The scikit-learn package

Ezequiel López-Rubio

Department of Computer Languages and Computer Science

University of Málaga, Spain

Contents

- Introduction
- Fundamentals
- Validation
- Models
- Performance measurement
- Conclusion

1. INTRODUCTION

1. Introduction

Overview

- The scikit-learn (also known as sklearn) package is one of the best-known Python libraries for machine learning
- It provides efficient implementations of a wide range of common machine learning algorithms
- The API is quite clean and uniform
  - This facilitates the use of different machine learning models
- The usual way to import the package is:

from sklearn import …

1. Introduction

General principles

- Consistency. All objects (basic or composite) share a consistent interface composed of a limited set of methods
- Inspection. Constructor parameters and parameter values determined by learning algorithms are stored and exposed as public attributes
- Non-proliferation of classes. Learning algorithms are the only objects to be represented using custom classes
  - Datasets are represented as NumPy arrays or SciPy sparse matrices

1. Introduction

General principles

- Composition. Whenever feasible, algorithms are implemented and composed from existing building blocks
- Sensible defaults. Whenever an operation requires a user-defined parameter, an appropriate default value is defined by the library, thereby giving a baseline solution for the task at hand

2. FUNDAMENTALS

2. Fundamentals

Data representation

- A features matrix is represented as a two-dimensional array
- Each row represents a sample, i.e., an individual
- Each column represents a feature, which may be quantitative or qualitative
- Therefore, the shape of the data table is (n_samples, n_features)
- Typically, the data table is stored as an ndarray of the numpy package, or a DataFrame of the pandas package
  - Some models accept sparse matrices of the scipy package

2. Fundamentals

Data representation

- For supervised learning, a label or target array is also required, which is usually one-dimensional with shape (n_samples,)
- Sometimes multiple target values are allowed so that the target array has shape (n_samples, n_targets)
- The target array contains the output feature that we wish to predict from the features matrix
- The target array may contain numerical values (for regression problems) or discrete labels (for classification problems)

2. Fundamentals

Basic workflow

- Select a model class by importing from sklearn
- Instantiate the class by providing model hyperparameters to the class constructor
- Load and prepare the data as described before
- Fit the model to the data by executing the fit() method of the instance
- Test the model with new, unseen data:
  - For supervised learning, their labels can be estimated with the predict() method
  - For unsupervised learning, the transform() and predict() methods may be available

