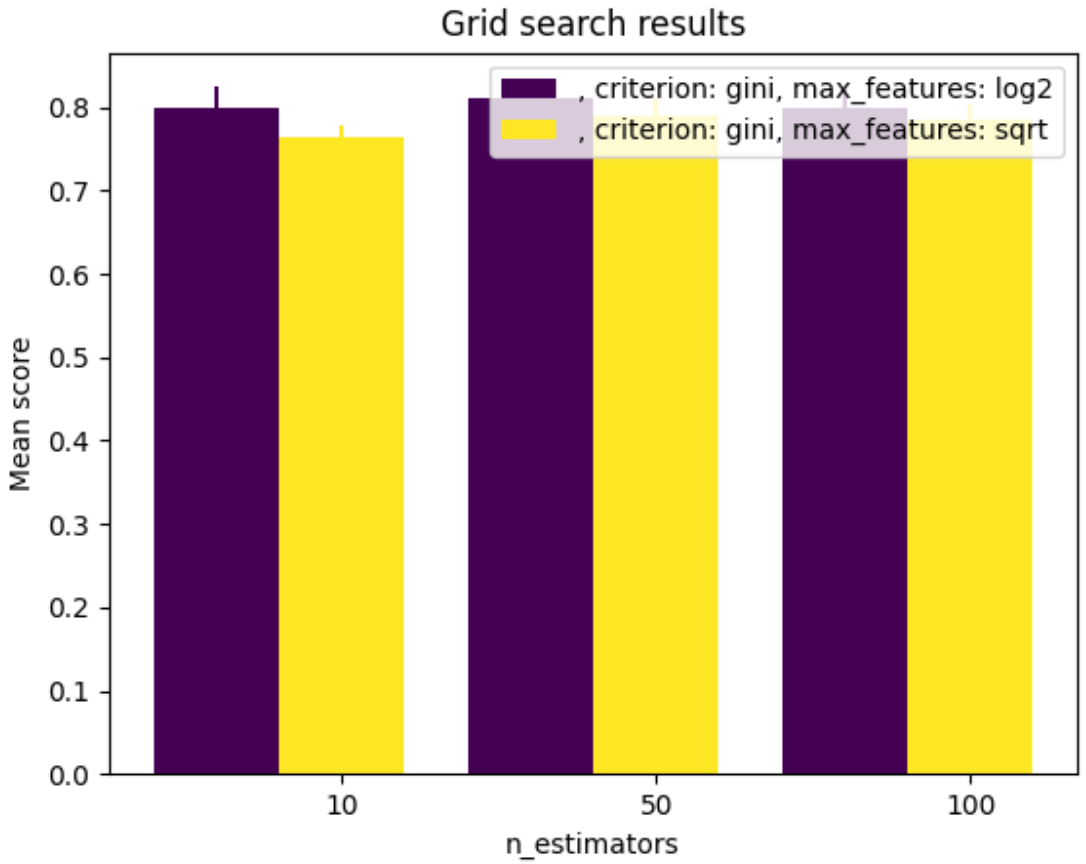
To generate the plot, we need to pass the grid_scores and the parameter(s) to change, let's see how the number of trees in the Random Forest affects the performance of the model.

```
In [9]: plot.grid_search(clf.cv_results_, change='n_estimators', kind='bar')
Out[9]: <AxesSubplot:title={'center':'Grid search results'}, xlabel='n_estimators',␣
↪ylabel='Mean score'>
```



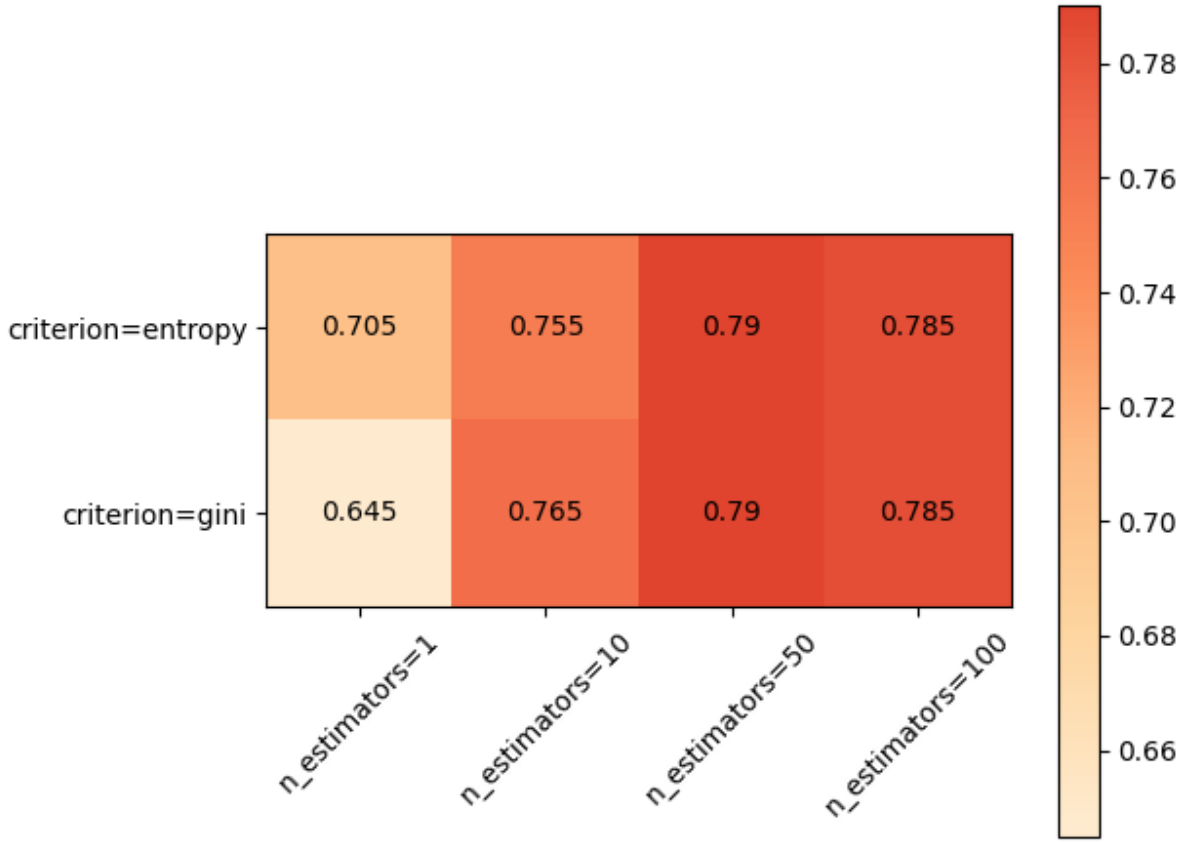
We can also subset the grid scores to plot by using the subset parameter (note that the hyperparameter in change can also appear in subset).

```
In [10]: plot.grid_search(clf.cv_results_, change='n_estimators',
   ....:                  subset={'n_estimators': [10, 50, 100],
   ....:                          'criterion': 'gini'},
   ....:                  kind='bar')
   ....:
Out[10]: <AxesSubplot:title={'center':'Grid search results'}, xlabel='n_estimators',␣
↪ylabel='Mean score'>
```



To evaluate the effect of two hyperparameters, we pass the two of them in change, note that for this to work we need to subset the grid scores to match only one group. In this case we'll plot n_estimators and criterion, so we need to subset max_features to one single value.

```
In [11]: plot.grid_search(clf.cv_results_, change=('n_estimators', 'criterion'),
   ....:                  subset={'max_features': 'sqrt'})
   ....:
Out[11]: <AxesSubplot:>
```



2.1.5 Advanced usage using matplotlib

As we mentioned in the previous section, using the functional interface provides great flexibility to evaluate your models, this sections includes some recipes for common tasks that involve the use of the matplotlib API.

Changing plot style

sklearn-evaluation uses whatever configuration matplotlib has, if you want to change the style of the plots easily you can use one of the many styles available:

```
In [1]: import matplotlib.style

In [2]: matplotlib.style.available
Out[2]:
['Solarize_Light2',
 '_classic_test_patch',
 '_mpl-gallery',
 '_mpl-gallery-nogrid',
 'bmh',
 'classic',
 'dark_background',
```

(continues on next page)

(continued from previous page)

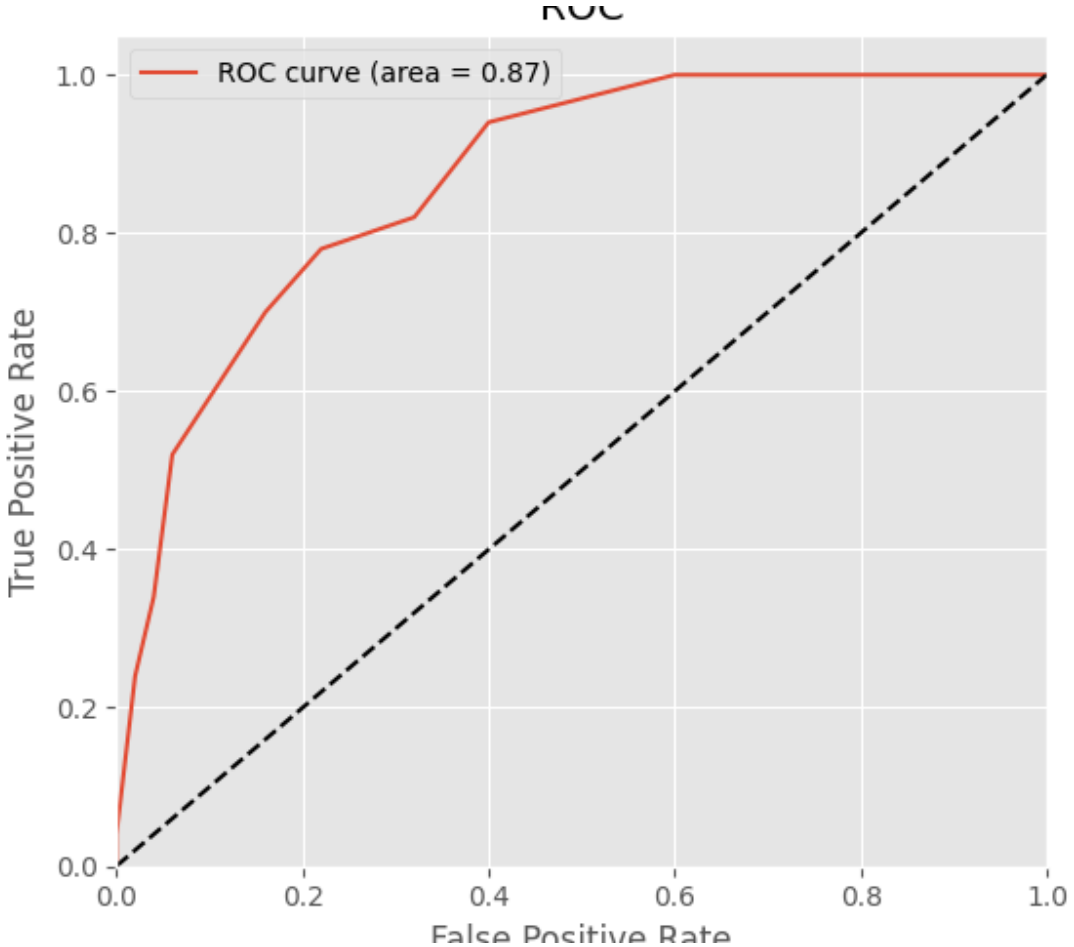
```
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark',
 'seaborn-dark-palette',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'tableau-colorblind10']
```

The change the style using

```
In [3]: matplotlib.style.use('ggplot')
```

Let's see how a ROC curve looks with the new style:

```
In [4]: plot.roc(y_true, y_score)
Out[4]: <AxesSubplot:title={'center':'ROC'}, xlabel='False Positive Rate', ylabel='True␣
↪Positive Rate'>
```



Saving plots

```
In [5]: ax = plot.roc(y_true, y_score)

In [6]: fig = ax.get_figure()

In [7]: fig.savefig('my-roc-curve.png')
```

