## **History**

This project was built by Google Summer of Code project by David Cournapeau in 2007. Matthieu Brucher started work on this project as part of his thesis.

In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel of INRIA took leadership of the project and made the first public release, February the 1st 2010.

Several releases have appeared following this time with a 3 months interval [1].

[Scikit-learn: Machine Learning in Python](http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html)

To implement machine learning, you need to manipulate multiple sets. The essential to achieve this goal [1,2]:

3 labeled set into several subsets included:

* a training set to train your machine learning models;
* a set of tests to measure the performance of your models on untrained data. The performance of a model on the test set corresponds to a measurement of this model on real data.
* using a validation set to determine the best hyper-parameters for your models. This is for gaining the best parameterization of your models without using the test set.

It is necessary that all the sets you manipulate are representative of the case to be modeled. The testing the model on data should not correspond to the training data.

Generally, random cuts are made in the dataset to determine the subsets.

Example done by done by Scikit-learn:

Select

1.  
2.  
3.  
4.

from sklearn.model\_selection import train\_test\_split

# L'ensemble de test aura 20 % des éléments de départ.

# L'ensemble d'entrainement contiendra les 80 % restant.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.3)

Cross validation

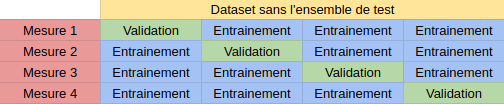
The concern with using a test set is that if you make your training set to isolate a test set, then your dataset is restricted, if we restrict it even more this may penalize learning:

The more generalize model's performance measurements on this small set , the more concern with using a test set is that it is necessarily smaller, and

. A small set will have more difficulty in remaining representative of the real data.

This is why we will often choose to use cross-validation to refine the hyper-parameters of a model.

**Cross-validation**, [3,4]sometimes called **rotation estimation**[[5]](https://en.wikipedia.org/wiki/Cross-validation_(statistics)" \l "cite_note-5)[[6]](https://en.wikipedia.org/wiki/Cross-validation_(statistics)" \l "cite_note-Kohavi95-6)[[7]](https://en.wikipedia.org/wiki/Cross-validation_(statistics)" \l "cite_note-Devijver82-7) or**out-of-sample testing**, is a [model validation](https://en.wikipedia.org/wiki/Model_validation) techniques for assessing how the results of a [statistical](https://en.wikipedia.org/wiki/Statistics) analysis could be  [generalize](https://en.wikipedia.org/wiki/Generalization_error) on data set.



Scikit-learn already implements this slicing mechanism:

from sklearn.model\_selection import cross\_val\_score

model = ...

scores = cross\_val\_score(model, X, Y, cv=5) #cv est le nombre de découpages à réaliser

