Classification & SVM with scikit-learn

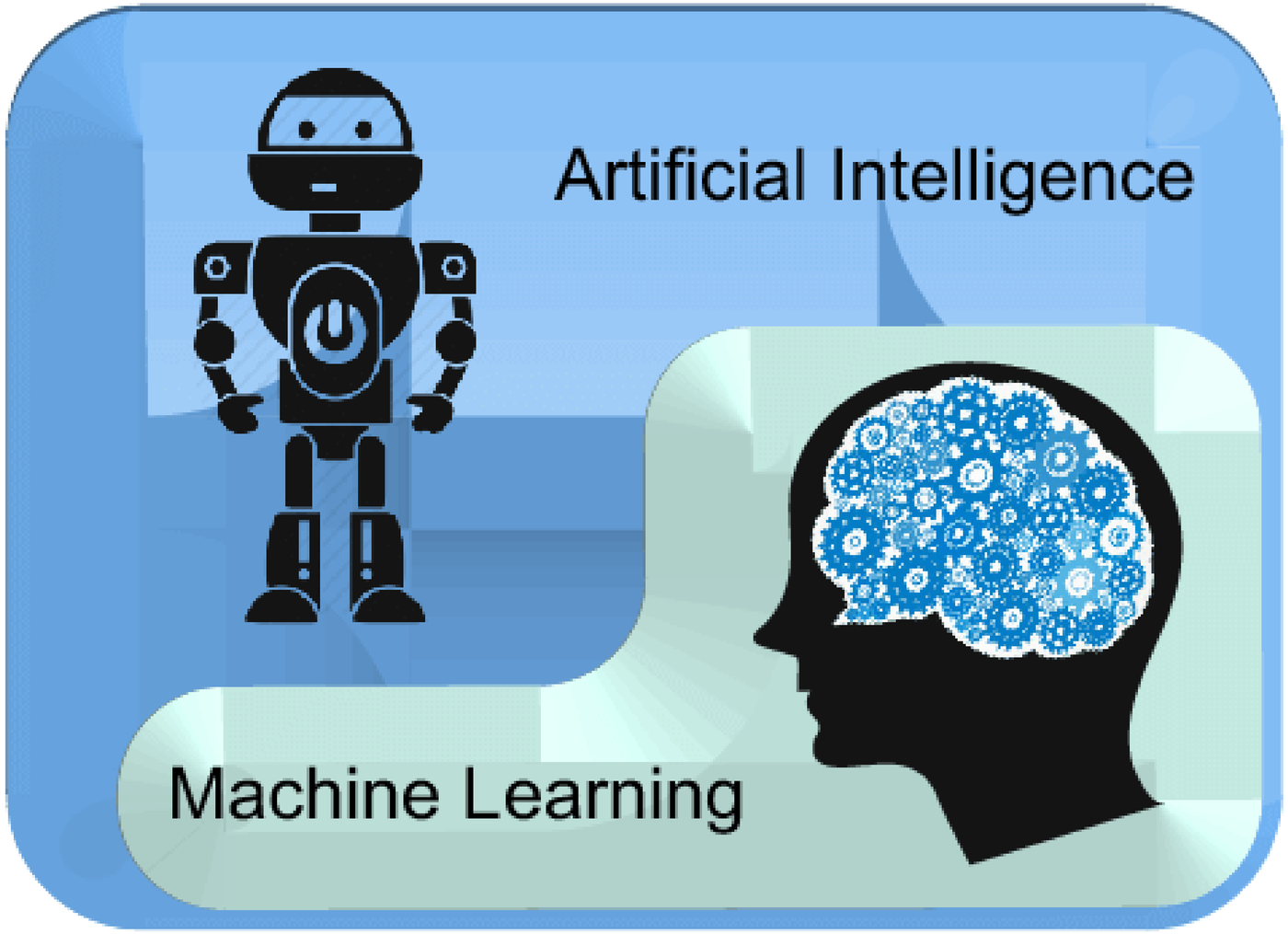
Gloria Beraldo (gloria.beraldo@unipd.it)
Department of Information Engineering, University of Padova

Topics:

- Introduction
- Scikit-learn for machine learning in Python
- Datasets in scikit-learn
- Digits dataset in scikit-learn
- Support Vector Machines (SVM): Recap
- SVM in scikit-learn
- train_test_split in scikit-learn
- Fit and predict in scikit-learn
- Metrics for classification & Confusion Matrix in scikit-learn
- Cross-validation & K-fold cross-validation in scikit-learn

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

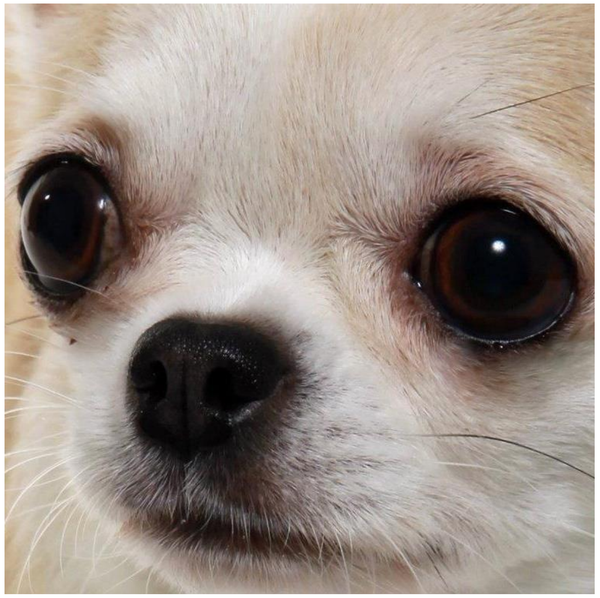
Introduction



AI is a bigger concept to create intelligent machines that can simulate human thinking capability and behavior, whereas,

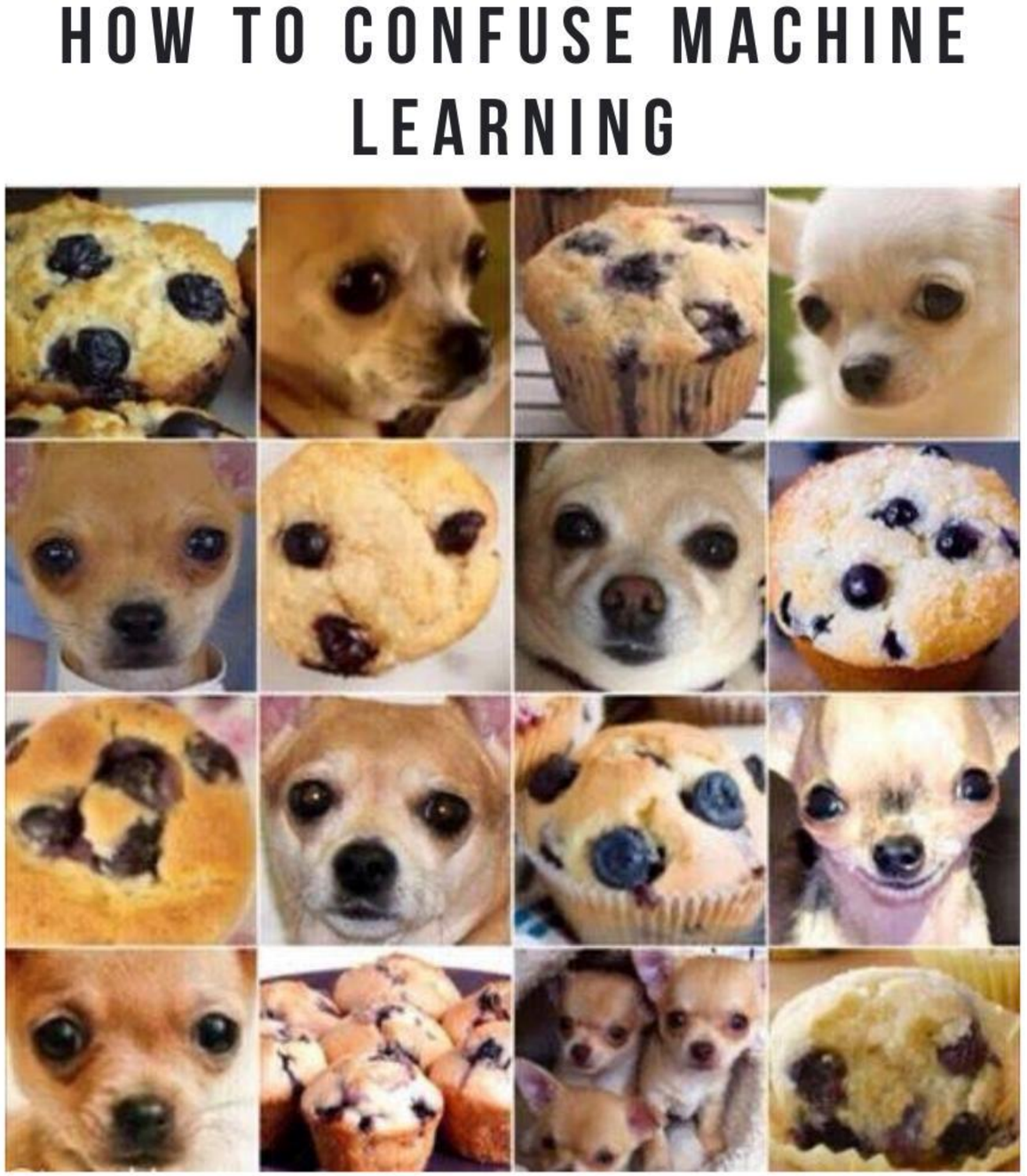
machine learning is an application or subset of AI that allows machines to learn from data without being programmed explicitly