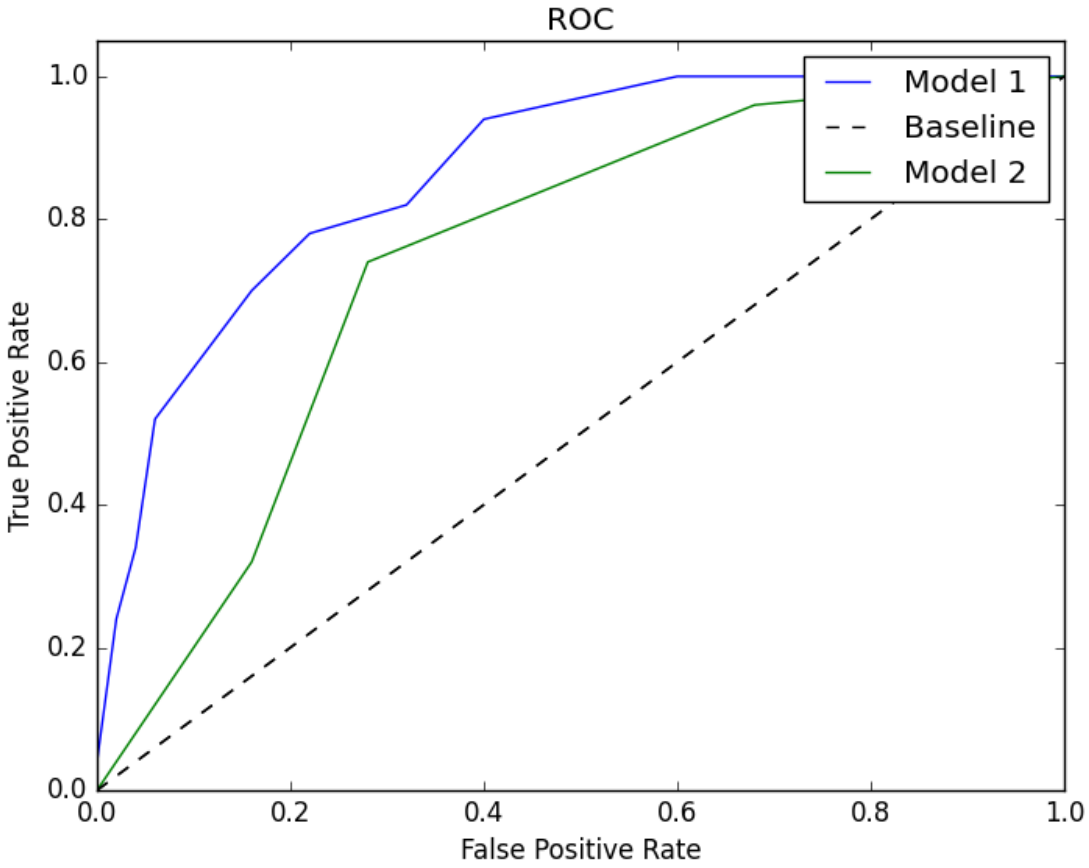
Comparing several models with one plot

```
In [8]: fig, ax = plt.subplots()

In [9]: plot.roc(y_true, y_score, ax=ax)
Out[9]: <AxesSubplot:title={'center':'ROC'}, xlabel='False Positive Rate', ylabel='True␣
↪Positive Rate'>

In [10]: plot.roc(y_true, y_score2, ax=ax)
Out[10]: <AxesSubplot:title={'center':'ROC'}, xlabel='False Positive Rate', ylabel='True␣
↪Positive Rate'>
```

(continues on next page)

(continued from previous page)

```
In [11]: ax.legend(['Model 1', 'Baseline', 'Model 2'])
Out[11]: <matplotlib.legend.Legend at 0x7f0ece4a00d0>

In [12]: fig
Out[12]: <Figure size 640x480 with 1 Axes>
```



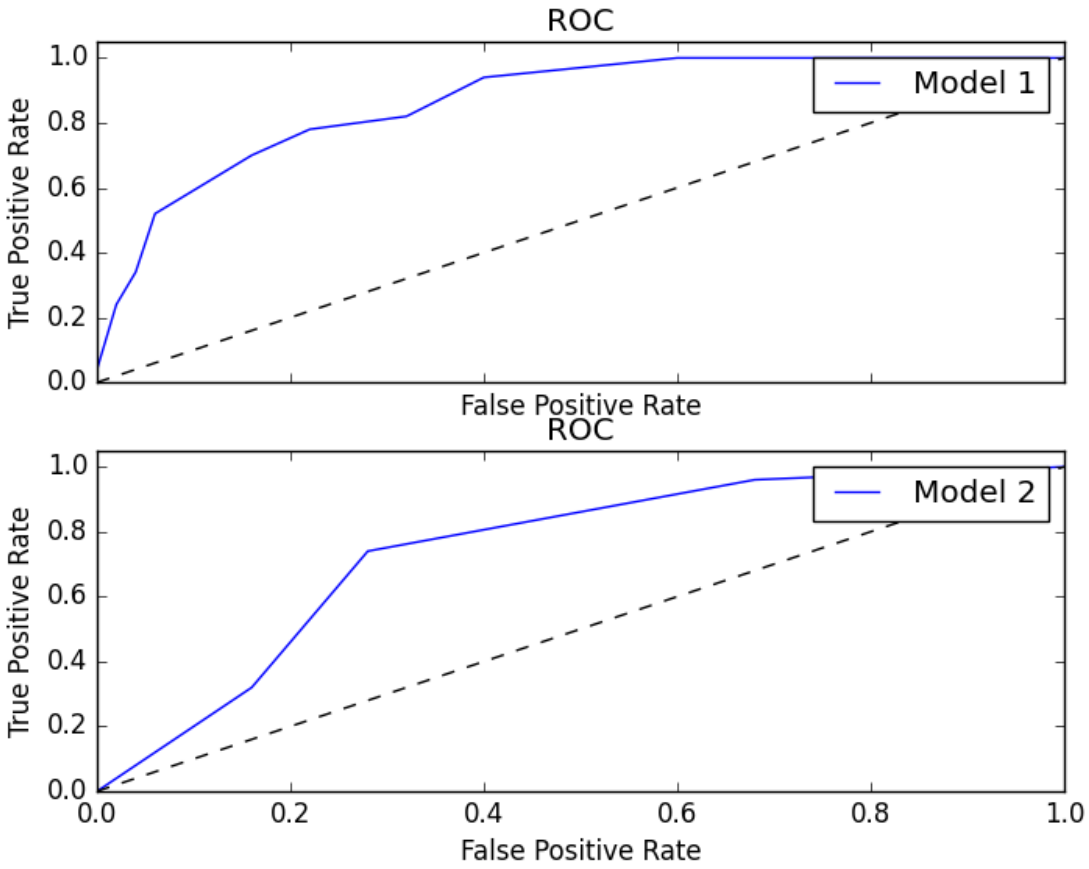
Grid plots

```
In [13]: fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

In [14]: plot.roc(y_true, y_score, ax=ax1)
Out[14]: <AxesSubplot:title={'center':'ROC'}, xlabel='False Positive Rate', ylabel='True␣
↪Positive Rate'>

In [15]: plot.roc(y_true, y_score2, ax=ax2)
Out[15]: <AxesSubplot:title={'center':'ROC'}, xlabel='False Positive Rate', ylabel='True␣
↪Positive Rate'>

In [16]: ax1.legend(['Model 1'])
Out[16]: <matplotlib.legend.Legend at 0x7f0ecc8147f0>
```

(continues on next page)

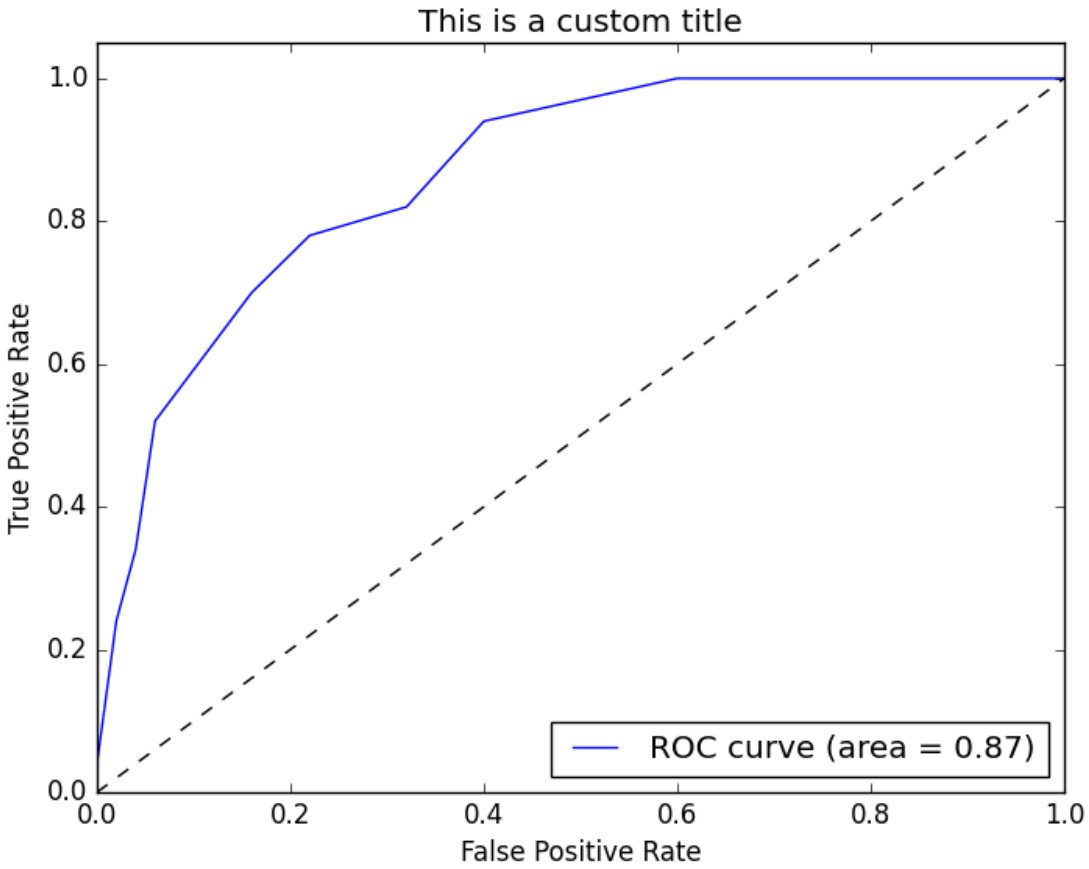
(continued from previous page)

```
In [17]: ax2.legend(['Model 2'])
Out[17]: <matplotlib.legend.Legend at 0x7f0ed8d05130>

In [18]: fig
Out[18]: <Figure size 640x480 with 2 Axes>
```



Customizing plots

```
In [19]: ax = plot.roc(y_true, y_score)

In [20]: ax.set_title('This is a custom title')
Out[20]: Text(0.5, 1.0, 'This is a custom title')

In [21]: ax
Out[21]: <AxesSubplot:title={'center':'This is a custom title'}, xlabel='False Positive␣
↪Rate', ylabel='True Positive Rate'>
```



2.1.6 Tracking Machine Learning experiments

`SQLiteTracker` provides a simple yet powerful way to track ML experiments using a SQLite database.

[1]:
```
from sklearn_evaluation import SQLiteTracker

from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

[4]:
```
iris = load_iris(as_frame=True)
X, y = iris['data'], iris['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_
↪state=42)

models = [RandomForestRegressor(), LinearRegression(), Lasso()]
```

[5]:
```
for m in models:
    model = type(m).__name__
    print(f'Fitting {model}')

    # .new() returns a uuid and creates an entry in the db
    uuid = tracker.new()
    m.fit(X_train, y_train)
    y_pred = m.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)

    # add data with the .update(uuid, {'param': 'value'}) method
    tracker.update(uuid, {'mse': mse, 'model': model, **m.get_params()})
```

```
Fitting RandomForestRegressor
Fitting LinearRegression
Fitting Lasso
```

Or use `.insert(uuid, params)` to supply your own ID:

[6]:
```
svr = SVR()
svr.fit(X_train, y_train)
y_pred = svr.predict(X_test)
mse = mean_squared_error(y_test, y_pred)

tracker.insert('my_uuid', {'mse': mse, 'model': type(svr).__name__, **svr.get_params()})
```

`tracker` shows last experiments by default:

[7]:
```
tracker
```

[7]:
```
SQLiteTracker
+----------------------------------+---------------------+------------------------------
↪------------------------------------------------------------------------------------
↪------------------------------------------------------------------------------------
↪------------------------------------------------------------------------------------
↪------------------------------------------------------------------------------------
↪-----------------------------------------------------------------------+-----------+
| uuid                             | created             | parameters                  ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                  | comment   |
+==================================+=====================+==========================================
| 560751f2aef34a87aee50cec0ca7ac1c | 2022-04-16 14:41:57 | {"mse": 0.042600341137617896,
↪"model": "LinearRegression", "copy_X": true, "fit_intercept": true, "n_jobs": null,
↪"normalize": "deprecated", "positive": false}                                       ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                    |           |
+----------------------------------+---------------------+------------------------------
↪------------------------------------------------------------------------------------
↪------------------------------------------------------------------------------------
↪------------------------------------------------------------------------------------
↪------------------------------------------------------------------------------------
↪-----------------------------------------------------------------------+-----------+
```

(continues on next page)

(continued from previous page)

```
| 2e5ecf364aa94c4da36acc8b12329431 | 2022-04-16 14:41:57 | {"mse": 0.4317655183287654,
↪"model": "Lasso", "alpha": 1.0, "copy_X": true, "fit_intercept": true, "max_iter":␣
↪1000, "normalize": "deprecated", "positive": false, "precompute": false, "random_state
↪": null, "selection": "cyclic", "tol": 0.0001, "warm_start": false}                  ␣
↪                                                                                     ␣
↪                                                                    |           |
+----------------------------------+---------------------+--------------------------------
↪-------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------
↪-----------------------------------------------------------------------+-----------+
| my_uuid                          | 2022-04-16 14:41:57 | {"mse": 0.03041912541362143,
↪"model": "SVR", "C": 1.0, "cache_size": 200, "coef0": 0.0, "degree": 3, "epsilon": 0.1,
↪ "gamma": "scale", "kernel": "rbf", "max_iter": -1, "shrinking": true, "tol": 0.001,
↪"verbose": false}                                                                    ␣
↪                                                                                     ␣
↪                                                                 |           |
+----------------------------------+---------------------+--------------------------------
↪-------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------
↪-----------------------------------------------------------------------+-----------+
| 1ad8f65486dd4ba3b899e00c97a4a72d | 2022-04-16 14:41:56 | {"mse": 0.009877999999999998,
↪"model": "RandomForestRegressor", "bootstrap": true, "ccp_alpha": 0.0, "criterion":
↪"squared_error", "max_depth": null, "max_features": "auto", "max_leaf_nodes": null,
↪"max_samples": null, "min_impurity_decrease": 0.0, "min_samples_leaf": 1, "min_samples_
↪split": 2, "min_weight_fraction_leaf": 0.0, "n_estimators": 100, "n_jobs": null, "oob_
↪score": false, "random_state": null, "verbose": 0, "warm_start": false} |           |
+----------------------------------+---------------------+--------------------------------
↪-------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------
↪-------------------------------------------------------------------------------------
↪-----------------------------------------------------------------------+-----------+
(Most recent experiments)
```

Querying experiments

[8]:
```
ordered = tracker.query("""
SELECT uuid,
       json_extract(parameters, '$.model') AS model,
       json_extract(parameters, '$.mse') AS mse
FROM experiments
ORDER BY json_extract(parameters, '$.mse') ASC
""")
ordered
```

[8]:

uuid	model	mse

(continues on next page)

(continued from previous page)

```
1ad8f65486dd4ba3b899e00c97a4a72d  RandomForestRegressor  0.009878
my_uuid                                             SVR  0.030419
560751f2aef34a87aee50cec0ca7ac1c       LinearRegression  0.042600
2e5ecf364aa94c4da36acc8b12329431                 Lasso  0.431766
```

The query method returns a data frame with "uuid" as the index:

[9]:
```
type(ordered)
```

[9]:
```
pandas.core.frame.DataFrame
```

Adding comments

[10]:
```
tracker.comment(ordered.index[0], 'Best performing experiment')
```

User `tracker[uuid]` to get a single experiment:

[11]:
```
tracker[ordered.index[0]]
```

[11]:
```
                                          created  \
uuid
1ad8f65486dd4ba3b899e00c97a4a72d  2022-04-16 14:41:56

                                                                                parameters  \
uuid
1ad8f65486dd4ba3b899e00c97a4a72d  {"mse": 0.009877999999999998, "model": "Random...