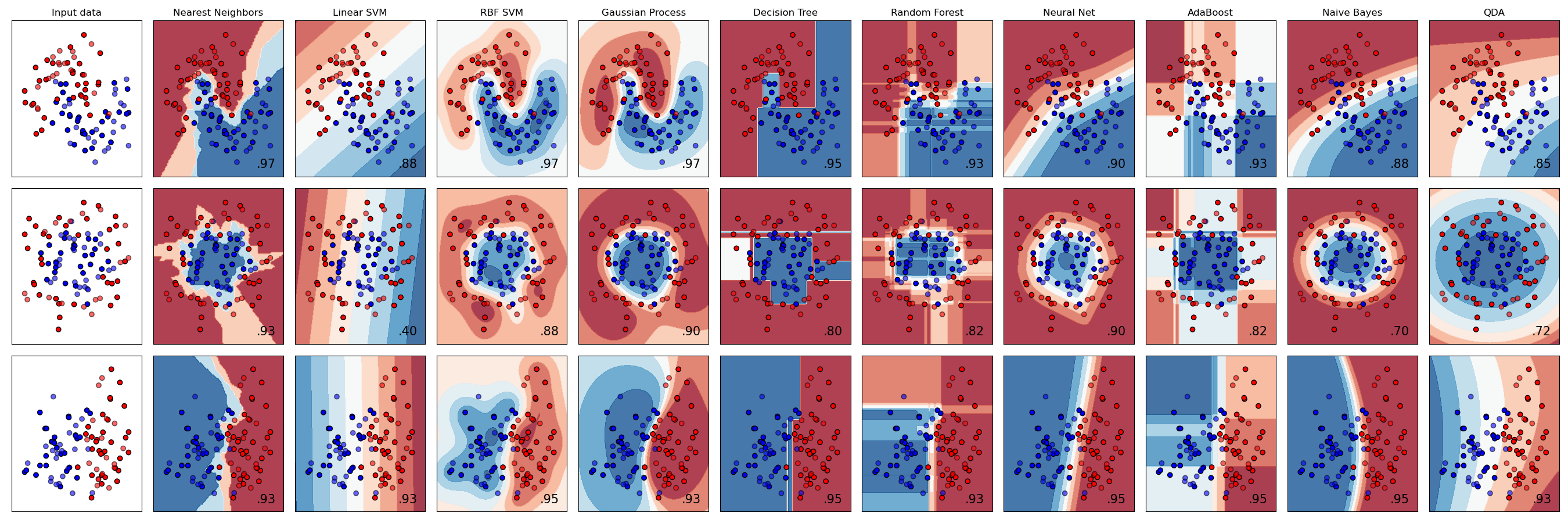
score = scores.mean()

# **Classifier comparison**

A comparison of a several classifiers in scikit-learn on synthetic datasets. This iss an example to show the nature of decision boundaries of different classifiers.[1] This is similar to a grain of salt.

In sepecial cases such as high-dimensional spaces, data can more easily be separated linearly and the simplicity of classifiers such as naive Bayes and linear SVMs.

The plots below show training points in solid colors and testing points semi-transparent. The lower right indicates the classification accuracy on the test set.



*# Code source: Gaël Varoquaux*

*# Andreas Müller*

*# Modified for documentation by Jaques Grobler*

*# License: BSD 3 clause*

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**from** **matplotlib.colors** **import** [ListedColormap](https://matplotlib.org/stable/api/_as_gen/matplotlib.colors.ListedColormap.html" \l "matplotlib.colors.ListedColormap)

**from** **sklearn.model\_selection** **import** [train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html" \l "sklearn.model_selection.train_test_split)

**from** **sklearn.preprocessing** **import** [StandardScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html" \l "sklearn.preprocessing.StandardScaler)

**from** **sklearn.datasets** **import** [make\_moons](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html" \l "sklearn.datasets.make_moons), [make\_circles](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_circles.html" \l "sklearn.datasets.make_circles), [make\_classification](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html" \l "sklearn.datasets.make_classification)

**from** **sklearn.neural\_network** **import** [MLPClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html" \l "sklearn.neural_network.MLPClassifier)

**from** **sklearn.neighbors** **import** [KNeighborsClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html" \l "sklearn.neighbors.KNeighborsClassifier)

**from** **sklearn.svm** **import** [SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html" \l "sklearn.svm.SVC)

**from** **sklearn.gaussian\_process** **import** [GaussianProcessClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessClassifier.html" \l "sklearn.gaussian_process.GaussianProcessClassifier)

**from** **sklearn.gaussian\_process.kernels** **import** [RBF](https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RBF.html" \l "sklearn.gaussian_process.kernels.RBF)

**from** **sklearn.tree** **import** [DecisionTreeClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html" \l "sklearn.tree.DecisionTreeClassifier)

**from** **sklearn.ensemble** **import** [RandomForestClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html" \l "sklearn.ensemble.RandomForestClassifier), [AdaBoostClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html" \l "sklearn.ensemble.AdaBoostClassifier)

**from** **sklearn.naive\_bayes** **import** [GaussianNB](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html" \l "sklearn.naive_bayes.GaussianNB)

**from** **sklearn.discriminant\_analysis** **import** [QuadraticDiscriminantAnalysis](https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html" \l "sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis)