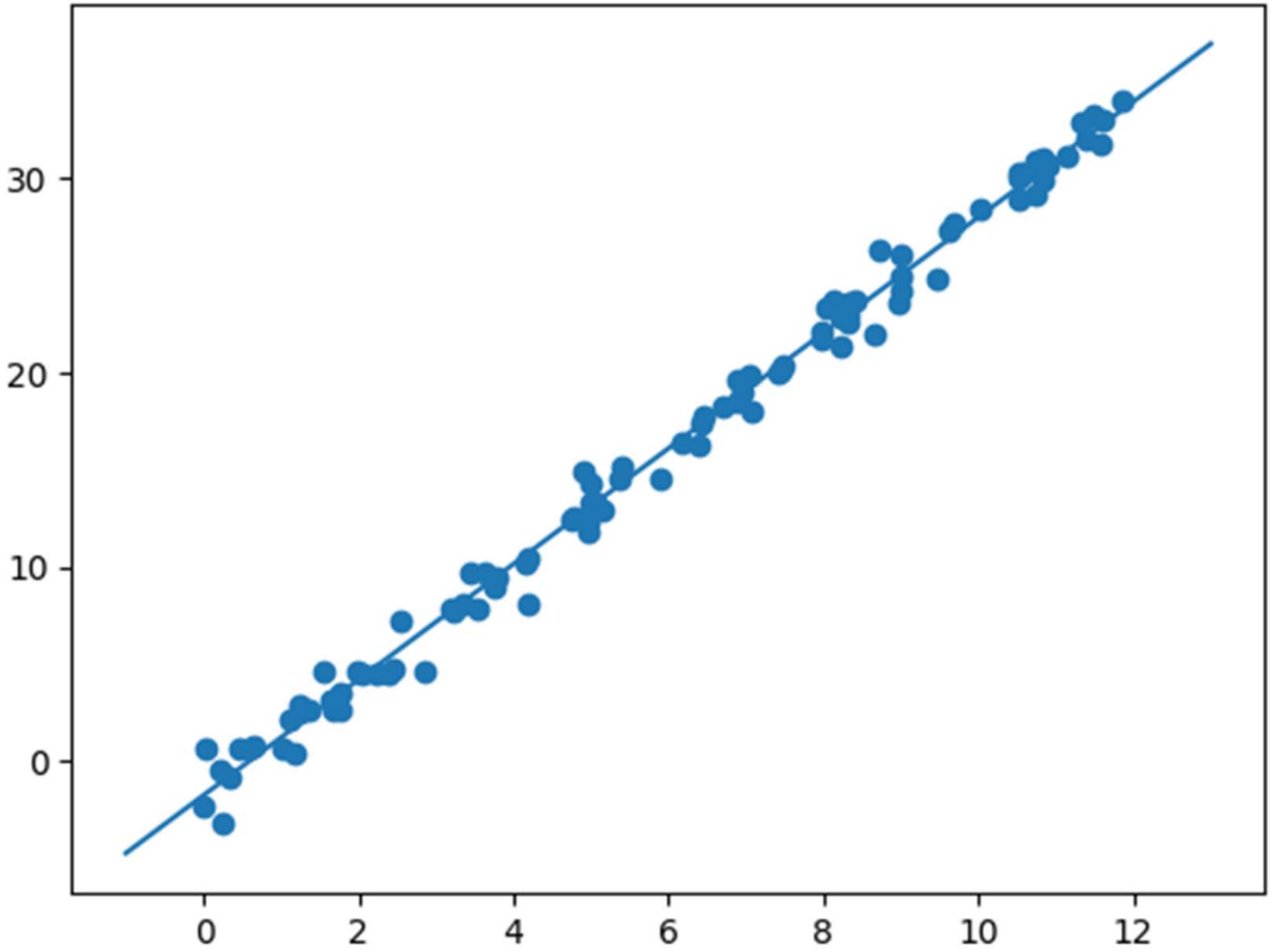
2. Fundamentals

Regression example

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
rng = np.random.RandomState(1)      # Fix the pseudorandom number generator seed
x = 12 * rng.rand(100)      # Generate some training samples for the x variable
y = 3 * x - 2 + rng.randn(100)     # Corresponding values for the y variable, with noise
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)   # Instantiate the linear regression model
X = x[:, np.newaxis]     # Convert x into a 2D array of shape (100, 1)
model.fit(X, y)             # Fit the model to the data
xfit = np.linspace(-1, 13)        # Generate test values for the x variable
Xfit = xfit[:, np.newaxis]        # Convert xfit into a 2D array
yfit = model.predict(Xfit)        # Estimate the values of the y variable for the test values of x
plt.scatter(x, y)                 # Plot the training data
plt.plot(xfit, yfit);             # Plot the estimated test values
```

2. Fundamentals

Regression example



2. Fundamentals

Classification example

```
from sklearn import datasets
iris = datasets.load_iris()              # Load the Iris standard dataset
X = iris.data                            # Set X to the features matrix
y = iris.target                          # Set y to the target array
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=1)
from sklearn.naive_bayes import GaussianNB   # Import the Naïve Bayes model class
model = GaussianNB()                     # Instantiate the model, no hyperparameters
model.fit(Xtrain, ytrain)                # Fit the model to the training data
y_model = model.predict(Xtest)           # Predict on new test data
from sklearn.metrics import accuracy_score
accuracy_score(ytest, y_model)       # Measure the classification performance