                                                    comment
uuid
1ad8f65486dd4ba3b899e00c97a4a72d  Best performing experiment
```

Getting recent experiments

The recent method also returns a data frame:

[12]:
```
df = tracker.recent()
df
```

[12]:
```
                                          created  \
uuid
560751f2aef34a87aee50cec0ca7ac1c  2022-04-16 14:41:57
2e5ecf364aa94c4da36acc8b12329431  2022-04-16 14:41:57
my_uuid                           2022-04-16 14:41:57
1ad8f65486dd4ba3b899e00c97a4a72d  2022-04-16 14:41:56

                                                                                parameters  \
uuid
560751f2aef34a87aee50cec0ca7ac1c  {"mse": 0.042600341137617896, "model": "Linear...
2e5ecf364aa94c4da36acc8b12329431  {"mse": 0.4317655183287654, "model": "Lasso", ...
my_uuid                           {"mse": 0.03041912541362143, "model": "SVR", "...
1ad8f65486dd4ba3b899e00c97a4a72d  {"mse": 0.009877999999999998, "model": "Random...
```

(continues on next page)

(continued from previous page)

```
                                                comment
uuid
560751f2aef34a87aee50cec0ca7ac1c                    None
2e5ecf364aa94c4da36acc8b12329431                    None
my_uuid                                             None
1ad8f65486dd4ba3b899e00c97a4a72d  Best performing experiment
```

Pass `normalize=True` to convert the nested JSON dictionary into columns:

[13]:
```
df = tracker.recent(normalize=True)
df
```

[13]:
```
                                        created       mse  \
uuid
560751f2aef34a87aee50cec0ca7ac1c  2022-04-16 14:41:57  0.042600
2e5ecf364aa94c4da36acc8b12329431  2022-04-16 14:41:57  0.431766
my_uuid                           2022-04-16 14:41:57  0.030419
1ad8f65486dd4ba3b899e00c97a4a72d  2022-04-16 14:41:56  0.009878

                                             model copy_X fit_intercept  \
uuid
560751f2aef34a87aee50cec0ca7ac1c       LinearRegression   True          True
2e5ecf364aa94c4da36acc8b12329431                  Lasso   True          True
my_uuid                                             SVR    NaN           NaN
1ad8f65486dd4ba3b899e00c97a4a72d  RandomForestRegressor    NaN           NaN

                                  n_jobs   normalize positive  alpha  \
uuid
560751f2aef34a87aee50cec0ca7ac1c     NaN  deprecated    False    NaN
2e5ecf364aa94c4da36acc8b12329431     NaN  deprecated    False    1.0
my_uuid                              NaN         NaN      NaN    NaN
1ad8f65486dd4ba3b899e00c97a4a72d     NaN         NaN      NaN    NaN

                                  max_iter  ... max_features  max_leaf_nodes  \
uuid                                        ...
560751f2aef34a87aee50cec0ca7ac1c       NaN  ...          NaN             NaN
2e5ecf364aa94c4da36acc8b12329431    1000.0  ...          NaN             NaN
my_uuid                               -1.0  ...          NaN             NaN
1ad8f65486dd4ba3b899e00c97a4a72d       NaN  ...         auto             NaN

                                 max_samples  min_impurity_decrease  \
uuid
560751f2aef34a87aee50cec0ca7ac1c         NaN                    NaN
2e5ecf364aa94c4da36acc8b12329431         NaN                    NaN
my_uuid                                  NaN                    NaN
1ad8f65486dd4ba3b899e00c97a4a72d         NaN                    0.0

                                 min_samples_leaf  min_samples_split  \
uuid
560751f2aef34a87aee50cec0ca7ac1c              NaN                NaN
2e5ecf364aa94c4da36acc8b12329431              NaN                NaN
my_uuid                                       NaN                NaN
```

(continues on next page)

(continued from previous page)

```
1ad8f65486dd4ba3b899e00c97a4a72d              1.0               2.0

                                 min_weight_fraction_leaf  n_estimators  \
uuid
560751f2aef34a87aee50cec0ca7ac1c                      NaN           NaN
2e5ecf364aa94c4da36acc8b12329431                      NaN           NaN
my_uuid                                               NaN           NaN
1ad8f65486dd4ba3b899e00c97a4a72d                      0.0         100.0

                                 oob_score                     comment
uuid
560751f2aef34a87aee50cec0ca7ac1c       NaN                        None
2e5ecf364aa94c4da36acc8b12329431       NaN                        None
my_uuid                                NaN                        None
1ad8f65486dd4ba3b899e00c97a4a72d     False  Best performing experiment

[4 rows x 38 columns]
```

[14]:
```
# delete our example database
from pathlib import Path
Path('my_experiments.db').unlink()
```

[ ]:
```

```

2.1.7 Analyzing results from notebooks

The `.ipynb` format is capable of storing tables and charts in a standalone file. This makes it a great choice for model evaluation reports. `NotebookCollection` allows you to retrieve results from previously executed notebooks to compare them.

[1]:
```
import papermill as pm
import jupytext

from sklearn_evaluation import NotebookCollection
```

Let's first generate a few notebooks, we have a `train.py` script that trains a single model, let's convert it to a jupyter notebook:

[2]:
```
nb = jupytext.read('train.py')
jupytext.write(nb, 'train.ipynb')
```

We use papermill to execute the notebook with different parameters, we'll train 4 models: 2 random forest, a linear regression and a support vector regression:

[3]:
```
# models with their corresponding parameters
params = [{
    'model': 'sklearn.ensemble.RandomForestRegressor',
    'params': {
        'n_estimators': 50
```

(continues on next page)

(continued from previous page)

```
    }
}, {
    'model': 'sklearn.ensemble.RandomForestRegressor',
    'params': {
        'n_estimators': 100
    }
}, {
    'model': 'sklearn.linear_model.LinearRegression',
    'params': {
        'normalize': True
    }
}, {
    'model': 'sklearn.svm.LinearSVR',
    'params': {}
}]

# ids to identify each experiment
ids = [
    'random_forest_1', 'random_forest_2', 'linear_regression',
    'support_vector_regression'
]

# output files
files = [f'{i}.ipynb' for i in ids]

# execute notebooks using papermill
for f, p in zip(files, params):
    pm.execute_notebook('train.ipynb', output_path=f, parameters=p)
```

```
Executing:   0%|          | 0/17 [00:00<?, ?cell/s]
Executing:   0%|          | 0/17 [00:00<?, ?cell/s]
Executing:   0%|          | 0/17 [00:00<?, ?cell/s]
Executing:   0%|          | 0/17 [00:00<?, ?cell/s]
```

To use `NotebookCollection`, we pass a a list of paths, and optionally, ids for each notebook (uses paths by default).

The only requirement is that cells whose output we want to extract must have tags, each tag then becomes a key in the notebook collection. For instructions on adding tags, see this.

Extracted tables add colors to certain cells to identify the best and worst metrics. By default, it assumes that metrics are errors (smaller is better). If you are using scores (larger is better), pass `scores=True`, if you have both, pass a list of scores:

[4]:
```
nbs = NotebookCollection(paths=files, ids=ids, scores=['r2'])
```

To get a list of tags available:

[5]:
```
list(nbs)
```

[5]:
```
['model_name', 'feature_names', 'model_params', 'plot', 'metrics', 'river']
```

`model_params` contains a dictionary with model parameters, let's get them (click on the tabs to switch):

[6]:
```
# pro-tip: then typing the tag, press the "Tab" key for autocompletion!
nbs['model_params']
```

[6]:
```
<sklearn_evaluation.nb.NotebookCollection.HTMLMapping at 0x7f1983d240a0>
```

`plot` has a `y_true` vs `y_pred` chart:

[7]:
```
nbs['plot']
```

[7]:
```
<sklearn_evaluation.nb.NotebookCollection.HTMLMapping at 0x7f198427b850>
```

On each notebook, `metrics` outputs a data frame with a single row with mean absolute error (mae) and mean squared error (mse) as columns.

For single-row tables, a "Compare" tab shows all results at once:

[8]:
```
nbs['metrics']
```

[8]:
```
<sklearn_evaluation.nb.NotebookCollection.HTMLMapping at 0x7f1984299dc0>
```

We can see that the second random forest is performing the best in both metrics.

`river` contains a multi-row table where with error metrics broken down by the `CHAS` indicator feature. Multi-row tables *do not* display the "Compare" tab:

[9]:
```
nbs['river']
```

[9]:
```
<sklearn_evaluation.nb.NotebookCollection.HTMLMapping at 0x7f1983d24160>
```

If we only compare two notebooks, the output is a bit different:

[10]:
```
# only compare two notebooks
nbs_two = NotebookCollection(paths=files[:2], ids=ids[:2], scores=['r2'])
```

Comparing single-row tables includes a diff column with the error difference between experiments. Error reductions are showed in green, increments in red:

[11]:
```
nbs_two['metrics']
```

[11]:
```
<sklearn_evaluation.nb.NotebookCollection.HTMLMapping at 0x7f198afb8a00>
```

When comparing multi-row tables, the "Compare" tab appears, showing the difference between the tables:

[12]:
```
nbs_two['river']
```

[12]:
```
<sklearn_evaluation.nb.NotebookCollection.HTMLMapping at 0x7f1984295ac0>
```

When displaying dictionaries, a "Compare" tab shows with a diff view:

[13]:
```
nbs_two['model_params']
```

[13]:
```
<sklearn_evaluation.nb.NotebookCollection.HTMLMapping at 0x7f19a1f6b310>
```

Lists (and sets) are compared based on elements existence:

[14]:
```
nbs_two['feature_names']
```

[14]:
```
<sklearn_evaluation.nb.NotebookCollection.HTMLMapping at 0x7f198423c730>
```

Using the mapping interface

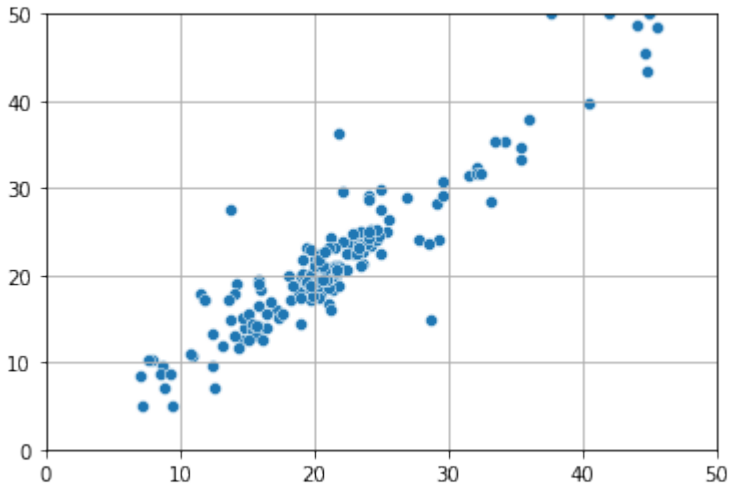
`NotebookCollection` has a dict-like interface, you can retrieve data from individual notebooks:

[15]:
```
nbs['model_params']['random_forest_1']
```

[15]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 50,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

[16]:
```
nbs['plot']['random_forest_2']
```

[16]:



2.2 API Reference

2.2.1 Plotting

Plotting functions

`sklearn_evaluation.plot.`**`confusion_matrix`**(*y_true*, *y_pred*, *target_names=None*, *normalize=False*, *cmap=None*, *ax=None*)

Plot confusion matrix.

Parameters

- **y_true** (*array-like, shape = [n_samples]*) – Correct target values (ground truth).
- **y_pred** (*array-like, shape = [n_samples]*) – Target predicted classes (estimator predictions).
- **target_names** (*list*) – List containing the names of the target classes. List must be in order e.g. `['Label for class 0', 'Label for class 1']`. If `None`, generic labels will be generated e.g. `['Class 0', 'Class 1']`
- **ax** (*matplotlib Axes*) – Axes object to draw the plot onto, otherwise uses current Axes
- **normalize** (*bool*) – Normalize the confusion matrix
- **cmap** (*matplotlib Colormap*) – If `None` uses a modified version of matplotlib's OrRd colormap.

Notes

http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

Returns **ax** – Axes containing the plot

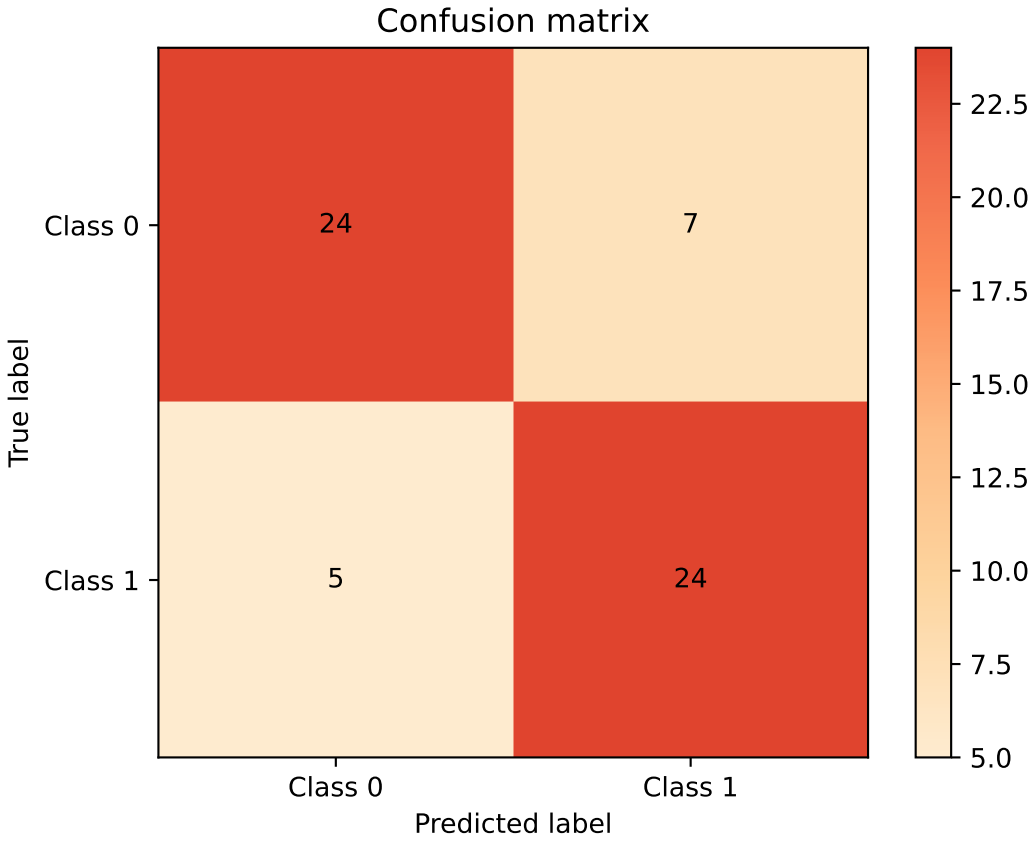
Return type matplotlib Axes

Examples

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

from sklearn_evaluation import plot

data = datasets.make_classification(200, 10, n_informative=5, class_sep=0.65)
X = data[0]
y = data[1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

est = RandomForestClassifier()
est.fit(X_train, y_train)

y_pred = est.predict(X_test)
y_score = est.predict_proba(X_test)
y_true = y_test

plot.confusion_matrix(y_true, y_pred)
plt.show()
```