Introduction



Example of samples classified as Chihuahua

Machine learning algorithms can **make mistakes**

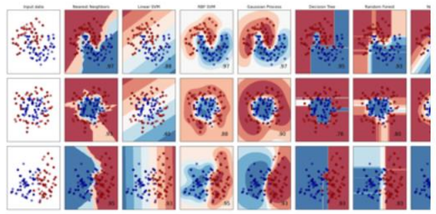


Scikit-learn for machine learning in Python

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

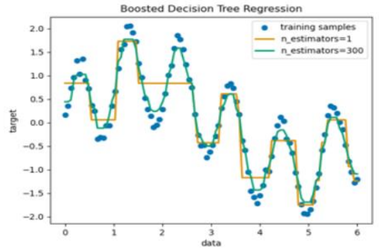


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...



Examples

Clustering

Automatic grouping of similar objects into sets.

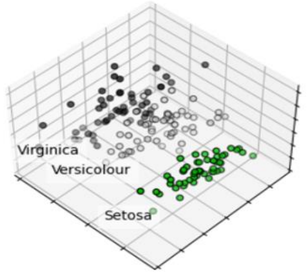
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

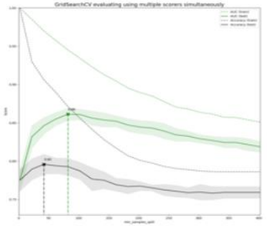


Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

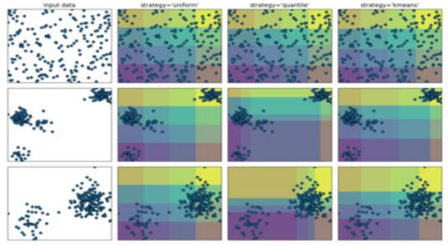


Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...



Examples

Simple and efficient tool for predictive data analysis

Accessible to everybody, and reusable in many contexts

Open source, commercially available

Who is using scikit-learn?

Spotify

Inria INVENTORS FOR THE DIGITAL WORLD

DataRobot

BNP PARIBAS CARDIF

Booking.com

betaworks

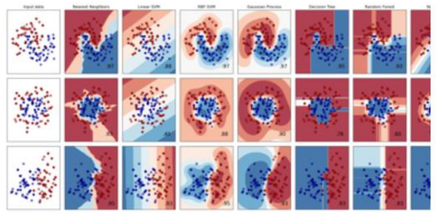
https://scikit-learn.org/stable/

Scikit-learn for machine learning in Python

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

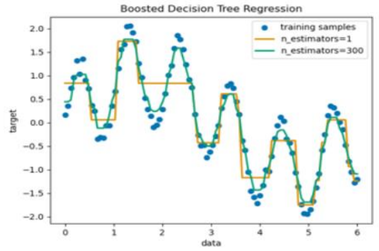


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

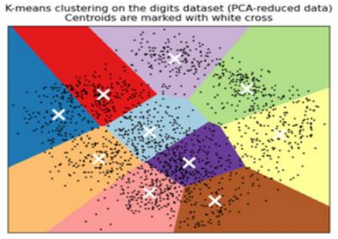


Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

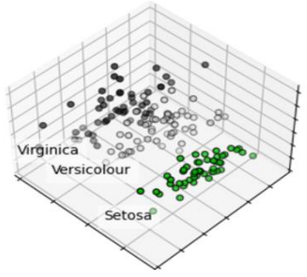


Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

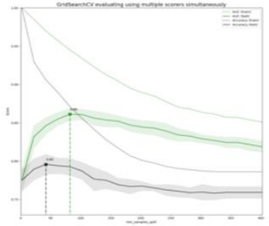


Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

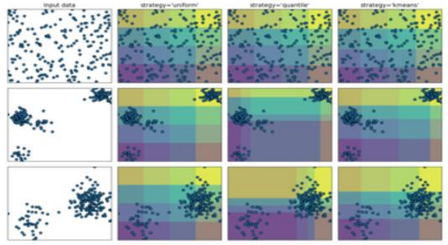


Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...



Examples

Simple and efficient tool for predictive data analysis

Accessible to everybody, and reusable in many contexts

Open source, commercially available

To install it:

```
pip install scikit-learn
```

https://scikit-learn.org/stable/

Datasets in scikit-learn

scikit-learn provides a few small standard datasets that do not require to download any file from some external website.

These datasets are useful to quickly illustrate the behavior of the various algorithms implemented in scikit-learn. They are however often too small to be representative of real world machine learning tasks.

For instance, the iris and digits datasets for classification and the diabetes dataset for regression.

They can be loaded using the following functions:
load_datasetname

load_iris(*[, return_X_y, as_frame])	Load and return the iris dataset (classification).
load_diabetes(*[, return_X_y, as_frame, scaled])	Load and return the diabetes dataset (regression).
load_digits(*[, n_class, return_X_y, as_frame])	Load and return the digits dataset (classification).
load_linnerud(*[, return_X_y, as_frame])	Load and return the physical exercise Linnerud dataset.
load_wine(*[, return_X_y, as_frame])	Load and return the wine dataset (classification).
load_breast_cancer(*[, return_X_y, as_frame])	Load and return the breast cancer wisconsin dataset (classification).

https://scikit-learn.org/stable/datasets/toy_dataset.html

Digits dataset in scikit-learn

The digits dataset consists of 8x8 pixel images of digits.
The images attribute of the dataset stores 8x8 arrays of grayscale values for each image.

Classes	10
Samples per class	~180
Samples total	1797
Dimensionality	64
Features	integers 0-16

Returns:

data : *Bunch*

Dictionary-like object, with the following attributes.

data : *{ndarray, dataframe} of shape (1797, 64)*
The flattened data matrix. If `as_frame=True`, `data` will be a pandas DataFrame.

target: {ndarray, Series} of shape (1797,)
The classification target. If `as_frame=True`, `target` will be a pandas Series.

feature_names: list
The names of the dataset columns.

target_names: list
The names of target classes.