# Output (test accuracy): 0.9736842105263158
```

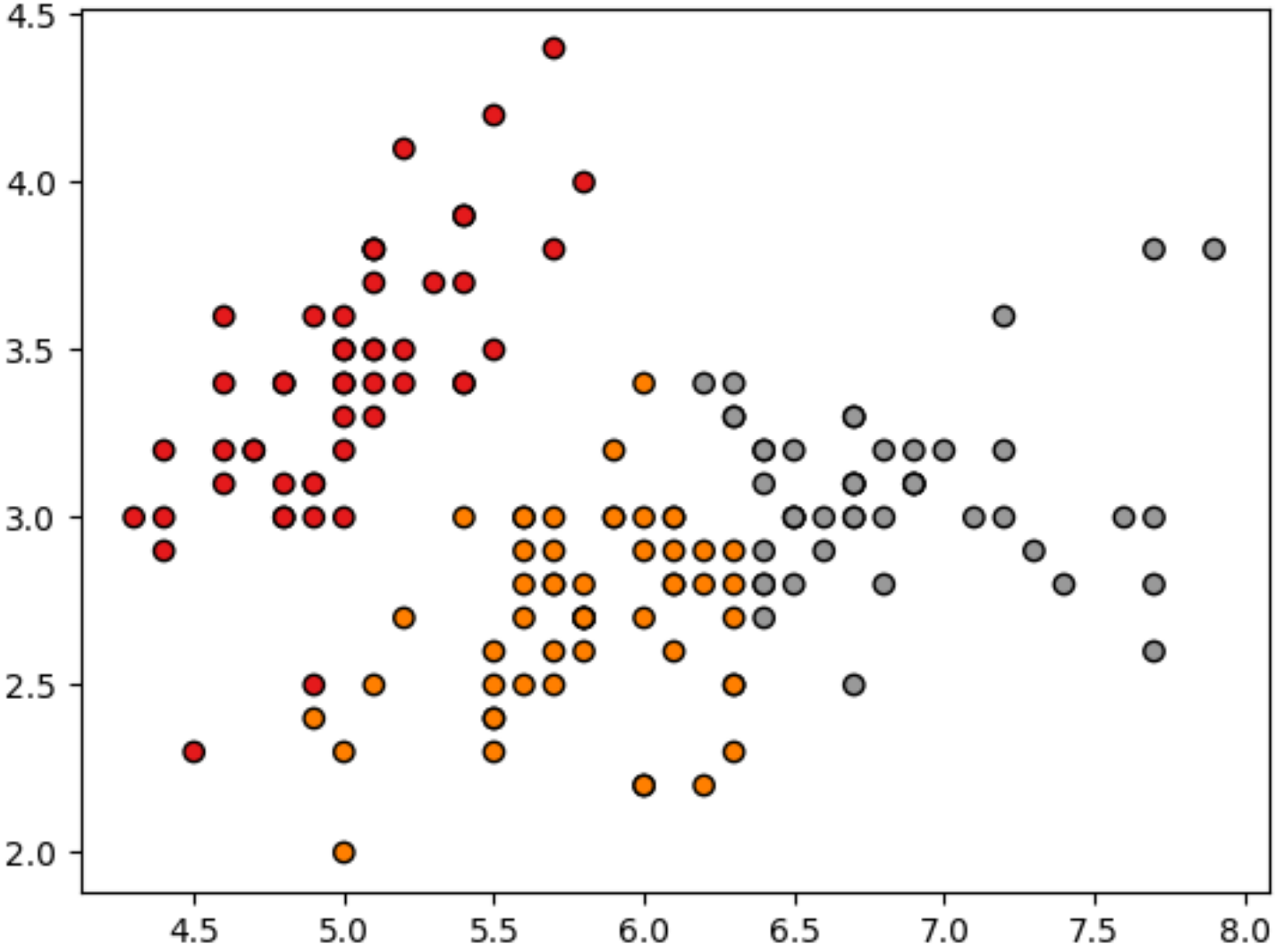
2. Fundamentals

Clustering example