sklearn_evaluation.plot.**feature_importances**(*data*, *top_n=None*, *feature_names=None*, *orientation='horizontal'*, *ax=None*)

Get and order feature importances from a scikit-learn model or from an array-like structure. If data is a scikit-learn model with sub-estimators (e.g. RandomForest, AdaBoost) the function will compute the standard deviation of each feature.


Confusion matrix
Class 0
Class 1
True label
24
7
5
24
Class 0
Class 1
Predicted label
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0

Parameters

- **data** (*sklearn model or array-like structure*) – Object to get the data from.
- **top_n** (*int*) – Only get results for the top_n features.
- **feature_names** (*array-like*) – Feature names
- **orientation** (*('horizontal', 'vertical')*) – Bar plot orientation
- **ax** (*matplotlib Axes*) – Axes object to draw the plot onto, otherwise uses current Axes

Returns **ax** – Axes containing the plot

Return type matplotlib Axes

Examples

```
"""
Feature importances plot
"""
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

from sklearn_evaluation import plot

X, y = datasets.make_classification(200, 20, n_informative=5, class_sep=0.65)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

model = RandomForestClassifier(n_estimators=1)
model.fit(X_train, y_train)

# plot all features
ax = plot.feature_importances(model)
plt.show()
```

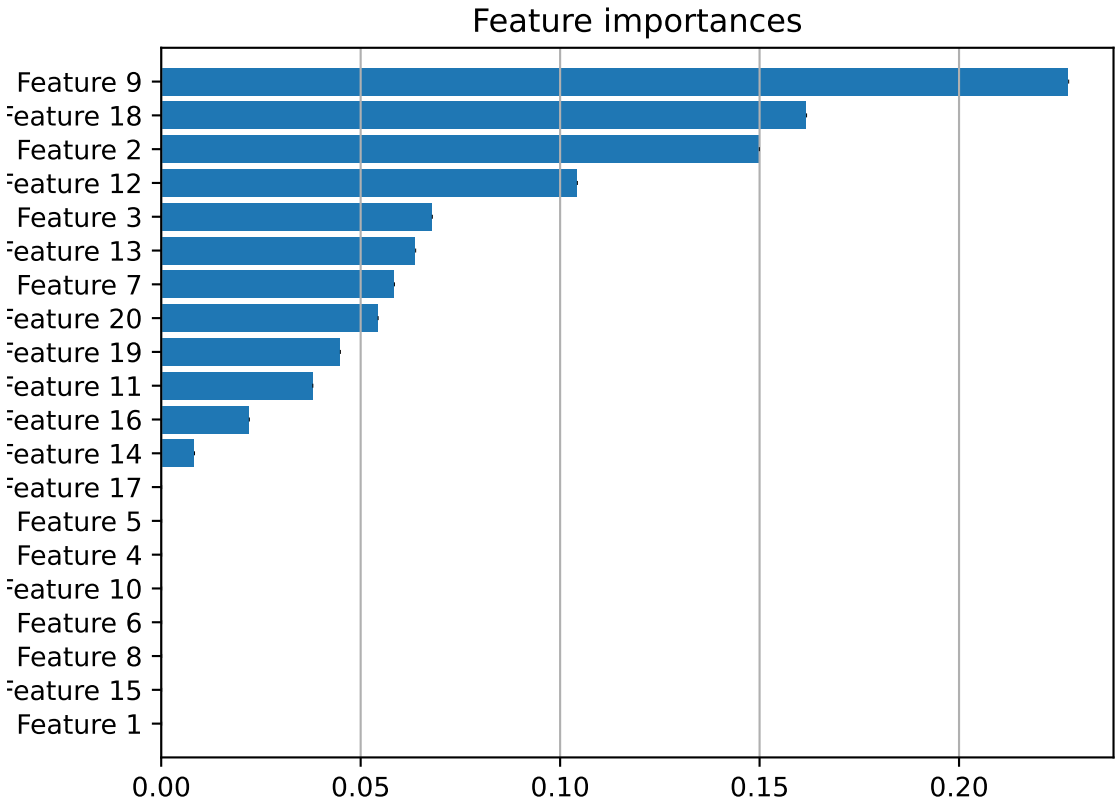
```
# only top 5
plot.feature_importances(model, top_n=5)
plt.show()
```

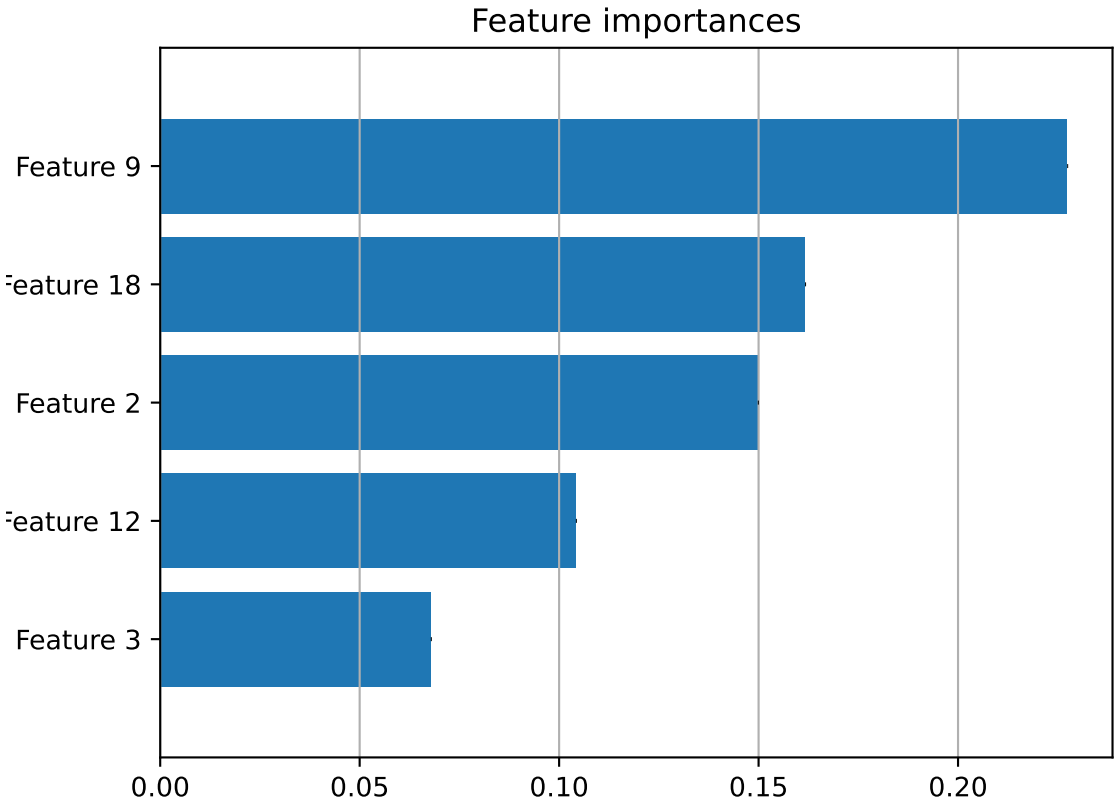
sklearn_evaluation.plot.**grid_search**(*cv_results_*, *change*, *subset=None*, *kind='line'*, *cmap=None*, *ax=None*)

Plot results from a sklearn grid search by changing two parameters at most.

Parameters

- **cv_results** (*list of named tuples*) – Results from a sklearn grid search (get them using the *cv_results_* parameter)
- **change** (*str or iterable with len<=2*) – Parameter to change
- **subset** (*dictionary-like*) – parameter-value(s) pairs to subset from grid_scores. (e.g. `{'n_estimators': [1, 10]}`), if None all combinations will be used.
- **kind** (*['line', 'bar']*) – This only applies whe change is a single parameter. Changes the type of plot


Feature importances
Feature 9
Feature 18
Feature 2
Feature 12
Feature 3
Feature 13
Feature 7
Feature 20
Feature 19
Feature 11
Feature 16
Feature 14
Feature 17
Feature 5
Feature 4
Feature 10
Feature 6
Feature 8
Feature 15
Feature 1
0.00
0.05
0.10
0.15
0.20


Feature importances
Feature 9
eature 18
Feature 2
eature 12
Feature 3
0.00
0.05
0.10
0.15
0.20

- **cmap** (*matplotlib Colormap*) – This only applies when change are two parameters. Colormap used for the matrix. If None uses a modified version of matplotlib's OrRd colormap.
- **ax** (*matplotlib Axes*) – Axes object to draw the plot onto, otherwise uses current Axes

Returns **ax** – Axes containing the plot

Return type matplotlib Axes

Examples

```
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import datasets

from sklearn_evaluation.plot import grid_search


iris = datasets.load_iris()


parameters = {
    'n_estimators': [1, 10, 50, 100],
    'criterion': ['gini', 'entropy'],
    'max_features': ['sqrt', 'log2'],
}

est = RandomForestClassifier()
clf = GridSearchCV(est, parameters, cv=5)

X, y = datasets.make_classification(1000, 10, n_informative=5, class_sep=0.7)
clf.fit(X, y)

# changing numeric parameter without any restrictions
# in the rest of the parameter set
grid_search(clf.cv_results_, change='n_estimators')
plt.show()
```