New in version 0.20.

frame: DataFrame of shape (1797, 65)
Only present when `as_frame=True`. DataFrame with `data` and `target`.

New in version 0.23.

images: {ndarray} of shape (1797, 8, 8)
The raw image data.

DESCR: str
The full description of the dataset.

(data, target) : *tuple if `return_X_y` is True*

This is a copy of the test set of the
UCI ML hand-written digits datasets:
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html

Digits dataset in scikit-learn

Let's load the dataset now and apply some functions in order to better understand the kind of data.

```
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as np
digits_data = datasets.load_digits()
print(digits_data.data)
```

We import the datasets from scikit-learn

We load the digit dataset

```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
```

We can check the dimensionality

```
print(digits_data.data.shape)
```

```
(1797, 64)
```

8 x 8 images of digits

We can check the classes

```
print(np.unique(digits_data.target))
print(digits_data.target.shape)
```

```
[0 1 2 3 4 5 6 7 8 9]
(1797,)
```

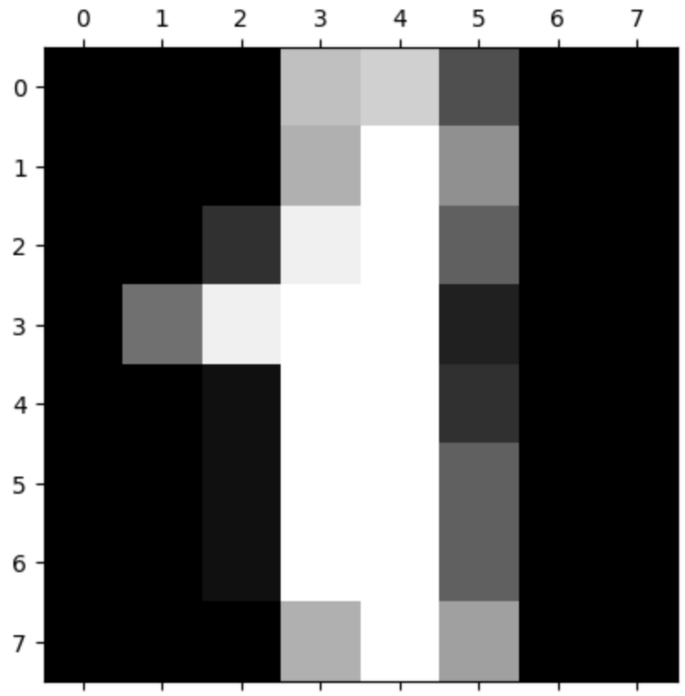
The number between 0 and 9

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html

Digits dataset in scikit-learn

Since the digits dataset contains images, we can also graphically display the samples.

```
plt.gray()
plt.matshow(digits_data.images[1])
plt.show()
```



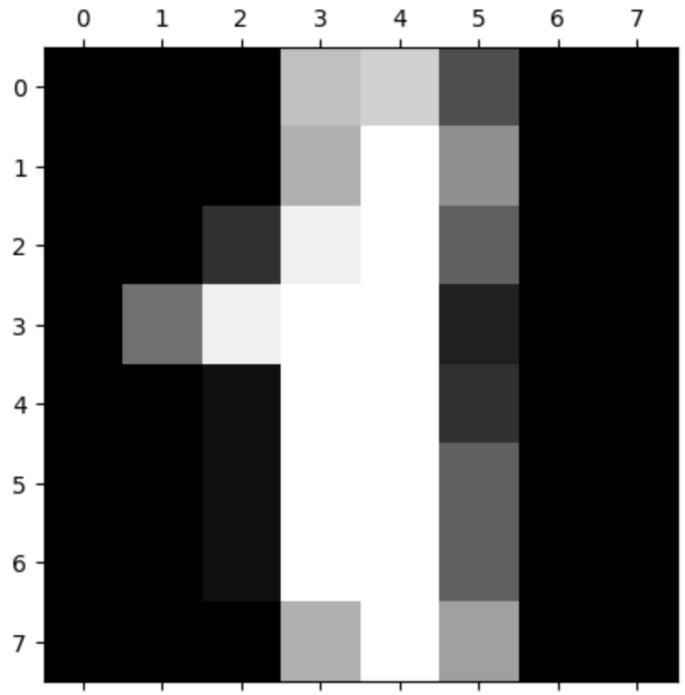
What is the label associated with this sample?

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html

Digits dataset in scikit-learn

Since the digits dataset contains images, we can also graphically display the samples.

```
plt.gray()
plt.matshow(digits_data.images[1])
plt.show()
```



Returns:

data : *Bunch*

Dictionary-like object, with the following attributes.

data : ***{ndarray, dataframe} of shape (1797, 64)***

The flattened data matrix. If `as_frame=True`, `data` will be a pandas DataFrame.

target: {ndarray, Series} of shape (1797,)

The classification target. If `as_frame=True`, `target` will be a pandas Series.

feature_names: list

The names of the dataset columns.

target_names: list

The names of target classes.

New in version 0.20.

frame: DataFrame of shape (1797, 65)

Only present when `as_frame=True`. DataFrame with `data` and `target`.

New in version 0.23.

images: {ndarray} of shape (1797, 8, 8)

The raw image data.

DESCR: str

The full description of the dataset.

(data, target) : ***tuple if `return_X_y` is True***

To retrieve the label, we can check the target

```
print(digits_data.target[1])
```

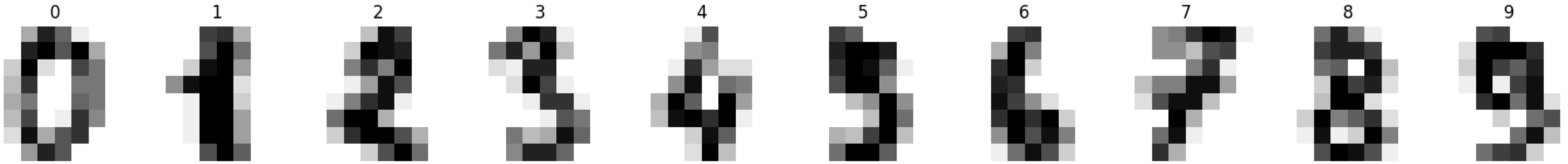
```
1
```

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html

Digits dataset in scikit-learn

We can also visualize other samples to get confidence with the datasets:

```
_, axes = plt.subplots(nrows=1, ncols=10, figsize=(20, 3))
for ax, image, label in zip(axes, digits_data.images, digits_data.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("%i" % label)
```



https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html

Support Vector Machines (SVM): Recap

SVMs are a set of supervised learning methods used for classification, regression and outliers detection.

PRO:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile → different Kernel functions

CONS:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.