**from** **sklearn.inspection** **import** DecisionBoundaryDisplay

names = [

"Nearest Neighbors",

"Linear SVM",

"RBF SVM",

"Gaussian Process",

"Decision Tree",

"Random Forest",

"Neural Net",

"AdaBoost",

"Naive Bayes",

"QDA",

]

classifiers = [

[KNeighborsClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html" \l "sklearn.neighbors.KNeighborsClassifier)(3),

[SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html" \l "sklearn.svm.SVC)(kernel="linear", C=0.025),

[SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html" \l "sklearn.svm.SVC)(gamma=2, C=1),

[GaussianProcessClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessClassifier.html" \l "sklearn.gaussian_process.GaussianProcessClassifier)(1.0 \* [RBF](https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RBF.html" \l "sklearn.gaussian_process.kernels.RBF)(1.0)),

[DecisionTreeClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html" \l "sklearn.tree.DecisionTreeClassifier)(max\_depth=5),

[RandomForestClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html" \l "sklearn.ensemble.RandomForestClassifier)(max\_depth=5, n\_estimators=10, max\_features=1),

[MLPClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html" \l "sklearn.neural_network.MLPClassifier)(alpha=1, max\_iter=1000),

[AdaBoostClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html" \l "sklearn.ensemble.AdaBoostClassifier)(),

[GaussianNB](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html" \l "sklearn.naive_bayes.GaussianNB)(),

[QuadraticDiscriminantAnalysis](https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html" \l "sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis)(),

]

X, y = [make\_classification](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html" \l "sklearn.datasets.make_classification)(

n\_features=2, n\_redundant=0, n\_informative=2, random\_state=1, n\_clusters\_per\_class=1

)

rng = [np.random.RandomState](https://numpy.org/doc/stable/reference/random/legacy.html" \l "numpy.random.RandomState)(2)

X += 2 \* rng.uniform(size=X.shape)

linearly\_separable = (X, y)

datasets = [

[make\_moons](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html" \l "sklearn.datasets.make_moons)(noise=0.3, random\_state=0),

[make\_circles](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_circles.html" \l "sklearn.datasets.make_circles)(noise=0.2, factor=0.5, random\_state=1),

linearly\_separable,

]

figure = [plt.figure](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.figure.html" \l "matplotlib.pyplot.figure)(figsize=(27, 9))

i = 1

*# iterate over datasets*

**for** ds\_cnt, ds **in** enumerate(datasets):

*# preprocess dataset, split into training and test part*

X, y = ds

X = [StandardScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html" \l "sklearn.preprocessing.StandardScaler)().fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = [train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html" \l "sklearn.model_selection.train_test_split)(

X, y, test\_size=0.4, random\_state=42

)

x\_min, x\_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5

y\_min, y\_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5

*# just plot the dataset first*

cm = plt.cm.RdBu

cm\_bright = [ListedColormap](https://matplotlib.org/stable/api/_as_gen/matplotlib.colors.ListedColormap.html" \l "matplotlib.colors.ListedColormap)(["#FF0000", "#0000FF"])

ax = [plt.subplot](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot.html" \l "matplotlib.pyplot.subplot)(len(datasets), len(classifiers) + 1, i)

**if** ds\_cnt == 0:

ax.set\_title("Input data")

*# Plot the training points*

ax.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap=cm\_bright, edgecolors="k")

*# Plot the testing points*

ax.scatter(

X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap=cm\_bright, alpha=0.6, edgecolors="k"

)

ax.set\_xlim(x\_min, x\_max)

ax.set\_ylim(y\_min, y\_max)

ax.set\_xticks(())

ax.set\_yticks(())

i += 1

*# iterate over classifiers*

**for** name, clf **in** zip(names, classifiers):

ax = [plt.subplot](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot.html" \l "matplotlib.pyplot.subplot)(len(datasets), len(classifiers) + 1, i)

clf.fit(X\_train, y\_train)

score = clf.score(X\_test, y\_test)