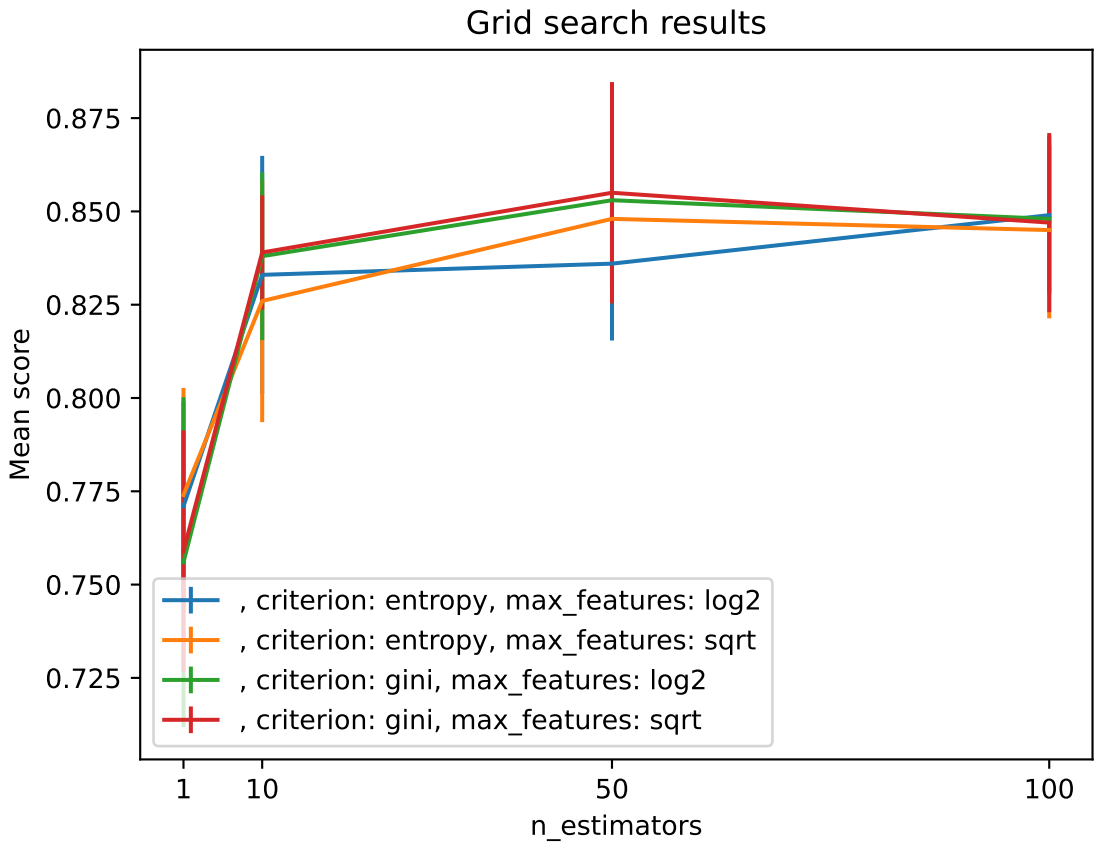
```
# you can also use bars
grid_search(clf.cv_results_, change='n_estimators', kind='bar')
plt.show()
```

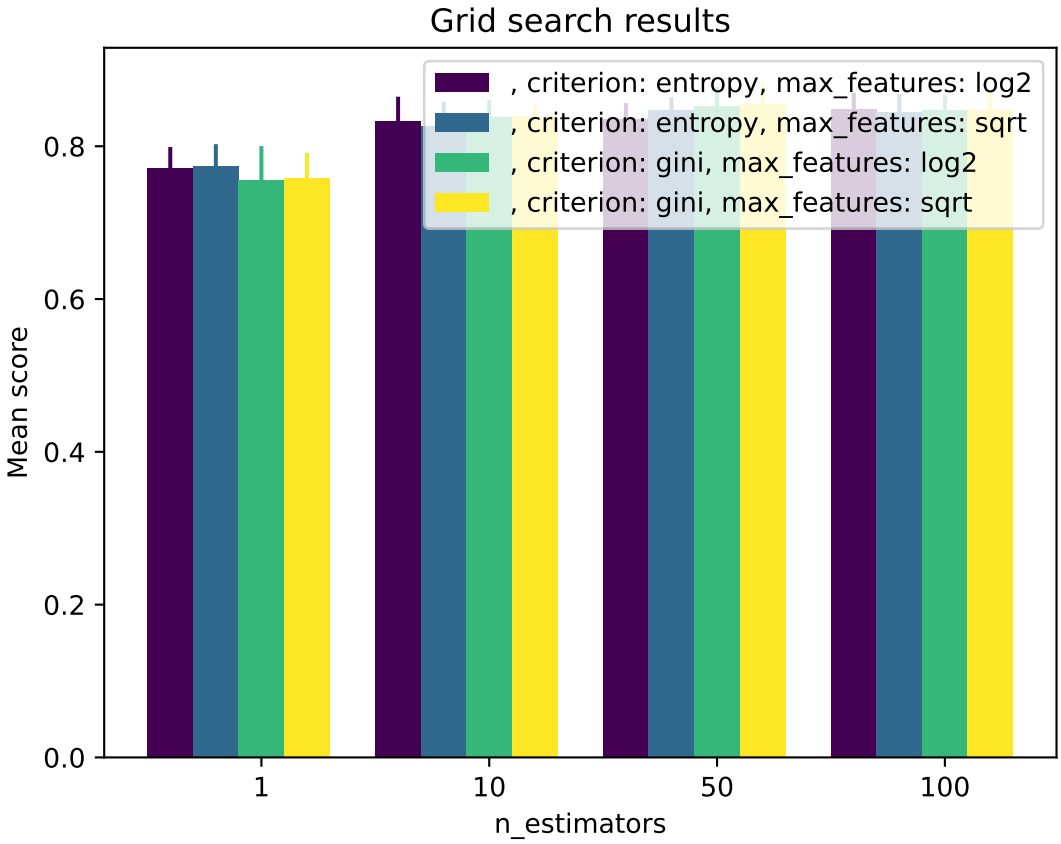
```
# changing a categorical variable without any constraints
grid_search(clf.cv_results_, change='criterion')
plt.show()
```

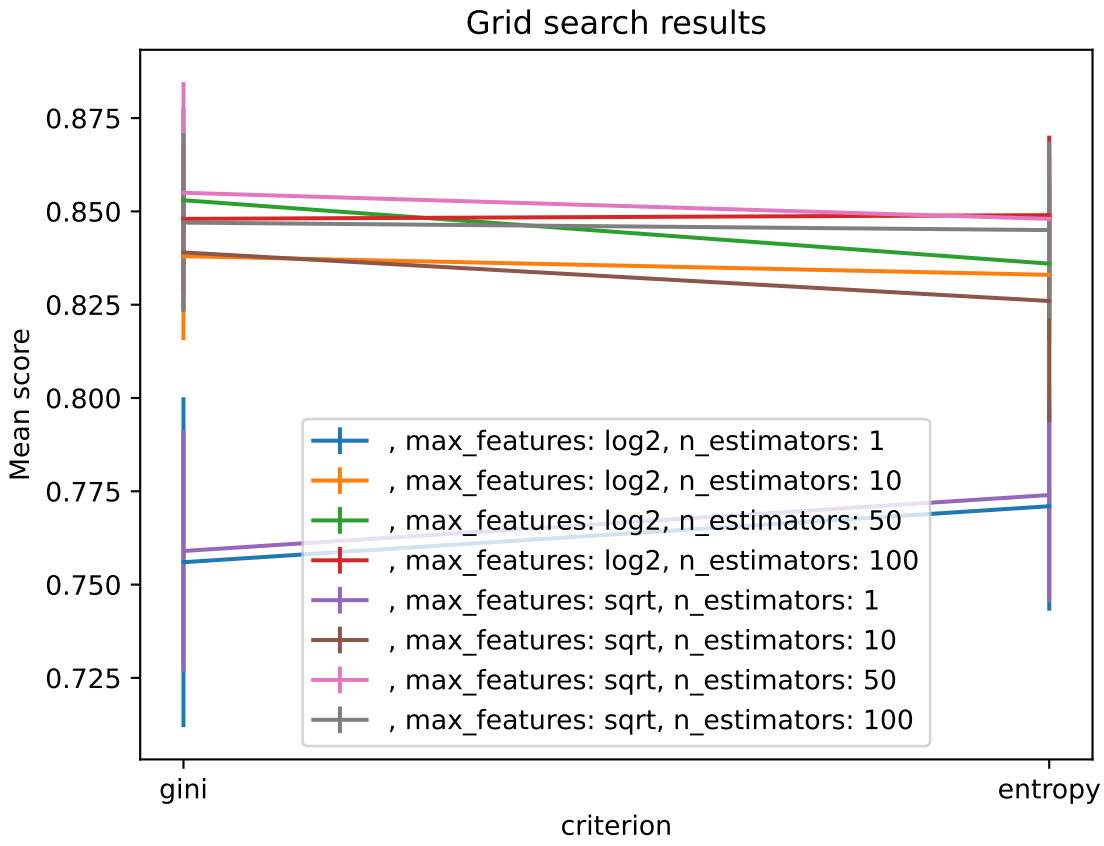
```
# bar
grid_search(clf.cv_results_, change='criterion', kind='bar')
plt.show()
```

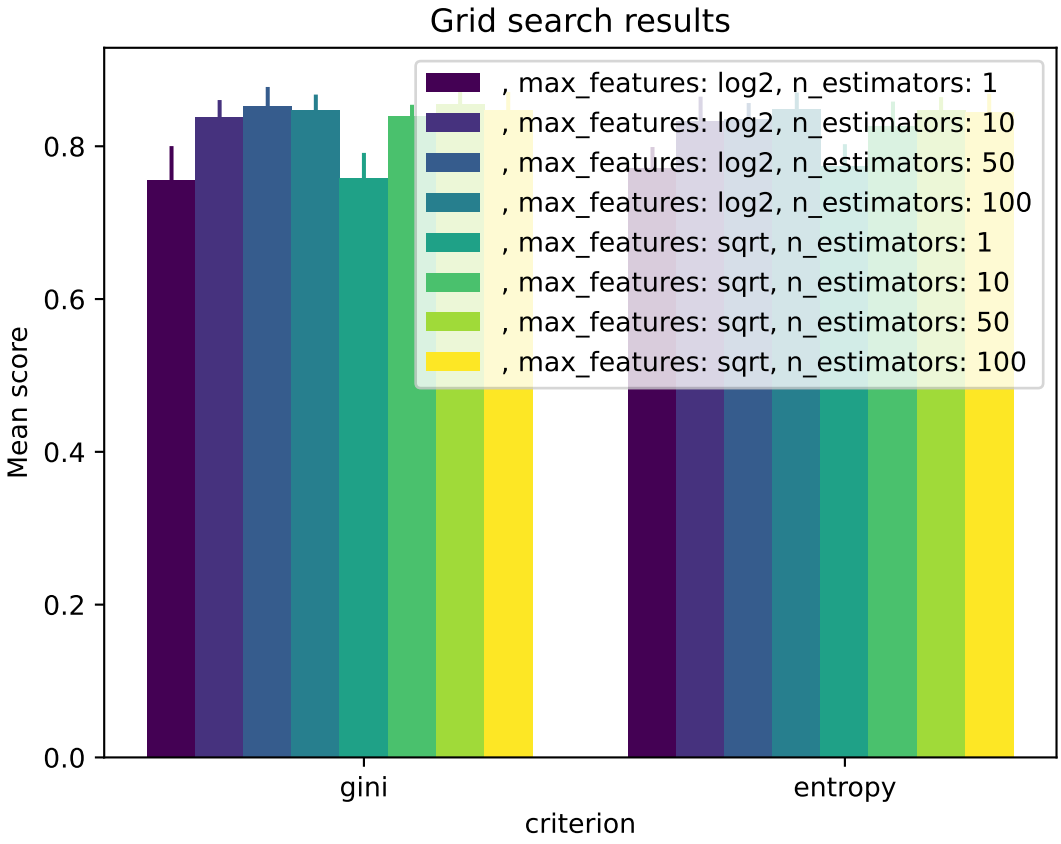
```
# varying a numerical parameter but constraining
# the rest of the parameter set
```

(continues on next page)


Grid search results
0.875
0.850
0.825
0.800
0.775
0.750
0.725
Mean score
1
10
50
100
n_estimators
, criterion: entropy, max_features: log2
, criterion: entropy, max_features: sqrt
, criterion: gini, max_features: log2
, criterion: gini, max_features: sqrt

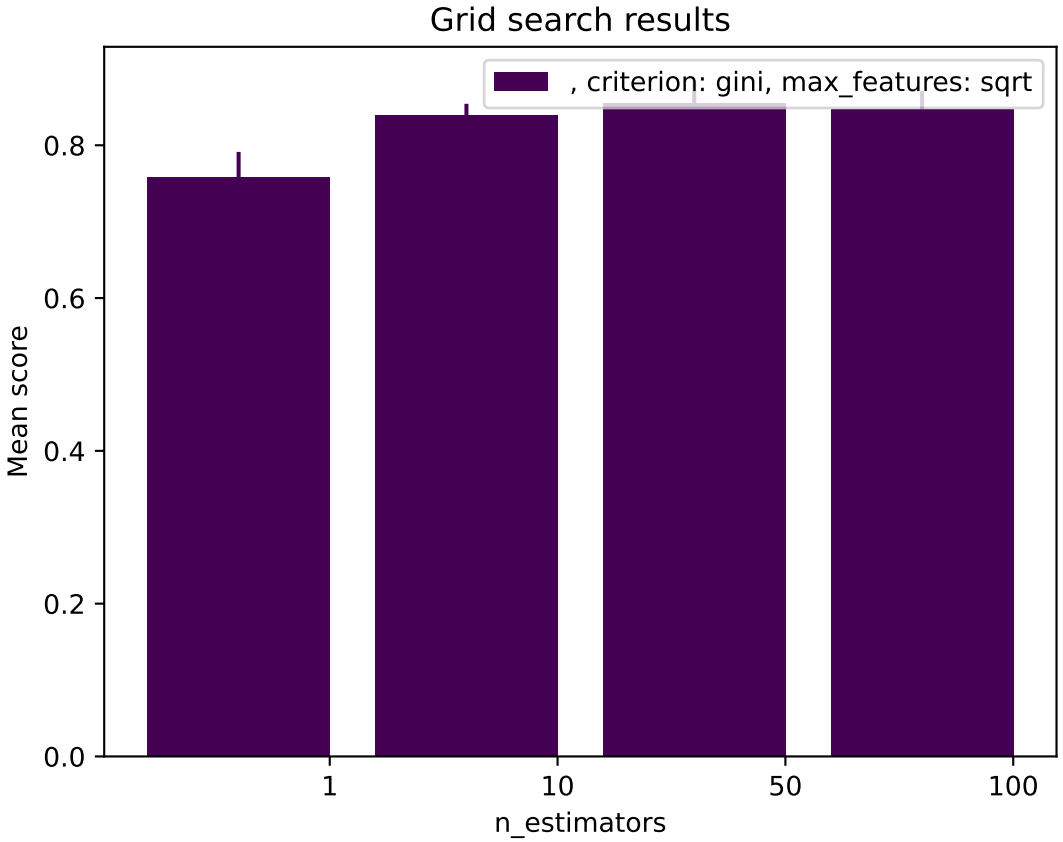

Grid search results
, criterion: entropy, max_features: log2
, criterion: entropy, max_features: sqrt
, criterion: gini, max_features: log2
, criterion: gini, max_features: sqrt
Mean score
0.0
0.2
0.4
0.6
0.8
1
10
50
100
n_estimators