SVM in scikit-learn

SVC, NuSVC and LinearSVC are classes capable of performing binary and multi-class classification on a dataset.

class `sklearn.svm.`**`SVC`**(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

class `sklearn.svm.`**`NuSVC`**(**, nu=0.5, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

class `sklearn.svm.`**`LinearSVC`**(*penalty='l2', loss='squared_hinge', *, dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000*) [source]

No kernel is provided, since it is linear as default – FAST IMPLEMENTATION

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html#sklearn.svm.NuSVC

https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC

SVM in scikit-learn

For instance, let's analyze SVC:

Parameters:

C : *float, default=1.0*
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*
Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, default=3*
Degree of the polynomial kernel function ('poly'). Must be non-negative. Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

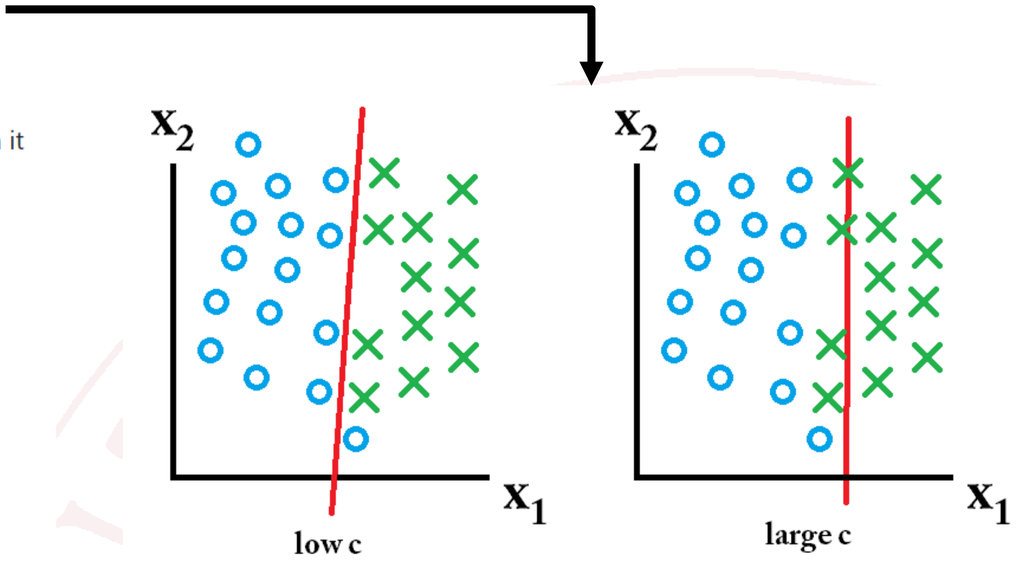
- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features
- if float, must be non-negative.

Changed in version 0.22: The default value of `gamma` changed from 'auto' to 'scale'.

coef0 : *float, default=0.0*
Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

shrinking : *bool, default=True*
Whether to use the shrinking heuristic. See the User Guide.

probability : *bool, default=False*
Whether to enable probability estimates. This must be enabled prior to calling `fit`, will slow down that method as it internally uses 5-fold cross-validation, and `predict_proba` may be inconsistent with `predict`. Read more in the User Guide.



C is a hypermeter in SVM to controls the trade-off between maximizing the margin and minimizing the classification errors on the training data.

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

SVM in scikit-learn

For instance, let's analyze SVC:

Parameters:

C : *float, default=1.0*
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*
Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, default=3*
Degree of the polynomial kernel function ('poly'). Must be non-negative. Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features
- if float, must be non-negative.

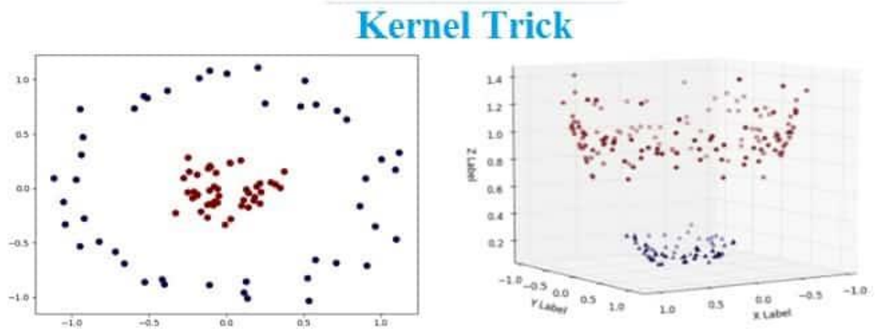
Changed in version 0.22: The default value of `gamma` changed from 'auto' to 'scale'.

coef0 : *float, default=0.0*
Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

shrinking : *bool, default=True*
Whether to use the shrinking heuristic. See the User Guide.

probability : *bool, default=False*
Whether to enable probability estimates. This must be enabled prior to calling `fit`, will slow down that method as it internally uses 5-fold cross-validation, and `predict_proba` may be inconsistent with `predict`. Read more in the User Guide.

The kernel is a crucial component that allows SVMs to handle non-linearly separable data by transforming the input features into a higher-dimensional feature space.



https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

SVM in scikit-learn

For instance, let's analyze SVC:

Parameters:

C : *float, default=1.0*
Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'*
Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, default=3*
Degree of the polynomial kernel function ('poly'). Must be non-negative. Ignored by all other kernels.

gamma : *{'scale', 'auto'} or float, default='scale'*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

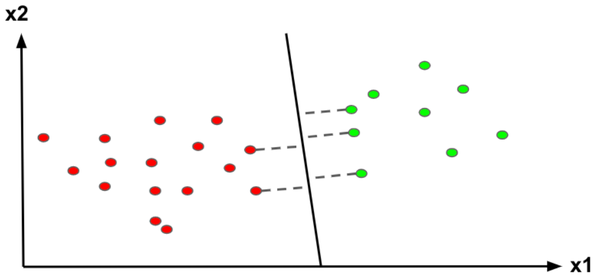
- if `gamma='scale'` (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features
- if float, must be non-negative.

Changed in version 0.22: The default value of `gamma` changed from 'auto' to 'scale'.

coef0 : *float, default=0.0*
Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

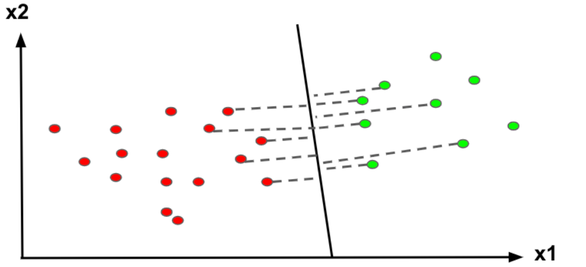
shrinking : *bool, default=True*
Whether to use the shrinking heuristic. See the User Guide.

probability : *bool, default=False*
Whether to enable probability estimates. This must be enabled prior to calling `fit`, will slow down that method as it internally uses 5-fold cross-validation, and `predict_proba` may be inconsistent with `predict`. Read more in the User Guide.



High Gamma

- only near points are considered.



Low Gamma

- far away points are also considered

γ is a hypermeter in SVM to determine the influence of a single training example on the decision boundary.

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

SVM in scikit-learn: Example

For example, we can create a SVM classifier based on SVC with gamma=0.001, while the other parameters have the default value.

```
from sklearn import svm
# Create a classifier: a support vector classifier
clf = svm.SVC(gamma=0.001)
```

C : *float*, *default=1.0*

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable*, *default='rbf'*

Specifies the kernel type to be used in the algorithm. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

We are going to train a SVM on the digits dataset.

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

SVM in scikit-learn

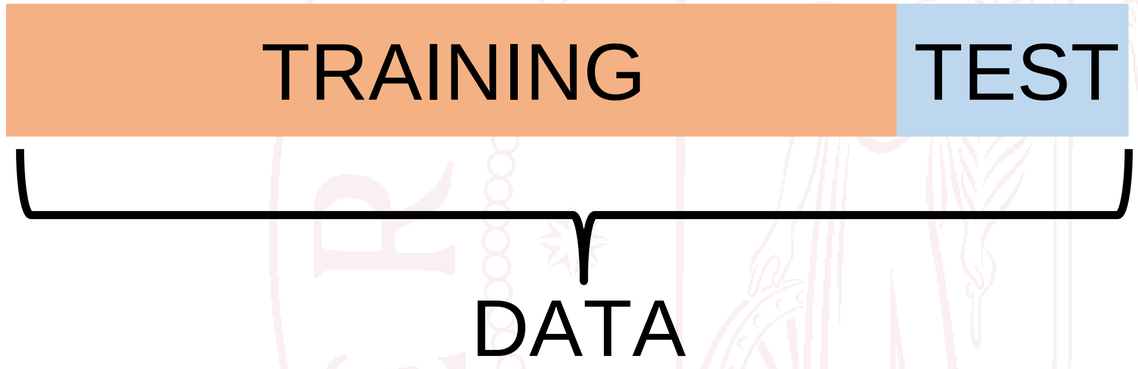
We are going to train a SVM on the digits dataset.