```
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn import datasets
iris = datasets.load_iris()                # Load the Iris standard dataset
X = iris.data[:, :2]                       # Set X to the two first features
from sklearn.cluster import KMeans   # Import the k-means model class
model = KMeans(n_clusters=3, n_init="auto")       # Instantiate the model
model.fit(X)                               # Fit the model to the training data
y = model.predict(X)                # Predict the cluster labels
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1, edgecolor="k");
```

2. Fundamentals

Clustering example



3. VALIDATION

3. Validation
Holdout

- Holdout validation can be performed with:

sklearn.model_selection.train_test_split(*arrays, test_size, train_size, random_state, shuffle, stratify)

- *arrays are the lists, ndarrys, or DataFrames to split (same number of rows)
- test_size and train_size represent the proportions of the dataset to be included in the test/train splits
- random_state sets the pseudorandom seed for reproducibility
- shuffle indicates whether or not to shuffle the data before splitting
- stratify indicates whether the split is done in a stratified way

3. Validation
Cross-validation

- K-fold cross-validation can be performed with:

sklearn.model_selection.KFold(n_splits, shuffle, random_state)

sklearn.model_selection.StratifiedKFold(n_splits, shuffle, random_state)

- n_splits is the number of folds
- shuffle indicates whether or not to shuffle the data before splitting
- random_state sets the pseudorandom seed for reproducibility

(train_indices, test_indices) = split(X, y, groups)

- X is the features matrix
- y is the target array (supervised learning only)
- groups is an array of group labels for the samples

3. Validation

Validation curves

```
import numpy as np
from sklearn import datasets
wine = datasets.load_wine()    # Load standard wine dataset
X = wine.data                  # Obtain the features matrix
y = wine.target                # Obtain the target array
indices = np.arange(y.shape[0])   # Get the indices of the samples
np.random.shuffle(indices)        # Shuffle the sample indices
X, y = X[indices], y[indices]       # Shuffle the samples
from sklearn.svm import SVC       # Import the Support Vector Classifier model
from sklearn.model_selection import validation_curve
my_range = np.logspace(-7, 3, 10)    # Range of the C hyperparameter to try
train_scores, valid_scores = validation_curve(SVC(kernel="linear"), X, y,
param_name="C", param_range=my_range)
```

3. Validation

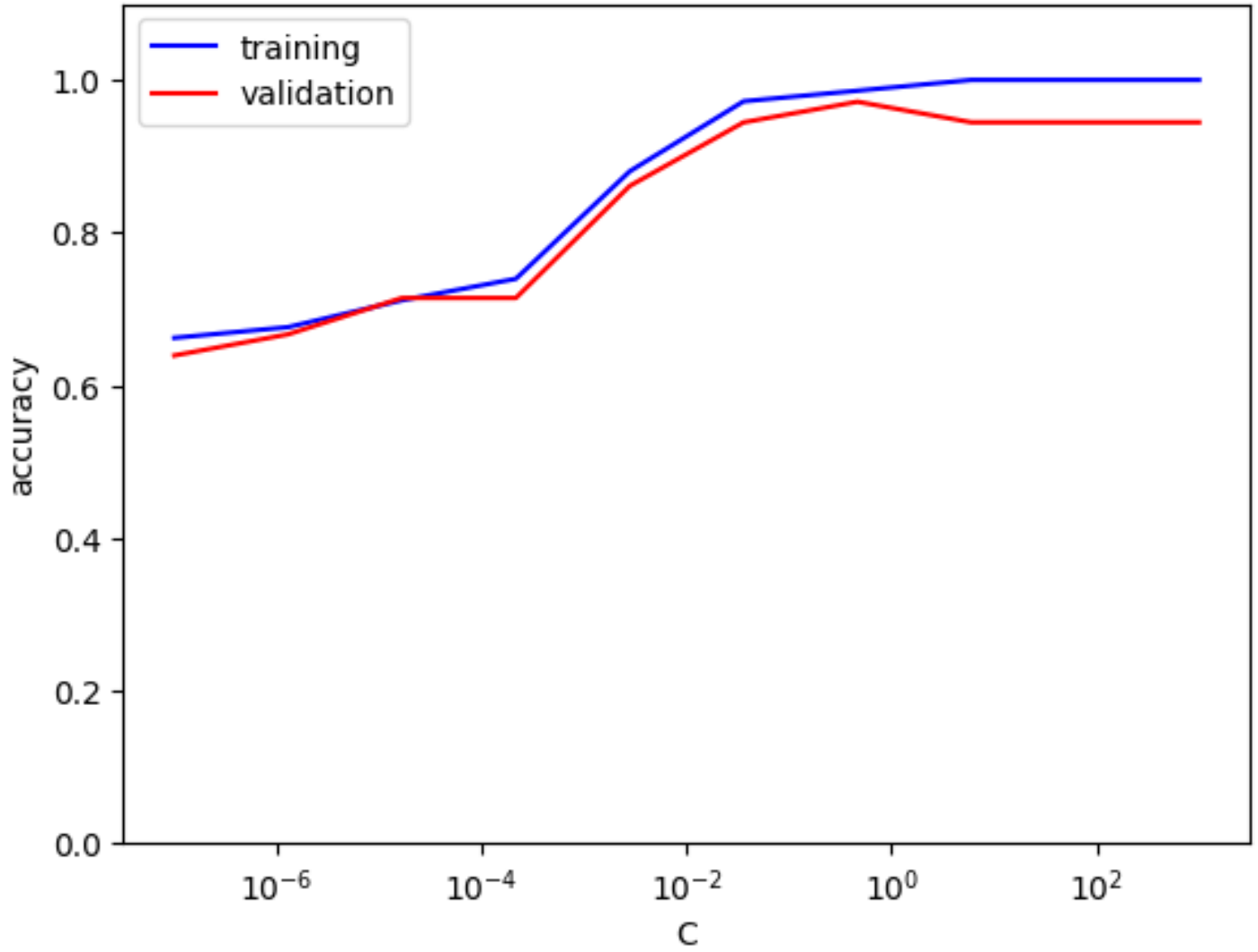
Validation curves