Grid search results
Mean score
0.875
0.850
0.825
0.800
0.775
0.750
0.725
gini
entropy
criterion
, max_features: log2, n_estimators: 1
, max_features: log2, n_estimators: 10
, max_features: log2, n_estimators: 50
, max_features: log2, n_estimators: 100
, max_features: sqrt, n_estimators: 1
, max_features: sqrt, n_estimators: 10
, max_features: sqrt, n_estimators: 50
, max_features: sqrt, n_estimators: 100


Grid search results
, max_features: log2, n_estimators: 1
, max_features: log2, n_estimators: 10
, max_features: log2, n_estimators: 50
, max_features: log2, n_estimators: 100
, max_features: sqrt, n_estimators: 1
, max_features: sqrt, n_estimators: 10
, max_features: sqrt, n_estimators: 50
, max_features: sqrt, n_estimators: 100
Mean score
0.0
0.2
0.4
0.6
0.8
gini
entropy
criterion

(continued from previous page)

```
grid_search(clf.cv_results_, change='n_estimators',
            subset={'max_features': 'sqrt', 'criterion': 'gini'},
            kind='bar')
plt.show()
```




```
# same as above but letting max_features to have two values
grid_search(clf.cv_results_, change='n_estimators',
            subset={'max_features': ['sqrt', 'log2'], 'criterion': 'gini'},
            kind='bar')
plt.show()
```

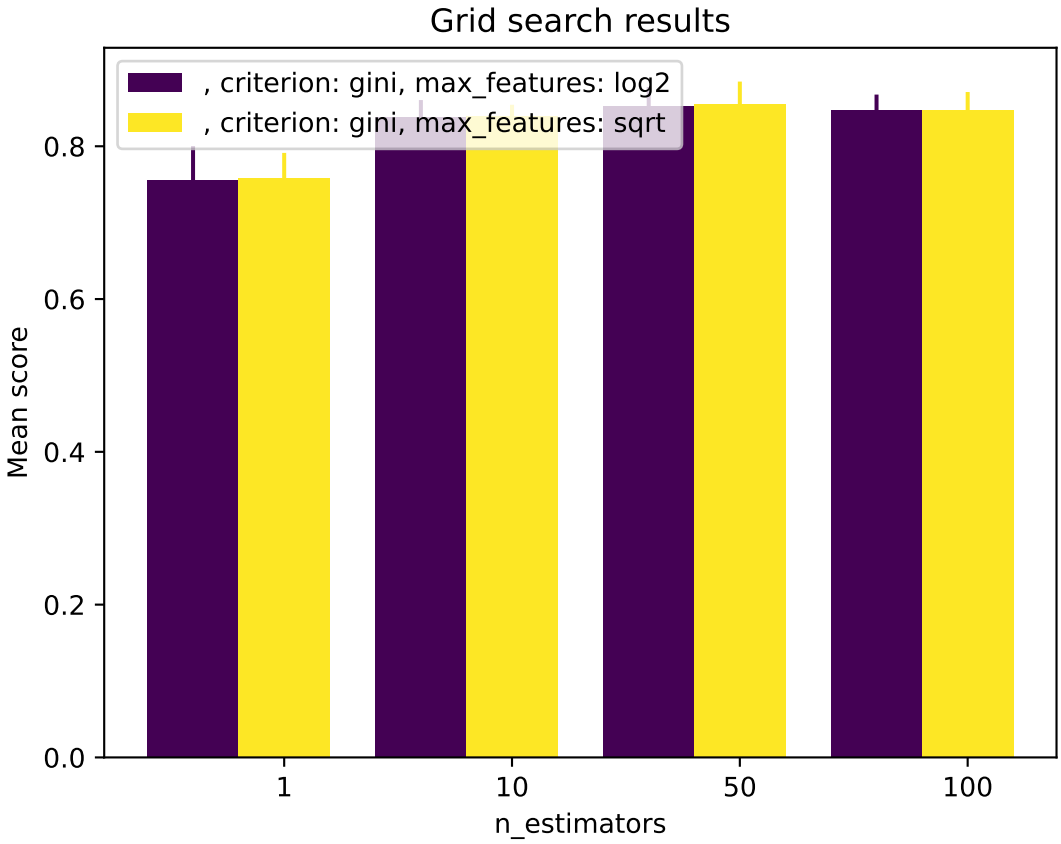
```
# varying two parameters - you can only show this as a
# matrix so the kind parameter will be ignored
grid_search(clf.cv_results_, change=('n_estimators', 'criterion'),
            subset={'max_features': 'sqrt'})
plt.show()
```

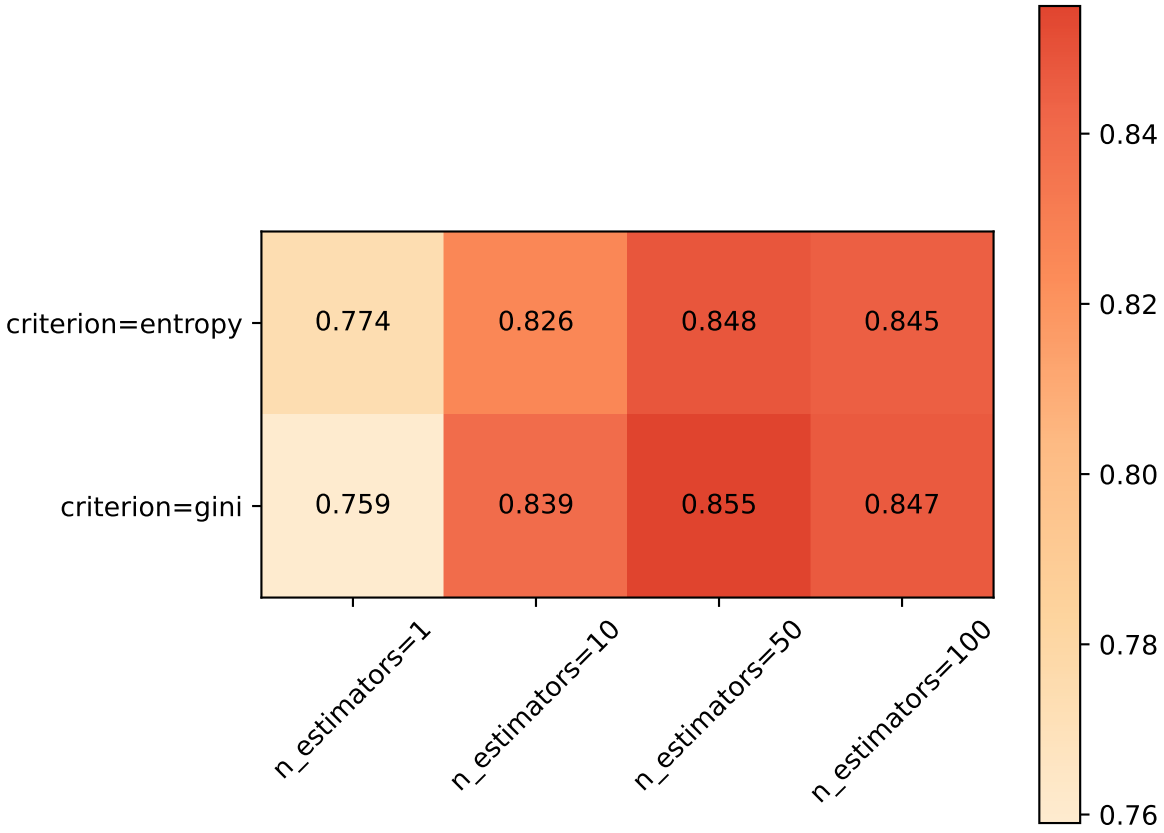
sklearn_evaluation.plot.**learning_curve**(*train_scores*, *test_scores*, *train_sizes*, *ax=None*)

Plot a learning curve

Plot a metric vs number of examples for the training and test set

Parameters


Grid search results
, criterion: gini, max_features: log2
, criterion: gini, max_features: sqrt
Mean score
0.0
0.2
0.4
0.6
0.8
1
10
50
100
n_estimators


criterion=entropy
criterion=gini
0.774
0.826
0.848
0.845
0.759
0.839
0.855
0.847
n_estimators=1
n_estimators=10
n_estimators=50
n_estimators=100
0.84
0.82
0.80
0.78
0.76

- **train_scores** (*array-like*) – Scores for the training set
- **test_scores** (*array-like*) – Scores for the test set
- **train_sizes** (*array-like*) – Relative or absolute numbers of training examples used to generate the learning curve
- **ax** (*matplotlib Axes*) – Axes object to draw the plot onto, otherwise uses current Axes