Before passing the data, we need to:

- flatten the images, turning each 2-D array of grayscale values from shape (8, 8) into shape (64,)

```
# flatten the images
n_samples = len(digits_data.images)
data = digits_data.images.reshape((n_samples, -1))
```

- Split the dataset in training and test parts.



To split the datasets in the training and the test parts, we can use the function train_test_split

train_test_split in scikit-learn

Split the dataset in training and test parts with the train_test_split function

sklearn.model_selection.train_test_split(**arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None*) [source]

Parameters:

arrays : sequence of indexables with same length / shape[0]
Allowed inputs are lists, numpy arrays, scipy-sparse matrices or pandas dataframes.

test_size : float or int, default=None
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the test split. If int, represents the absolute number of test samples. If None, the value is set to the complement of the train size. If `train_size` is also None, it will be set to 0.25.

train_size : float or int, default=None
If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the train split. If int, represents the absolute number of train samples. If None, the value is automatically set to the complement of the test size.

random_state : *int, RandomState instance or None, default=None*
Controls the shuffling applied to the data before applying the split. Pass an int for reproducible output across multiple function calls. See Glossary.

shuffle : *bool, default=True*
Whether or not to shuffle the data before splitting. If shuffle=False then stratify must be None.

stratify : *array-like, default=None*
If not None, data is split in a stratified fashion, using this as the class labels. Read more in the User Guide.

Returns:

splitting : *list, length=2 * len(arrays)*
List containing train-test split of inputs.

New in version 0.16: If the input is sparse, the output will be a `scipy.sparse.csr_matrix`. Else, output type is the same as the input type.

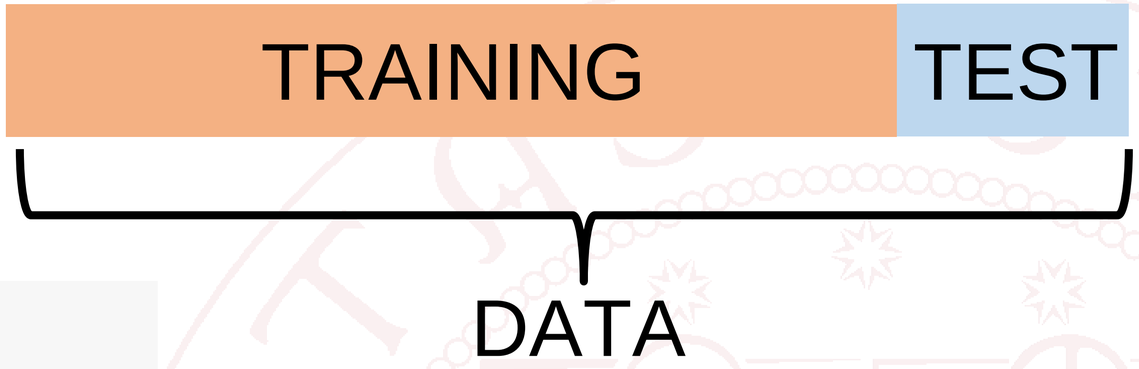
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

train_test_split in scikit-learn: Example

Split the dataset in training and test parts with the train_test_split function

For instance, in our example, we split the dataset as follows:

75% training
25% test




```
from sklearn.model_selection import train_test_split

# Split data into 75% train and 25% test subsets
X_train, X_test, y_train, y_test = train_test_split(
    data, digits_data.target, test_size=0.25, shuffle=False
)
```

Let's fit the SVM that we defined before on the training set.

fit in scikit-learn

Let's fit the SVM that we defined before on the training set. To do that, we can use the fit function:

fit(*X*, *y*, *sample_weight=None*) [source]

Fit the SVM model according to the given training data.

Parameters:	**X : *{array-like, sparse matrix} of shape (n_samples, n_features) or (n_samples, n_samples)*** Training vectors, where `n_samples` is the number of samples and `n_features` is the number of fea kernel="precomputed", the expected shape of X is (n_samples, n_samples). **y : *array-like of shape (n_samples,)*** Target values (class labels in classification, real numbers in regression). **sample_weight : *array-like of shape (n_samples,), default=None*** Per-sample weights. Rescale C per sample. Higher weights force the classifier to put more emphas these points.
Returns:	**self : *object*** Fitted estimator.

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.fit

fit in scikit-learn: Example

Let's fit the SVM that we defined before on the training set, with the fit function:

```
# Learn the digits on the train subset
clf.fit(X_train, y_train)
```

```
SVC(gamma=0.001)
```

The fitted classifier can subsequently be used to **predict** the value of the digit for the samples in the test subset.

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.fit

predict in scikit-learn

The fitted classifier can subsequently be used to **predict** the value of the digit for the samples in the test subset. To do that, we can use the **predict** function:

predict(*X*) [source]

Perform classification on samples in X.

For an one-class model, +1 or -1 is returned.

Parameters:	**X : *{array-like, sparse matrix} of shape (n_samples, n_features) or (n_samples_test, n_samples_train)*** For kernel="precomputed", the expected shape of X is (n_samples_test, n_samples_train).
Returns:	**y_pred : *ndarray of shape (n_samples,)*** Class labels for samples in X.

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.predict

predict in scikit-learn: Example

Now, we can apply the **predict** function on the fitted classifier to predict the values for the samples in the test set:

```
# Predict the value of the digit on the test subset
predicted = clf.predict(X_test)
```

Are you curious about the results?

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.predict

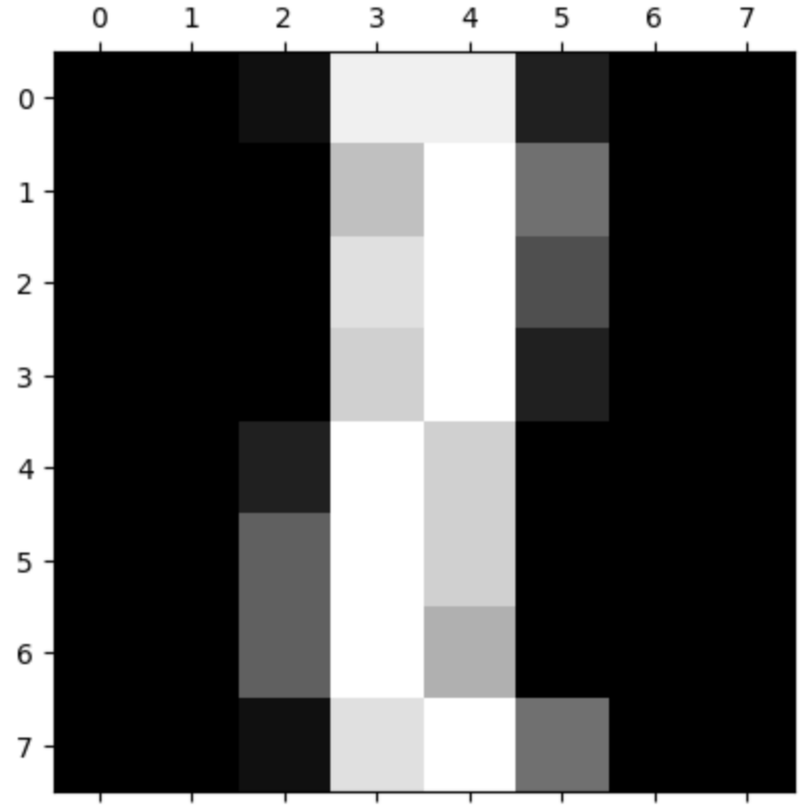
Results from prediction: Example

For instance, we can visualize one of the sample in the test set and see the predicted class.

```
# Select an id
id = 10

# Show the image
plt.gray()
plt.matshow(X_test[id].reshape(8,8))
plt.show()

# Print the prediction
print(predicted[id])
```