```
%matplotlib inline
import matplotlib.pyplot as plt


plt.plot(my_range, np.median(train_scores,1), color='blue', label='training')
plt.plot(my_range, np.median(valid_scores,1), color='red', label='validation')
plt.legend(loc='best')
plt.xscale('log')  # Set the x axis to log scale
plt.ylim(0, 1.1)    # Set the limits of the y axis
plt.xlabel('C')
plt.ylabel('accuracy');
```

3. Validation

Validation curves



3. Validation
Learning curves

```
import numpy as np
from sklearn import datasets
wine = datasets.load_wine()    # Load standard wine dataset
X = wine.data                  # Obtain the features matrix
y = wine.target                # Obtain the target array
indices = np.arange(y.shape[0])   # Get the indices of the samples
np.random.shuffle(indices)        # Shuffle the sample indices
X, y = X[indices], y[indices]       # Shuffle the samples
from sklearn.svm import SVC       # Import the Support Vector Classifier model
from sklearn.model_selection import learning_curve
my_sizes = np.linspace(0.01, 1, 25)   # Training set sizes
N, train_lc, val_lc = learning_curve(SVC(kernel="linear"), X, y, cv=10,
train_sizes=my_sizes)
```

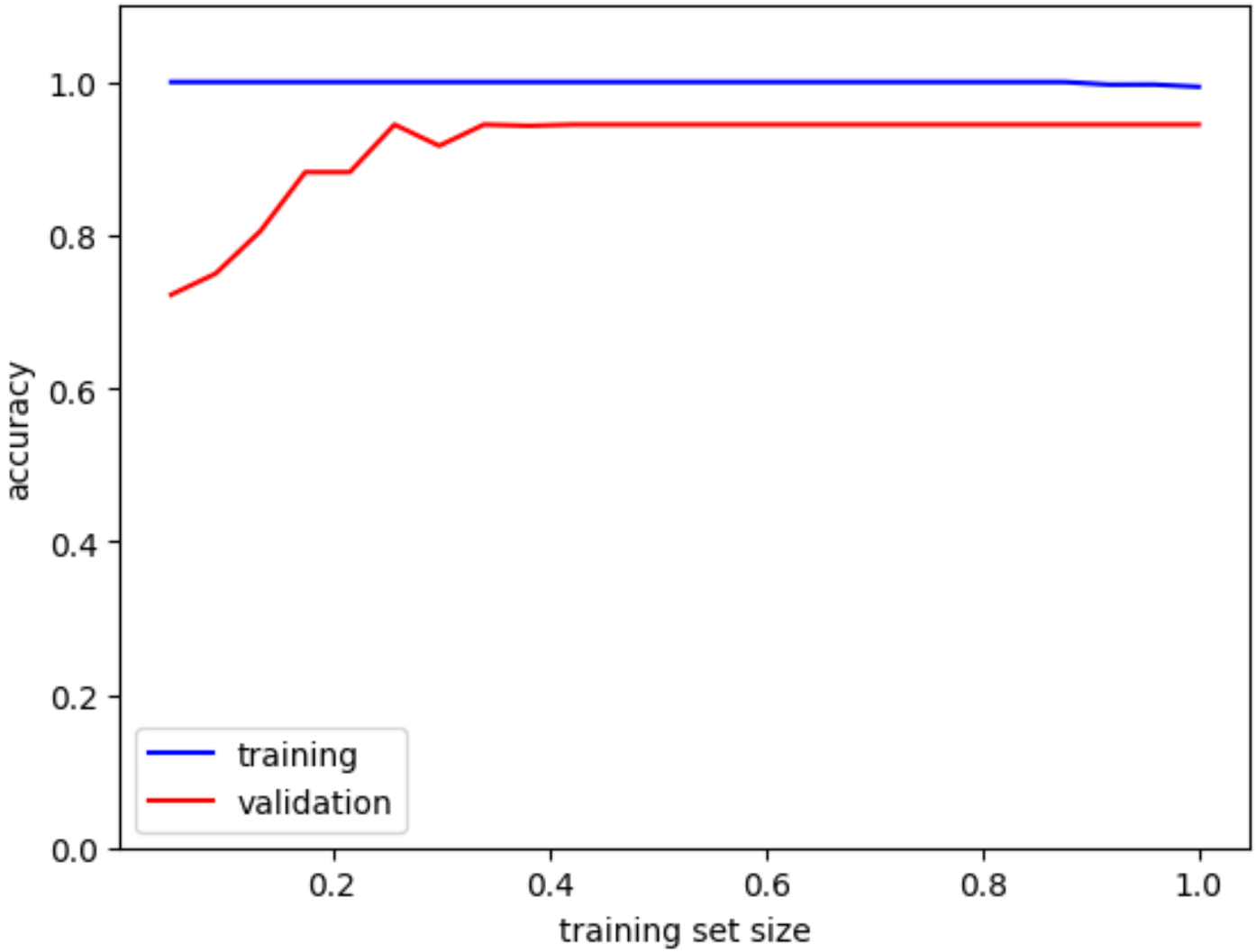
3. Validation
Learning curves