Returns **ax** – Axes containing the plot

Return type matplotlib Axes

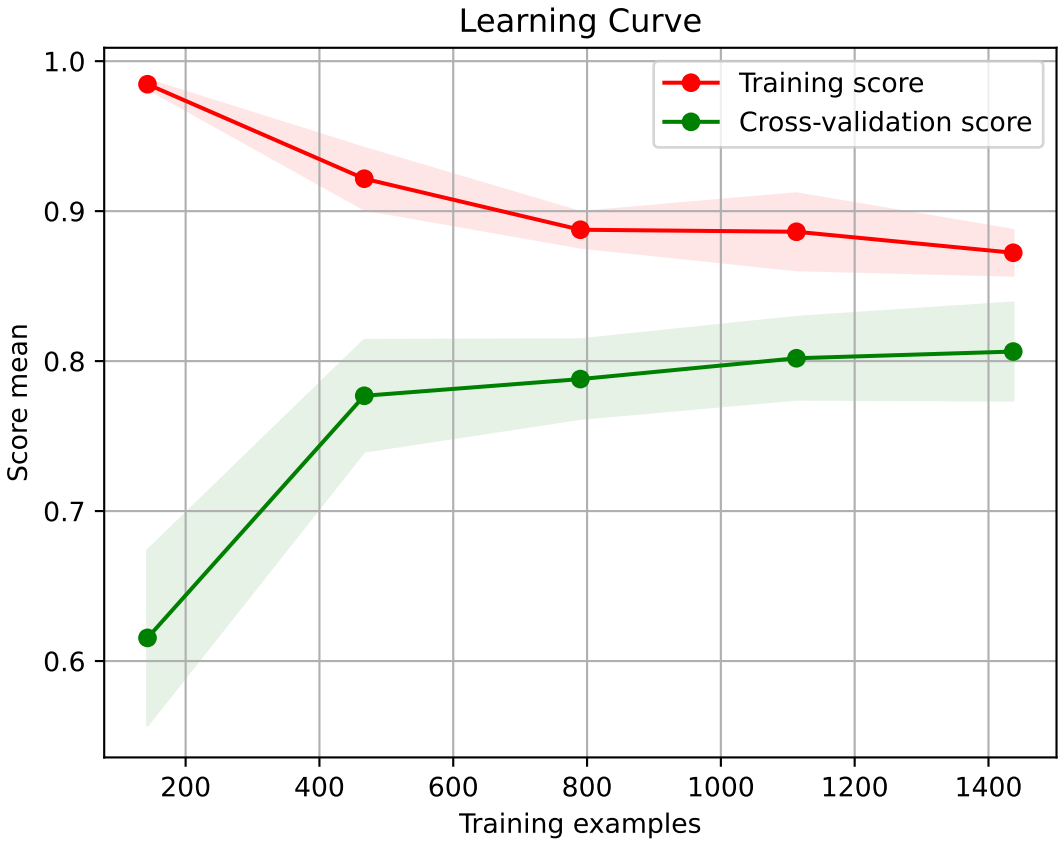
Examples

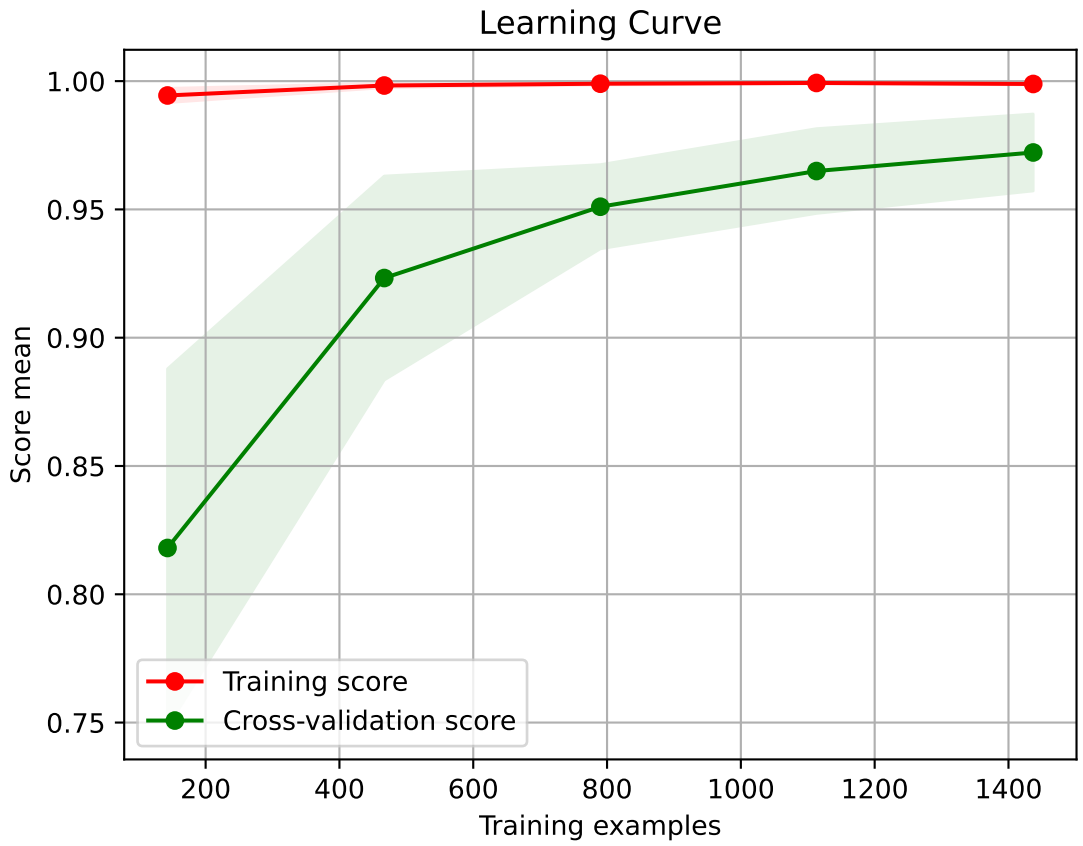
```
from sklearn.model_selection import learning_curve
from sklearn import model_selection
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
import numpy as np

from sklearn_evaluation import plot

digits = load_digits()
X, y = digits.data, digits.target

# Cross validation with 100 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = model_selection.ShuffleSplit(digits.data.shape[0],
                                  test_size=0.2, random_state=0)
cv = 5
estimator = GaussianNB()
train_sizes = np.linspace(.1, 1.0, 5)
train_sizes, train_scores, test_scores = learning_curve(
    estimator, X, y, cv=cv, n_jobs=1, train_sizes=train_sizes)
plot.learning_curve(train_scores, test_scores, train_sizes)
plt.show()
```

```
# SVC is more expensive so we do a lower number of CV iterations:
cv = model_selection.ShuffleSplit(digits.data.shape[0],
                                  test_size=0.2, random_state=0)
cv = 5
estimator = SVC(gamma=0.001)
train_sizes = np.linspace(.1, 1.0, 5)
train_sizes, train_scores, test_scores = learning_curve(
    estimator, X, y, cv=cv, n_jobs=1, train_sizes=train_sizes)

plot.learning_curve(train_scores, test_scores, train_sizes)
plt.show()
```

sklearn_evaluation.plot.**metrics_at_thresholds**(*fn*, *y_true*, *y_score*, *n_thresholds=10*, *start=0.0*, *ax=None*)

Plot metrics at increasing thresholds


Learning Curve
Training score
Cross-validation score
Score mean
1.0
0.9
0.8
0.7
0.6
200
400
600
800
1000
1200
1400
Training examples


Learning Curve
Score mean
1.00
0.95
0.90
0.85
0.80
0.75
200
400
600
800
1000
1200
1400
Training examples
Training score
Cross-validation score

sklearn_evaluation.plot.**precision_at_proportions**(*y_true*, *y_score*, *ax=None*)

Plot precision values at different proportions.

Parameters

- **y_true** (*array-like*) – Correct target values (ground truth).
- **y_score** (*array-like*) – Target scores (estimator predictions).
- **ax** (*matplotlib Axes*) – Axes object to draw the plot onto, otherwise uses current Axes

Returns **ax** – Axes containing the plot

Return type matplotlib Axes

sklearn_evaluation.plot.**precision_recall**(*y_true*, *y_score*, *ax=None*)

Plot precision-recall curve.

Parameters

- **y_true** (*array-like, shape = [n_samples]*) – Correct target values (ground truth).
- **y_score** (*array-like, shape = [n_samples] or [n_samples, 2] for binary*) – classification or [n_samples, n_classes] for multiclass

  Target scores (estimator predictions).
- **ax** (*matplotlib Axes*) – Axes object to draw the plot onto, otherwise uses current Axes

Notes

It is assumed that the y_score parameter columns are in order. For example, if `y_true = [2, 2, 1, 0, 0, 1, 2]`, then the first column in y_score must contain the scores for class 0, second column for class 1 and so on.

Returns **ax** – Axes containing the plot

Return type matplotlib Axes

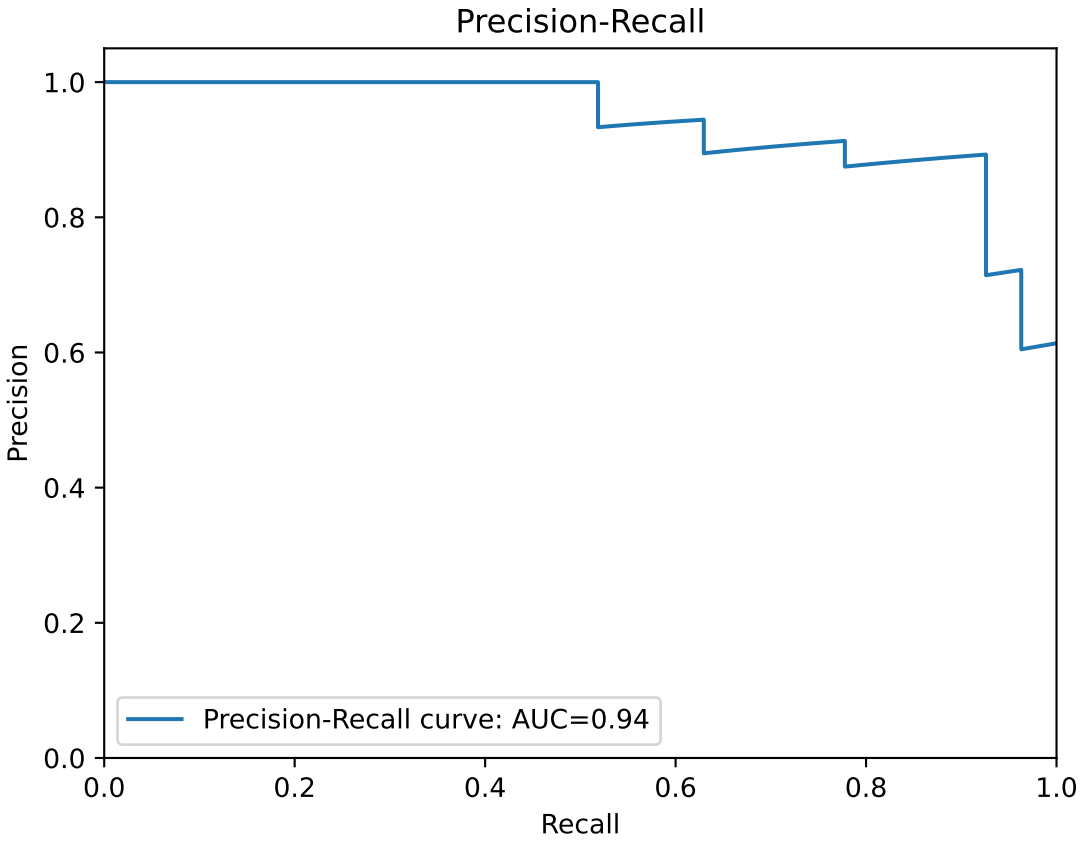
Examples

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

from sklearn_evaluation import plot

data = datasets.make_classification(200, 10, n_informative=5, class_sep=0.65)
X = data[0]
y = data[1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

est = RandomForestClassifier()
est.fit(X_train, y_train)

y_pred = est.predict(X_test)
y_score = est.predict_proba(X_test)
```

(continues on next page)

(continued from previous page)

```
y_true = y_test

plot.precision_recall(y_true, y_score)
plt.show()
```



sklearn_evaluation.plot.**roc**(*y_true*, *y_score*, *ax=None*)

Plot ROC curve.

Parameters

- **y_true** (*array-like, shape = [n_samples]*) – Correct target values (ground truth).
- **y_score** (*array-like, shape = [n_samples] or [n_samples, 2] for binary*) – classification or [n_samples, n_classes] for multiclass

  Target scores (estimator predictions).
- **ax** (*matplotlib Axes*) – Axes object to draw the plot onto, otherwise uses current Axes

Notes

It is assumed that the y_score parameter columns are in order. For example, if `y_true = [2, 2, 1, 0, 0, 1, 2]`, then the first column in y_score must contain the scores for class 0, second column for class 1 and so on.

Returns **ax** – Axes containing the plot

Return type matplotlib Axes

Examples

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

from sklearn_evaluation import plot

data = datasets.make_classification(200, 10, n_informative=5, class_sep=0.65)
X = data[0]
y = data[1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

est = RandomForestClassifier()
est.fit(X_train, y_train)

y_pred = est.predict(X_test)
y_score = est.predict_proba(X_test)
y_true = y_test

plot.roc(y_true, y_score)
plt.show()
```

sklearn_evaluation.plot.**validation_curve**(*train_scores*, *test_scores*, *param_range*, *param_name=None*, *semilogx=False*, *ax=None*)

Plot a validation curve

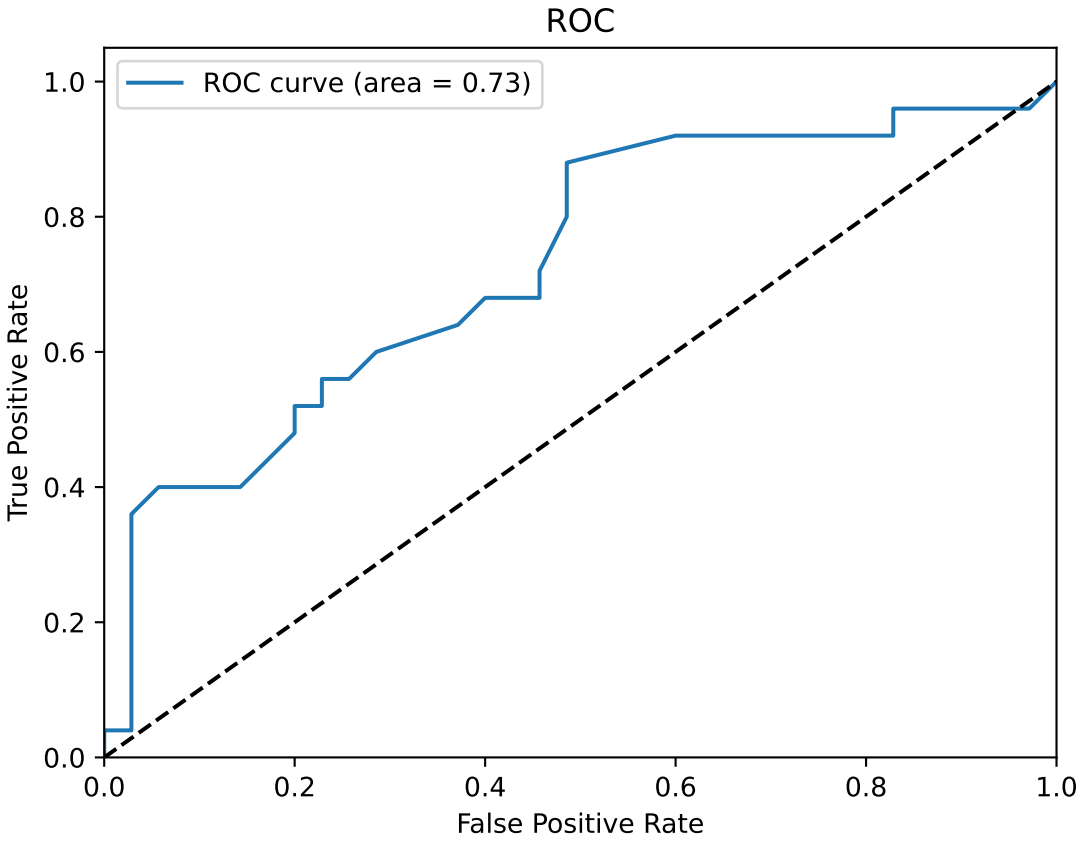
Plot a metric vs hyperparameter values for the training and test set

Parameters

- **train_scores** (*array-like*) – Scores for the training set
- **test_scores** (*array-like*) – Scores for the test set
- **param_range** (*str*) – Hyperparameter values used to generate the curve
- **param_range** – Hyperparameter name
- **semilgo** (*bool*) – Sets a log scale on the x axis
- **ax** (*matplotlib Axes*) – Axes object to draw the plot onto, otherwise uses current Axes