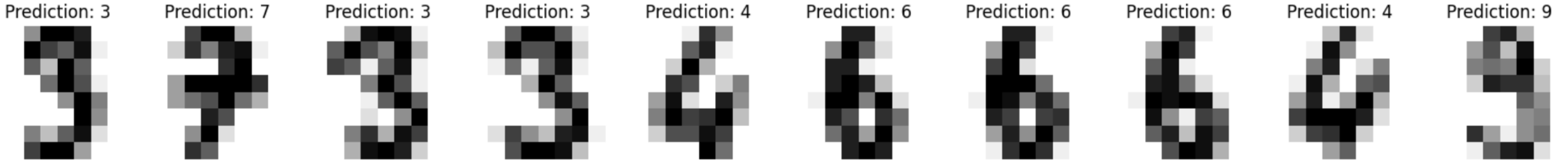


As you can see, it is a challenging task: after all, the images are of poor resolution. Do you agree with the classifier?

Results from prediction: Example

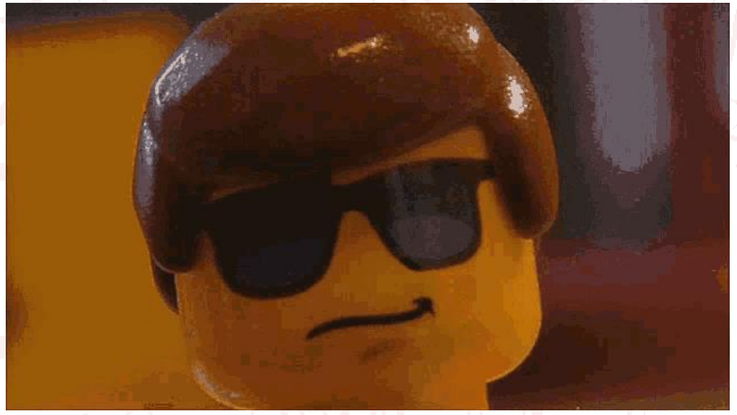
We can display other samples similarly how we did before:

```
_, axes = plt.subplots(nrows=1, ncols=10, figsize=(20, 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}")
```



NB: These are **qualitative** results!

We are more interested in **numerical performance**!

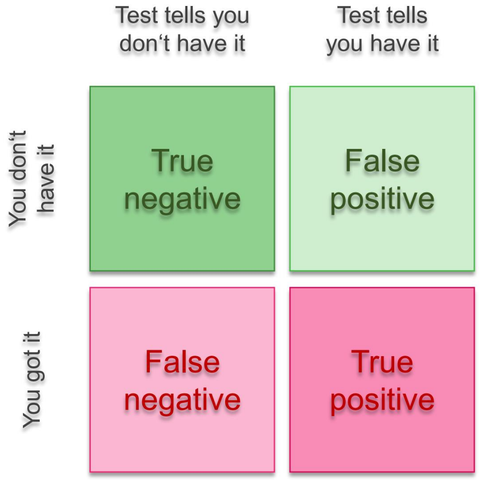


Metrics for classification in scikit-learn

Typically, it is interesting to quantitively measure the performance of the model.

In the case of classification, we can use classification_report, that builds a text report showing the main classification metrics:

- ACCURACY: $\frac{TP + TN}{TP + TN + FP + FN}$
- PRECISION: $\frac{TP}{TP+FP}$
- RECALL: $\frac{TP}{TP+FN}$
- F1 SCORE: $\frac{2*(precision*recall)}{precision+recall}$



`sklearn.metrics.classification_report`(*y_true, y_pred, *, labels=None, target_names=None, sample_weight=None, digits=2, output_dict=False, zero_division='warn'*) [source]

Build a text report showing the main classification metrics.

Read more in the User Guide.

Parameters:

y_true : ***1d array-like, or label indicator array / sparse matrix***
Ground truth (correct) target values.

y_pred : ***1d array-like, or label indicator array / sparse matrix***
Estimated targets as returned by a classifier.

labels : ***array-like of shape (n_labels,), default=None***
Optional list of label indices to include in the report.

target_names : ***list of str of shape (n_labels,), default=None***
Optional display names matching the labels (same order).

sample_weight : ***array-like of shape (n_samples,), default=None***
Sample weights.

digits : ***int, default=2***
Number of digits for formatting output floating point values. When `output_dict` is `True`, this will be ignored and the returned values will not be rounded.

output_dict : ***bool, default=False***
If True, return output as dict.

New in version 0.20.

zero_division : ***"warn", 0 or 1, default="warn"***
Sets the value to return when there is a zero division. If set to "warn", this acts as 0, but warnings are also raised.

Returns:

report : ***str or dict***
Text summary of the precision, recall, F1 score for each class. Dictionary returned if output_dict is True.

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

Classification_report in scikit-learn: Example

Let's use classification_report on the predicted data

```
from sklearn import metrics
print(
    f"Classification report for classifier {clf}:\n"
    f"{metrics.classification_report(y_test, predicted)}\n"
)
```

```
Classification report for classifier SVC(gamma=0.001):
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        43
           1       0.98      1.00      0.99        46
           2       1.00      1.00      1.00        43
           3       0.98      0.85      0.91        47
           4       0.98      0.94      0.96        48
           5       0.94      0.98      0.96        45
           6       0.98      1.00      0.99        47
           7       0.98      1.00      0.99        45
           8       0.89      0.98      0.93        41
           9       0.96      0.96      0.96        45

    accuracy                           0.97       450
   macro avg       0.97      0.97      0.97       450
weighted avg       0.97      0.97      0.97       450
```

classes

metrics

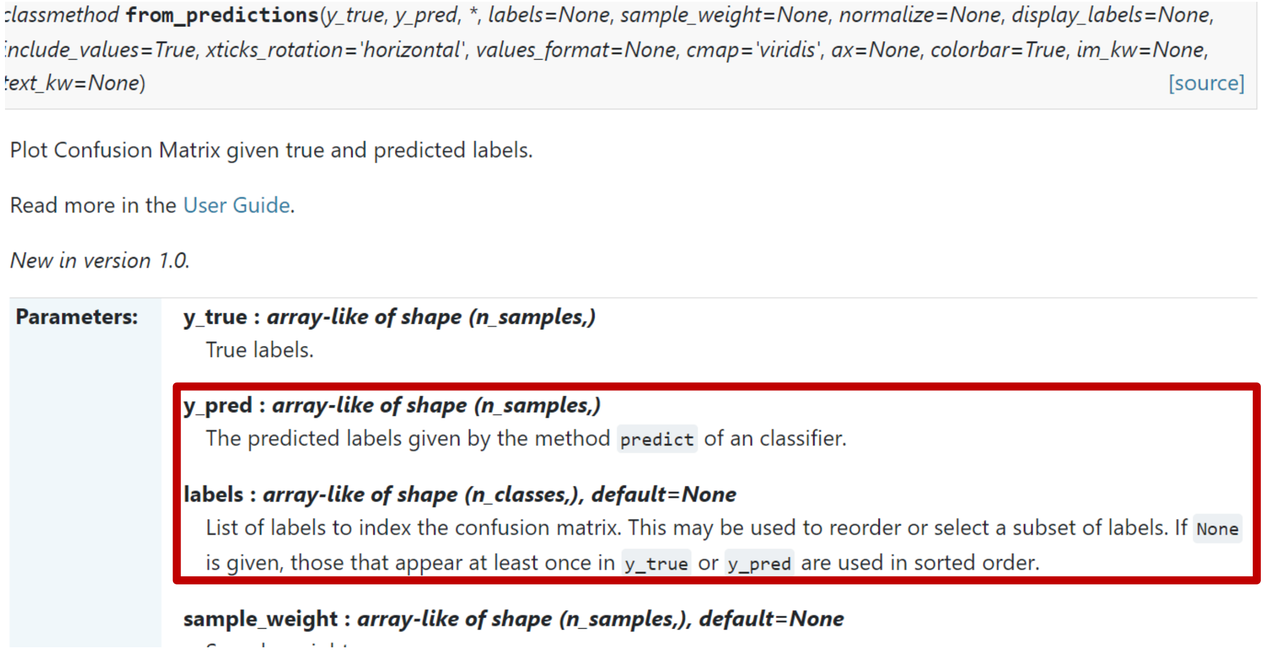
Confusion matrix in scikit-learn:

Another common visualization for showing the performance is the **confusion matrix**.
The confusion matrix is a tabular representation that provides a detailed breakdown of a model's predictions by comparing them to the actual ground truth values.