[DecisionBoundaryDisplay.from\_estimator](https://scikit-learn.org/stable/modules/generated/sklearn.inspection.DecisionBoundaryDisplay.html" \l "sklearn.inspection.DecisionBoundaryDisplay.from_estimator)(

clf, X, cmap=cm, alpha=0.8, ax=ax, eps=0.5

)

*# Plot the training points*

ax.scatter(

X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap=cm\_bright, edgecolors="k"

)

*# Plot the testing points*

ax.scatter(

X\_test[:, 0],

X\_test[:, 1],

c=y\_test,

cmap=cm\_bright,

edgecolors="k",

alpha=0.6,

)

ax.set\_xlim(x\_min, x\_max)

ax.set\_ylim(y\_min, y\_max)

ax.set\_xticks(())

ax.set\_yticks(())

**if** ds\_cnt == 0:

ax.set\_title(name)

ax.text(

x\_max - 0.3,

y\_min + 0.3,

("*%.2f*" % score).lstrip("0"),

size=15,

horizontalalignment="right",

)

i += 1

[plt.tight\_layout](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.tight_layout.html" \l "matplotlib.pyplot.tight_layout)()

[plt.show](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.show.html" \l "matplotlib.pyplot.show)()

# **Recognizing hand-written digits**

Second example illustrate how scikit-learn can be used to recognize images of hand-written digits, from 0-9.

*# Author: Gael Varoquaux <gael dot varoquaux at normalesup dot org>*

*# License: BSD 3 clause*

*# Standard scientific Python imports*

**import** **matplotlib.pyplot** **as** **plt**

*# Import datasets, classifiers and performance metrics*

**from** **sklearn** **import** datasets, svm, metrics

**from** **sklearn.model\_selection** **import** [train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html" \l "sklearn.model_selection.train_test_split)

Dataset

The digits dataset includes of 8x8 pixel images of digits. These images attributes of the dataset stores 8x8 arrays of grayscale values for each image[2].We will use these arrays to visualize the first 4 images. The goal attribute of the dataset preserves the digit each image represents.

we would load the images in the first step using:

**[matplotlib.pyplot.imread](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imread.html" \l "matplotlib.pyplot.imread)**.

digits = [datasets.load\_digits](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html" \l "sklearn.datasets.load_digits)()

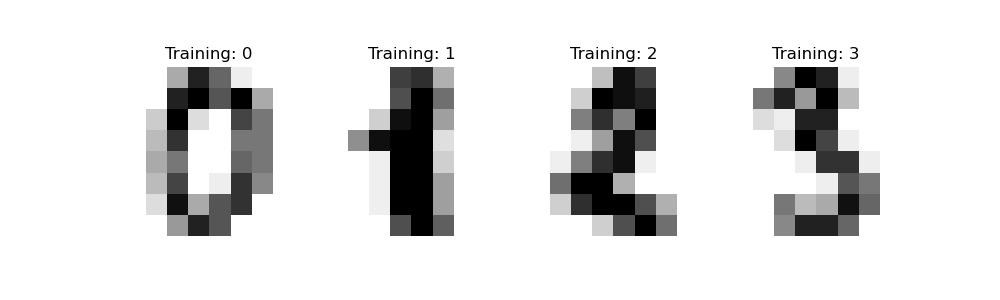
\_, axes = [plt.subplots](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html" \l "matplotlib.pyplot.subplots)(nrows=1, ncols=4, figsize=(10, 3))

**for** ax, image, label **in** zip(axes, digits.images, digits.target):

ax.set\_axis\_off()

ax.imshow(image, cmap=plt.cm.gray\_r, interpolation="nearest")

ax.set\_title("Training: *%i*" % label)



## **Classification**

Second step in this procedure is to turning each 2-D array of grayscale values from shape (8, 8)into shape (64,). Subsequently, the entire dataset will be of shape (n\_samples, n\_features), where n\_samples is the number of images and n\_features is the total number of pixels in each image.