```
%matplotlib inline
import matplotlib.pyplot as plt


plt.plot(my_sizes, np.median(train_lc,1), color='blue', label='training')
plt.plot(my_sizes, np.median(val_lc,1), color='red', label='validation')
plt.legend(loc='best')
plt.ylim(0, 1.1)
plt.xlabel('training set size')
plt.ylabel('accuracy');
```

3. Validation

Learning curves



3. Validation
Grid search

- Exhaustive grid search cross-validation:

sklearn.model_selection.GridSearchCV(estimator, param_grid, scoring, cv)

- estimator is the estimator object, assumed to implement the scikit-learn estimator interface
- param_grid is a dict with parameters names as keys and lists of parameter settings to try as values, or a list of such dicts
- scoring is the strategy to evaluate the performance of the cross-validated model on the test set
- cv is the cross-validation splitting strategy

4. MODELS

4. Models

Naïve Bayes classifier

- Gaussian Naïve Bayes for continuous features:

sklearn.naive_bayes.GaussianNB(priors, var_smoothing)

- priors is an array-like of shape (n_classes,) with the prior probabilities of the classes. If specified, the priors are not adjusted according to the data
- var_smoothing is the portion of the largest variance of all features that is added to variances for calculation stability

4. Models

Naïve Bayes classifier

- Multinomial Naïve Bayes for discrete features:

sklearn.naive_bayes.MultinomialNB(alpha, force_alpha, fit_prior, class_prior)

- alpha is an additive smoothing parameter (set alpha=0 and force_alpha=True, for no smoothing)
- force_alpha If False and alpha is less than 1e-10, it will set alpha to 1e-10. If True, alpha will remain unchanged
- fit_prior indicates whether to learn class prior probabilities or not
- class_prior are the prior probabilities of the classes. If specified, the priors are not adjusted according to the data

4. Models

Nearest neighbor classifier

sklearn.neighbors.KNeighborsClassifier(n_neighbors, p=2, metric)

- n_neighbors is the number of neighbors to use
- p is the power parameter for the Minkowski metric
- metric is the metric to use for distance computation

4. Models

Linear regression

- Ordinary least squares linear regression:

sklearn.linear_model.LinearRegression(fit_intercept)

- fit_intercept indicates whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e., data is expected to be centered)

4. Models

Support vector machines

- Support Vector Classification:

sklearn.svm.SVC(C, kernel, degree, gamma, coef0, probability)

- C is the regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared L2 penalty.
- kernel specifies the kernel type to be used in the algorithm
- degree is the degree of the polynomial kernel function
- gamma is the kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’ kernels
- coef0 is the independent term for ‘poly’ and ‘sigmoid’ kernel functions
- probability indicates whether to enable probability estimates

4. Models

Support vector machines

- Support Vector Regression:

sklearn.svm.SVR(kernel, degree, gamma, coef0, C)

- kernel specifies the kernel type to be used in the algorithm
- degree is the degree of the polynomial kernel function
- gamma is the kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’ kernels
- coef0 is the independent term for ‘poly’ and ‘sigmoid’ kernel functions
- C is the regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared L2 penalty.

4. Models

Decision tree classifier

- Decision tree classifier:

sklearn.tree.DecisionTreeClassifier(criterion, splitter, max_depth, min_samples_split, min_samples_leaf)

- criterion is the function to measure the quality of a split
- splitter is the strategy used to choose the split at each node
- max_depth is the maximum depth of the tree
- min_samples_split is the minimum number of samples required to split an internal node
- min_samples_leaf is the minimum number of samples required to be at a leaf node

4. Models

Random forest classifier

- Random forest classifier:

sklearn.ensemble.RandomForestClassifier(n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf)

- n_estimators is the number of trees in the forest
- criterion is the function to measure the quality of a split
- max_depth is the maximum depth of the tree
- min_samples_split is the minimum number of samples required to split an internal node
- min_samples_leaf is the minimum number of samples required to be at a leaf node

4. Models

k-means clustering

sklearn.cluster.KMeans(n_clusters, init, n_init, max_iter)

- n_clusters is the number of clusters to form as well as the number of centroids to generate
- init is the method for initialization
- n_init is the number of times the k-means algorithm is run with different centroid seeds
- max_iter is the maximum number of iterations

5. PERFORMANCE MEASUREMENT

5. Performance measures

Classification performance

- Performance measures can be imported from the sklearn.metrics module
- Accuracy: accuracy_score()
- Balanced accuracy: balanced_accuracy_score()
- Top k accuracy: top_k_accuracy_score()
- F1 measure: f1_score()
- Precision, recall: precision_score(), recall_score()
- Receiver Operating Characteristic (ROC): roc_auc_score()
- Plot ROC curve: RocCurveDisplay class
- Confusion matrix: confusion_matrix()
- Plot confusion matrix: ConfusionMatrixDisplay class

5. Performance measures

Regression performance

- Also from the sklearn.metrics module
- Mean squared error (MSE): mean_squared_error()
- Mean absolute error (MAE): mean_absolute_error()
- Median absolute error: median_absolute_error()
- Explained variance: explained_variance_score()
- R2 (coefficient of determination) score: r2_score()

5. Performance measures

Clustering performance

- From the sklearn.metrics module
- Mutual information: mutual_info_score()
- Normalized mutual information: normalized_mutual_info_score()
- Rand score: rand_score()
- Adjusted Rand score: adjusted_rand_score()

6. CONCLUSION

6. Conclusion

- The sklearn package provides reliable implementations for a wide range of standard machine learning models for classification, regression and clustering
- The features matrix and the label array must belong to standard classes of the numpy, pandas or scipy packages
- The interface of the model classes is standardized
- Utilities for cross validation, hyperparameter optimization, and performance measurement are provided