In scikit-learn, you can achieve from:

- confusion_matrix: compute Confusion Matrix to evaluate the accuracy of a classification.
- confusionMatrixDisplay.from_estimator: plot the confusion matrix given an estimator, the data, and the label.
- confusionMatrixDisplay.from_predictions: plot the confusion matrix given the true and predicted labels.

For instance, we can use confusionMatrixDisplay.from_predictions given the predictions on the test set.



classmethod **from_predictions**(*y_true, y_pred, *, labels=None, sample_weight=None, normalize=None, display_labels=None, include_values=True, xticks_rotation='horizontal', values_format=None, cmap='viridis', ax=None, colorbar=True, im_kw=None, text_kw=None*) [source]

Plot Confusion Matrix given true and predicted labels.

Read more in the User Guide.

New in version 1.0.

Parameters:

y_true : *array-like of shape (n_samples,)*
True labels.

y_pred : *array-like of shape (n_samples,)*
The predicted labels given by the method `predict` of an classifier.

labels : *array-like of shape (n_classes,), default=None*
List of labels to index the confusion matrix. This may be used to reorder or select a subset of labels. If `None` is given, those that appear at least once in `y_true` or `y_pred` are used in sorted order.

sample_weight : *array-like of shape (n_samples,), default=None*

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html#sklearn.metrics.ConfusionMatrixDisplay.from_predictions

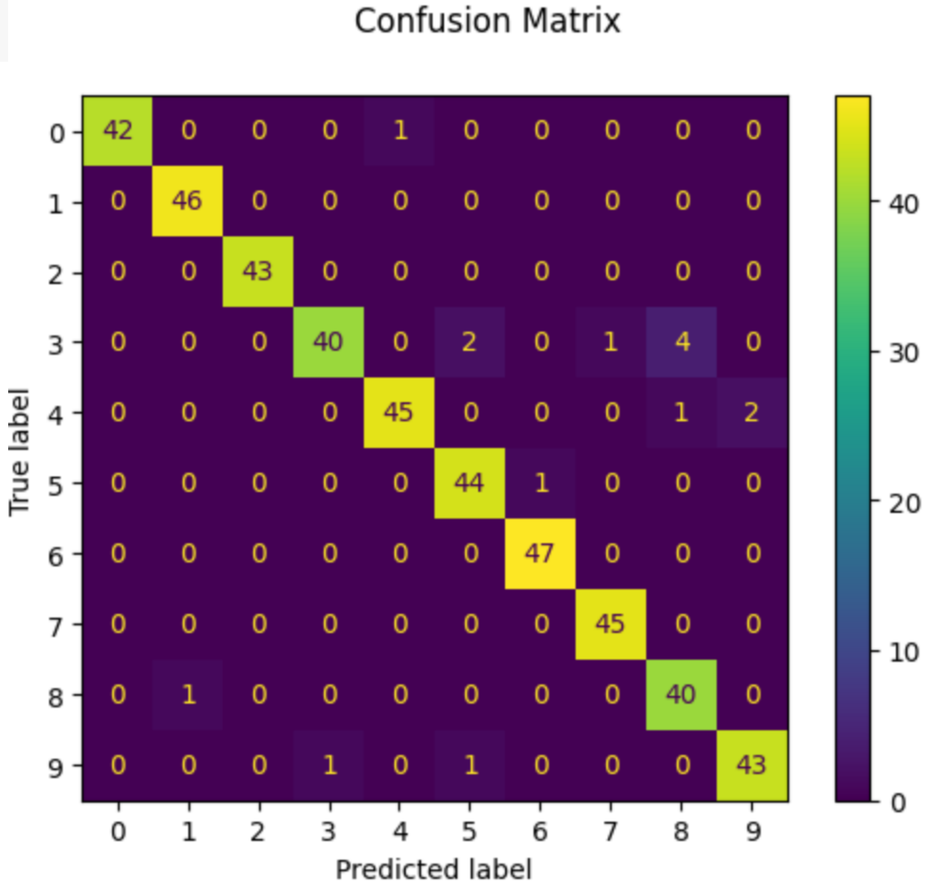
Confusion matrix in scikit-learn: Example

We can also plot a confusion matrix of the true digit values and the predicted digit values.

```
disp = metrics.ConfusionMatrixDisplay.from_predictions(y_test, predicted)
disp.figure_.suptitle("Confusion Matrix")
print(f"Confusion matrix:\n{disp.confusion_matrix}")

plt.show()
```

A confusion matrix is a tabular representation that provides a detailed breakdown of a model's predictions by comparing them to the actual ground truth values. It is particularly useful in evaluating the performance of a classification model.

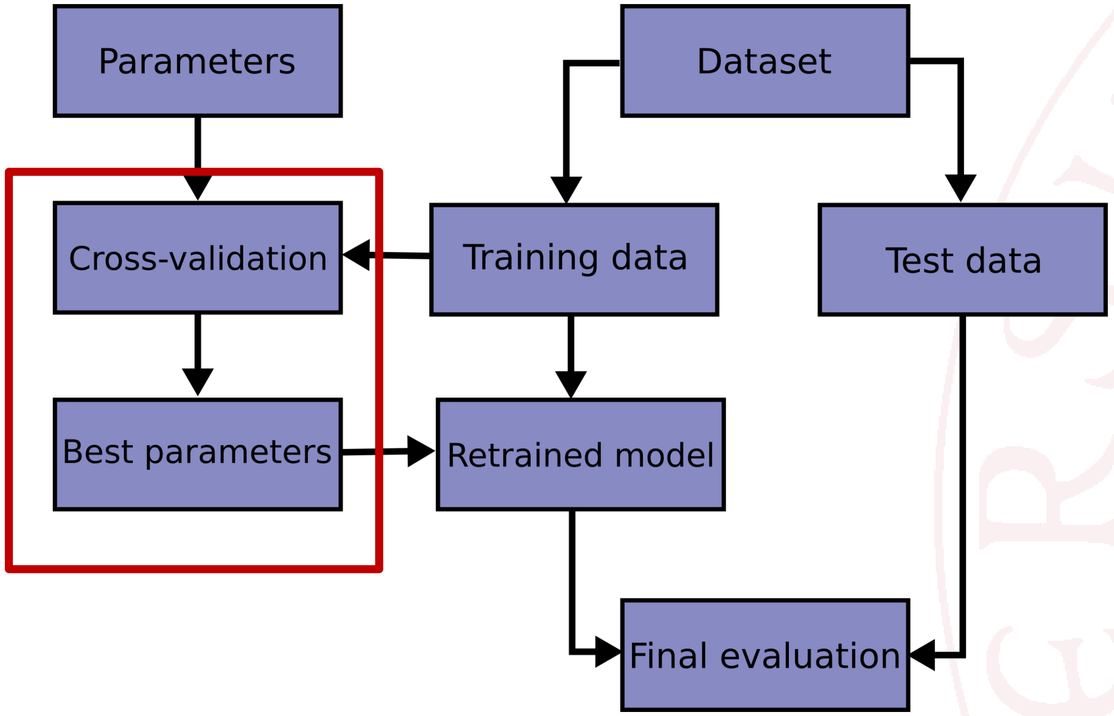


Cross-validation

In the previous example, we fixed a priori the hyperparameters of the model such as γ and C. However, typically the hyperparameters are set via cross-validation.