We can then split the data into train and test subsets and fit a support vector classifier on the train samples. The fitted classifier can subsequently be used to predict the value of the digit for the samples in the test subset.

*# flatten the images*

n\_samples = len(digits.images)

data = digits.images.reshape((n\_samples, -1))

*# Create a classifier: a support vector classifier*

clf = [svm.SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html" \l "sklearn.svm.SVC)(gamma=0.001)

*# Split data into 50% train and 50% test subsets*

X\_train, X\_test, y\_train, y\_test = [train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html" \l "sklearn.model_selection.train_test_split)(

data, digits.target, test\_size=0.5, shuffle=**False**

)

*# Learn the digits on the train subset*

clf.fit(X\_train, y\_train)

*# Predict the value of the digit on the test subset*

predicted = clf.predict(X\_test)

Below we visualize the first 4 test samples and show their predicted digit value in the title.

\_, axes = [plt.subplots](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html" \l "matplotlib.pyplot.subplots)(nrows=1, ncols=4, figsize=(10, 3))

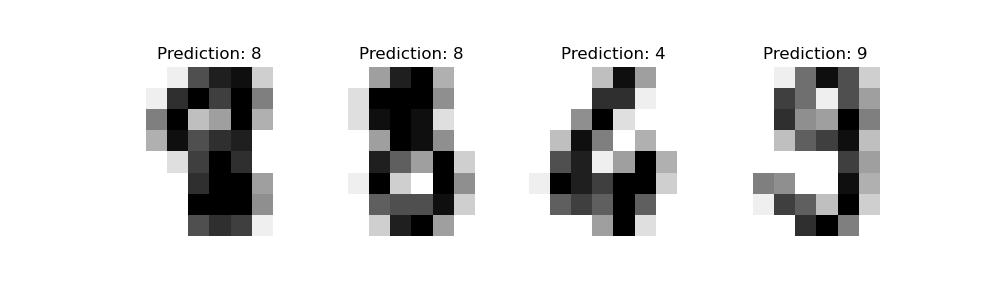
**for** ax, image, prediction **in** zip(axes, X\_test, predicted):

ax.set\_axis\_off()

image = image.reshape(8, 8)

ax.imshow(image, cmap=plt.cm.gray\_r, interpolation="nearest")

ax.set\_title(f"Prediction: *{*prediction*}*")



**classification\_report** builds a text report showing the main classification metrics.

print(

f"Classification report for classifier *{*clf*}*:**\n**"

f"*{*[metrics.classification\_report](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html" \l "sklearn.metrics.classification_report)(y\_test, predicted)*}***\n**"

)

Out:

Classification report for classifier SVC(gamma=0.001):

precision recall f1-score support

0 1.00 0.99 0.99 88

1 0.99 0.97 0.98 91

2 0.99 0.99 0.99 86

3 0.98 0.87 0.92 91

4 0.99 0.96 0.97 92

5 0.95 0.97 0.96 91

6 0.99 0.99 0.99 91

7 0.96 0.99 0.97 89

8 0.94 1.00 0.97 88

9 0.93 0.98 0.95 92

accuracy 0.97 899

macro avg 0.97 0.97 0.97 899

weighted avg 0.97 0.97 0.97 899

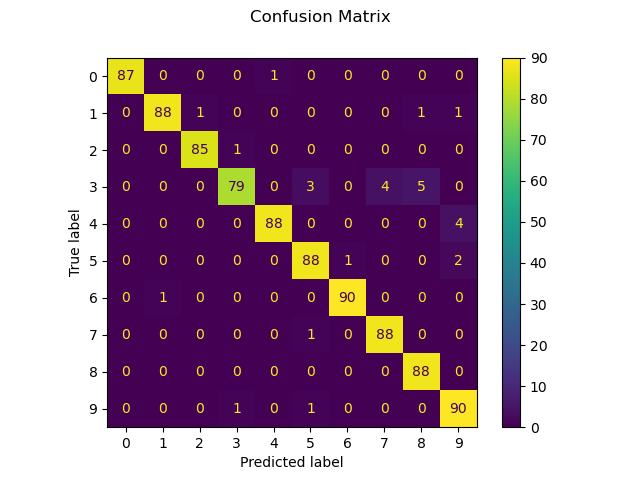
We can also plot a [confusion matrix](https://scikit-learn.org/stable/modules/model_evaluation.html" \l "confusion-matrix) of the true digit values and the predicted digit values.