Returns **ax** – Axes containing the plot

Return type matplotlib Axes


ROC
ROC curve (area = 0.73)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

Examples

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_digits
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import validation_curve

from sklearn_evaluation import plot

digits = load_digits()
X, y = digits.data, digits.target

param_range = np.logspace(-6, -1, 5)
param_name = "gamma"
train_scores, test_scores = validation_curve(
    SVC(), X, y, param_name=param_name,
    param_range=param_range,
    cv=5,
    scoring="accuracy",
    n_jobs=1)

plot.validation_curve(train_scores, test_scores, param_range, param_name,
                      semilogx=True)
plt.show()
```

```
param_range = np.array([1, 10, 100])
param_name = "n_estimators"
train_scores, test_scores = validation_curve(
    RandomForestClassifier(), X, y,
    param_name=param_name,
    param_range=param_range,
    cv=10, scoring="accuracy", n_jobs=1)

plot.validation_curve(train_scores, test_scores, param_range, param_name,
                      semilogx=False)
plt.show()
```

2.2.2 Tables

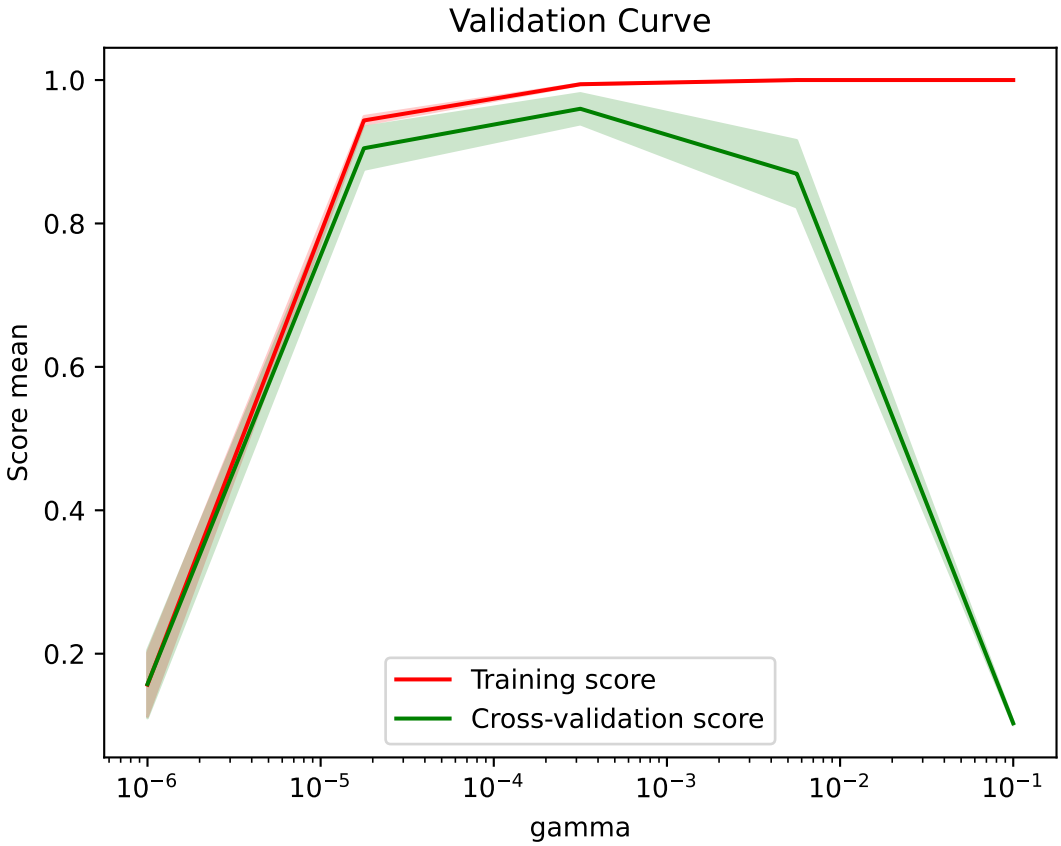
sklearn_evaluation.table.**feature_importances**(*data*, *top_n=None*, *feature_names=None*)

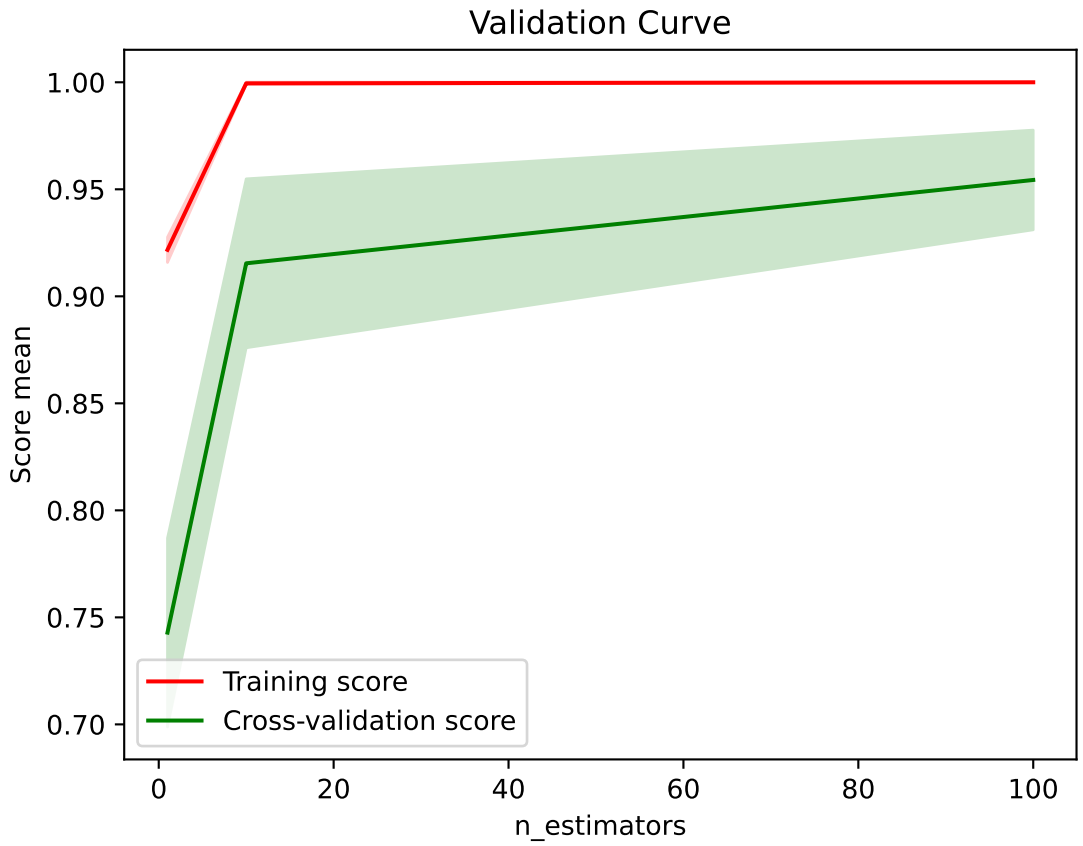
Get and order feature importances from a scikit-learn model or from an array-like structure.

If data is a scikit-learn model with sub-estimators (e.g. RandomForest, AdaBoost) the function will compute the standard deviation of each feature.

Parameters

- **data** (*sklearn model or array-like structure*) – Object to get the data from.
- **top_n** (*int*) – Only get results for the top_n features.
- **feature_names** (*array-like*) – Feature_names


Validation Curve
Score mean
1.0
0.8
0.6
0.4
0.2
Training score
Cross-validation score
10^{-6}
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
gamma


Validation Curve
Score mean
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
20
40
60
80
100
n_estimators
Training score
Cross-validation score

Returns Table object with the data. Columns are feature_name, importance (*std_* only included for models with sub-estimators)

Return type table

2.2.3 ClassifierEvaluator

class sklearn_evaluation.**ClassifierEvaluator**(*estimator=None*, *y_true=None*, *y_pred=None*, *y_score=None*, *feature_names=None*, *target_names=None*, *estimator_name=None*, *X=None*)

Encapsulates results from an estimator on a testing set to provide a simplified API from other modules. All parameters are optional, just fill the ones you need for your analysis.

Parameters

- **estimator** (*sklearn estimator*) – Must have a `feature_importances_` attribute.
- **y_true** (*array-like*) – Target predicted classes (estimator predictions).
- **y_pred** (*array-like*) – Correct target values (ground truth).
- **y_score** (*array-like*) – Target scores (estimator predictions).
- **feature_names** (*array-like*) – Feature names.
- **target_names** (*list*) – List containing the names of the target classes
- **estimator_name** (*str*) – Identifier for the model. This can be later used to identify the estimator when generating reports.

confusion_matrix()

Confusion matrix plot

property estimator_class

Estimator class (e.g. sklearn.ensemble.RandomForestClassifier)

property estimator_type

Estimator name (e.g. RandomForestClassifier)

feature_importances()

Feature importances plot

feature_importances_table()

Feature importances table

html_serializable()

Returns a EvaluatorHTMLSerializer instance, which is an object with the same methods and properties than a ClassifierEvaluator, but it returns HTML serialized versions of each (i.e. evaluator.feature_importances_table() returns a string with the table in HTML format, evaluator.confusion_matrix() returns a HTML image element with the image content encoded in base64), useful for generating reports using some template system

make_report(*template=None*)

Make HTML report

Parameters

- **template** (*str, or pathlib.Path, optional*) – HTML or Markdown template with jinja2 format. If a pathlib.Path object is passed, the content of the file is read. Within the template, the evaluator is passed as "e", so you can use things like

{{e.confusion_matrix()}} or any other attribute/method. If None, a default template is used

- **style** (*str*) – Path to a css file to apply style to the report. If None, no style will be applied

Returns Returns the contents of the report if path is None.

Return type Report

precision_at_proportions()

Precision at proportions plot

precision_recall()

Precision-recall plot

roc()

ROC plot

2.2.4 Training

Data selector

When training models, it is common to try out different subsets of features or subpopulations. `DataSelector` allows you to define a series of transformations on your data so you can succinctly define a subsetting pipeline as a series of dictionaries.

class sklearn_evaluation.training.**DataSelector**(**steps*)

Subset a pandas.DataFrame by passing a series of steps

Parameters ***steps** – Steps to apply to the data sequentially (order matters). Each step must be a dictionary with a key "kind" whose value must be one of "column_drop", "row_drop" or "column_keep". The rest of the key-value pairs must match the signature for the corresponding Step objects

transform(*df*, *return_summary: bool = False*)

Apply steps

Parameters

- **df** – Data frame to transform
- **return_summary** – If False, the function only returns the output data frame, if True, it also returns a summary table

class sklearn_evaluation.training.selector.**ColumnDrop**(*names: list = None*, *prefix: str = None*, *suffix: str = None*, *contains: str = None*, *max_na_prop: float = None*)

Drop columns

Parameters

- **names** – List of columns to drop
- **prefix** – Drop columns with this prefix (or list of)
- **suffix** – Drop columns with this suffix (or list of)
- **contains** – Drop columns if they contains this substring
- **max_na_prop** – Drop columns whose proportion of NAs [0, 1] is larger than this

class sklearn_evaluation.training.selector.**RowDrop**(*if_nas: bool = False*, *query: str = None*)

Drop rows

Parameters

- **if_nas** – If True, deletes all rows where there is at leat one NA
- **query** – Drops all rows matching the query (passed via pandas.query)

class sklearn_evaluation.training.selector.**ColumnKeep**(*names: Optional[list] = None*, *dotted_path: Optional[str] = None*)

Subset columns

Parameters **names** – List of columns to keep

2.2.5 SQLiteTracker

class sklearn_evaluation.**SQLiteTracker**(*path: str*)

A simple experiment tracker using SQLite

Click here to see the user guide.

Parameters **path** – Database location

comment(*uuid*, *comment*)

Add a comment to an experiment given its uuid

insert(*uuid*, *parameters*)

Insert a new experiment

new()

Create a new experiment, returns a uuid

query(*code*)

Query the database, returns a pandas.DataFrame

Examples

```
>>> from sklearn_evaluation import SQLiteTracker
>>> tracker = SQLiteTracker(':memory:') # example in-memory db
>>> tracker.insert('my_uuid', {'a': 1})
>>> df = tracker.query(
... "SELECT uuid, json_extract(parameters, '$.a') FROM experiments")
```

recent(*n=5*, *normalize=False*)

Get most recent experiments as a pandas.DataFrame

update(*uuid*, *parameters*)

Update the parameters of an empty experiment given its uuid

2.2.6 Notebooks

class sklearn_evaluation.**NotebookCollection**(*paths*, *ids=None*, *scores=False*)

Compare output from a collection of notebooks

To access output, notebooks must tag the cells (one tag per cell). For instructions on tagging cells, see this.

Click here to see the user guide.

Parameters

- **paths** (*list*) – Paths to notebooks to load
- **ids** (*list or 'filenames', default=None*) – List of ids (one per notebook), if None, paths are used as identifiers, if 'filenames', the file name is extracted from each path and used as identifier (ignores extension)

class sklearn_evaluation.**NotebookIntrospector**(*path*, *literal_eval=True*, *to_df=False*)

Retrieve output from a notebook file with tagged cells.

For instructions on tagging cells, see this.

Notes

Ignores untagged cells, if a cell has more than one tag, it uses the first one as identifier. If a cell has more than one output, it uses the last one and discards the rest.

CHAPTER

THREE

LICENSE

The MIT License (MIT)

Copyright (c) 2016 Eduardo Blancas Reyes

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

S

T

U

V