NB: learning the parameters of a prediction function and testing it on the same data is a methodological mistake. Indeed, the resulted model would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is called overfitting. To avoid it, it is common practice when performing a (supervised) machine learning experiment to hold out part of the available data as a test set.

different configurations are evaluated



Once, the best parameters are chosen, the model is retrained.

https://scikit-learn.org/stable/modules/cross_validation.html

Cross-validation

When evaluating different settings (“hyperparameters”) for estimators, there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally.

This way, knowledge about the test set can “leak” into the model and evaluation metrics no longer report on generalization performance.

To solve this problem, yet another part of the dataset can be held out as a so-called “**validation set**”: training proceeds on the training set, after which evaluation is done on the validation set, and when the experiment seems to be successful, final evaluation can be done on the test set.

However, by partitioning the available data into three sets, we **drastically reduce the number of samples** which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.

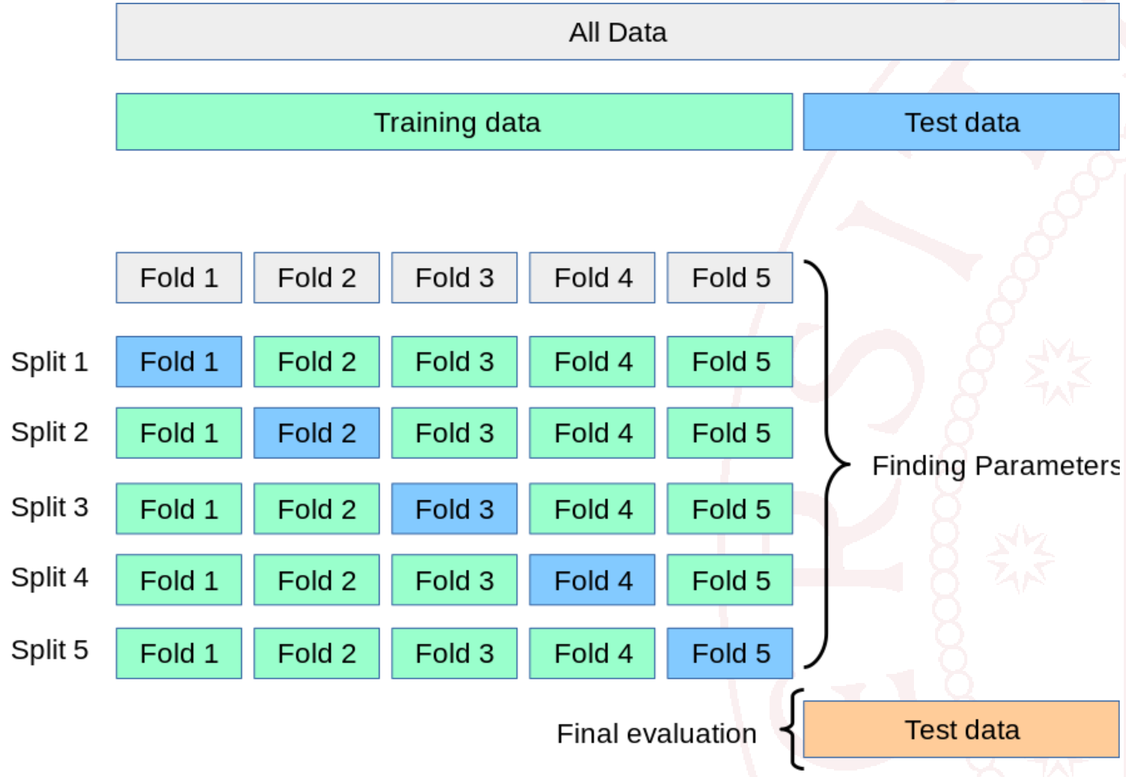
A solution to this problem is the **cross validation**. A test set should still be held out for final evaluation, but the **validation set is no longer needed** when doing cross-validation.

https://scikit-learn.org/stable/modules/cross_validation.html

K-fold cross-validation

The training set is split into k smaller sets called “folds”:
• a model is trained using **k-1 of the folds as training data**;
• the resulting model is validated on **the remaining part of the data** (i.e., it is used as a test set to compute a performance measure such as accuracy).

The final **performance** measure reported by **k-fold cross-validation is then the average of the values** computed in the loop.



https://scikit-learn.org/stable/modules/cross_validation.html

K-fold cross-validation in scikit-learn

The simplest way to use cross-validation is to call the **cross_val_score** function on the estimator and the dataset.

`sklearn.model_selection.cross_val_score(estimator, X, y=None, *, groups=None, scoring=None, cv=None, n_jobs=None, verbose=0, fit_params=None, pre_dispatch='2*n_jobs', error_score=nan)` [source]

Evaluate a score by cross-validation.

Read more in the User Guide.

Parameters:

estimator : *estimator object implementing 'fit'*
The object to use to fit the data.

It receives in input the estimator

X : *array-like of shape (n_samples, n_features)*
The data to fit. Can be for example a list, or an array.

y : *array-like of shape (n_samples,) or (n_samples, n_outputs), default=None*
The target variable to try to predict in the case of supervised learning.

groups : *array-like of shape (n_samples,), default=None*
Group labels for the samples used while splitting the dataset into train/test set. Only used in conjunction with a "Group" cv instance (e.g., `GroupKFold`).

scoring : *str or callable, default=None*
A str (see model evaluation documentation) or a scorer callable object / function with signature `scorer(estimator, X, y)` which should return only a single value.

Similar to `cross_validate` but only a single metric is permitted.

If `None`, the estimator's default scorer (if available) is used.

cv : *int, cross-validation generator or an iterable, default=None*
Determines the cross-validation splitting strategy. Possible inputs for cv are:

- `None`, to use the default 5-fold cross validation,
- int, to specify the number of folds in a `(Stratified)KFold`,
- CV splitter,
- An iterable that generates (train, test) splits as arrays of indices.

Specify the k-folder

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html#sklearn.model_selection.cross_val_score

K-fold cross-validation in scikit-learn: Example

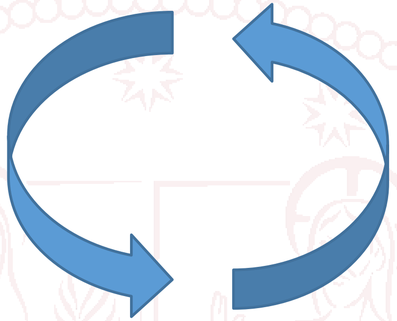
The simplest way to use cross-validation is to call the **cross_val_score** function on the estimator and the dataset. Please visit the documentation to see other cross-validation strategies.

For instance, we can apply a 5-fold cross-validation on the digit dataset:

```
#5-fold cross validation
from sklearn.model_selection import cross_val_score
clf = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf, data, digits_data.target, cv=5) # the entire (flatten) dataset
scores
```

```
array([0.96388889, 0.91944444, 0.96657382, 0.9637883 , 0.92479109])
```

the accuracy performance of each fold



Repeat with other estimators in input to choose the best configuration

The mean score and the standard deviation are hence given by:

```
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
```

```
0.95 accuracy with a standard deviation of 0.02
```

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html#sklearn.model_selection.cross_val_score

Questions