disp = [metrics.ConfusionMatrixDisplay.from\_predictions](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html" \l "sklearn.metrics.ConfusionMatrixDisplay.from_predictions)(y\_test, predicted)

disp.figure\_.suptitle("Confusion Matrix")

print(f"Confusion matrix:**\n***{*disp.confusion\_matrix*}*")

[plt.show](https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.show.html" \l "matplotlib.pyplot.show)()



Confusion matrix:

[[87 0 0 0 1 0 0 0 0 0]

[ 0 88 1 0 0 0 0 0 1 1]

[ 0 0 85 1 0 0 0 0 0 0]

[ 0 0 0 79 0 3 0 4 5 0]

[ 0 0 0 0 88 0 0 0 0 4]

[ 0 0 0 0 0 88 1 0 0 2]

[ 0 1 0 0 0 0 90 0 0 0]

[ 0 0 0 0 0 1 0 88 0 0]

[ 0 0 0 0 0 0 0 0 88 0]

[ 0 0 0 1 0 1 0 0 0 90]]

*Reference:*

1) sklearn\_api, Lars Buitinck,Gilles Louppe, Mathieu Blondel, Fabian Pedregosa,Andreas Mueller, Olivier Grisel,Vlad Niculae,Peter Prettenhofer ,Alexandre Gramfort,Jaques Grobler, Robert Layton,Jake VanderPlas ,Arnaud Joly and Brian Holt, API design for machine learning software: experiences from the scikit-learn project, ECML PKDD Workshop: Languages for Data Mining and Machine Learning, 2013, 108--122,

2) Scikit-learn: Machine Learning in Python, Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V., and Thirion, B. and Grisel, O. and Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D. and Brucher, M., Perrot, M.,Duchesnay, E.},Journal of Machine Learning Research,12,2825–2830, 2011

**3)** Stone, M (1974). "Cross-Validatory Choice and Assessment of Statistical Predictions". *Journal of the Royal Statistical Society, Series B (Methodological)*. **36** (2): 111–147. [doi](https://en.wikipedia.org/wiki/Doi_(identifier)):[10.1111/j.2517-6161.1974.tb00994.x](https://doi.org/10.1111%2Fj.2517-6161.1974.tb00994.x).

**4) [^](https://en.wikipedia.org/wiki/Cross-validation_(statistics)" \l "cite_ref-4)** Stone, M (1977). "An Asymptotic Equivalence of Choice of Model by Cross-Validation and Akaike's Criterion". *Journal of the Royal Statistical Society, Series B (Methodological)*. **39** (1): 44–47. [doi](https://en.wikipedia.org/wiki/Doi_(identifier)):[10.1111/j.2517-6161.1977.tb01603.x](https://doi.org/10.1111%2Fj.2517-6161.1977.tb01603.x). [JSTOR](https://en.wikipedia.org/wiki/JSTOR_(identifier)) [2984877](https://www.jstor.org/stable/2984877).

Scikit-Learn is a community driven project, however institutional and private grants help to assure its sustainability.

The project would like to thank the following funders.

The [Members](https://scikit-learn.fondation-inria.fr/en/home/" \l "sponsors) of the [Scikit-Learn Consortium at Inria Foundation](https://scikit-learn.fondation-inria.fr/en/home/) fund Olivier Grisel, Guillaume Lemaitre, and Jérémie du Boisberranger.

[Hugging Face](https://huggingface.co/) funds Adrin Jalali since 2022.



[Microsoft](https://microsoft.com/) funds Andreas Müller since 2020.



[Quansight Labs](https://labs.quansight.org/) funds Thomas J. Fan since 2021.