SCIKIT-LEARN 基础

金　林
中南财经政法大学统计系
jinlin82@qq.com

2020 年 2 月 23 日

大　纲

What is scikit-learn

1. Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language.
2. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN,
3. and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Facts

1. Initial release: June 2007;
2. Website: https://scikit-learn.org/stable/
3. History:
   1. This project was started in 2007 as a Google Summer of Code project by David Cournapeau.
   2. Later that year, Matthieu Brucher started work on this project as part of his thesis.
   3. The first public release, February the 1st 2010.
   4. Since then, several releases have appeared following a ~3 month cycle.

the inclusion criteria for new algorithms

1. only consider well-established algorithms for inclusion.
2. A rule of thumb is at least 3 years since publication, 200+ citations and wide use and usefulness.
3. A technique that provides a clear-cut improvement (e.g. an enhanced data structure or a more efficient approximation technique) on a widely-used method will also be considered for inclusion.
4. From the algorithms or techniques that meet the above criteria, only those which fit well within the current API of scikit-learn, that is a fit, predict/transform interface and ordinarily having input/output that is a numpy array or sparse matrix, are accepted.

Related Projects

1. https://scikit-learn.org/stable/related_projects.html

2 基本概念

- Machine learning
- Datasets
- Estimator and parameters
- Model selection

概念

1. Machine learning tackles Problems range from building a prediction function linking different observations, to classifying observations, or learning the structure in an unlabeled dataset.
2. In general, a learning problem considers a set of n samples of data and then tries to predict properties of unknown data.
3. If each sample is more than a single number and, for instance, a multi-dimensional entry (aka multivariate data), it is said to have several `attributes` or `features`.

statistical learning

- the use of machine learning techniques with the goal of statistical inference: drawing conclusions on the data at hand.

分类

supervised learning

- the data comes with additional attributes that we want to predict.

unsupervised learning

1. in which the training data consists of a set of input vectors x without any corresponding target values.
2. The goal in such problems may be to discover groups of similar examples within the data, where it is called clustering,
3. or to determine the distribution of data within the input space, known as density estimation,
4. or to project the data from a high-dimensional space down to two or three dimensions for the purpose of visualization.

supervised learning

classification

1. samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data.
2. Another way to think of classification is as a discrete (as opposed to continuous) form of supervised learning where one has a limited number of categories and for each of the n samples provided, one is to try to label them with the correct category or class.

regression

1. if the desired output consists of one or more continuous variables, then the task is called regression.

Training set and testing set

1. Machine learning is about learning some properties of a data set and then testing those properties against another data set.
2. While experimenting with any learning algorithm, it is important not to test the prediction of an estimator on the data used to fit the estimator as this would not be evaluating the performance of the estimator on new data. This is why datasets are often split into train and test data.
3. A common practice in machine learning is to evaluate an algorithm by splitting a data set into two.
4. We call one of those sets the training set, on which we learn some properties;
5. we call the other set the testing set, on which we test the learned properties.

2 基本概念

- Machine learning
- Datasets
- Estimator and parameters
- Model selection

Included datasets

1. scikit-learn comes with a few standard datasets, for instance the iris and digits datasets for classification and the boston house prices dataset for regression.
2. A dataset is a dictionary-like object that holds all the data and some metadata about the data.
3. This data is stored in the `.data` member, which is a `n_samples`, `n_features` array.

Included datasets

4. We say that the first axis of these arrays is the samples axis, while the second is the features axis.
5. In the case of supervised problem, one or more response variables are stored in the `.target` member.
6. The data is always a 2D array, shape (n_samples, n_features), although the original data may have had a different shape.
7. When the data is not initially in the (n_samples, n_features) shape, it needs to be preprocessed in order to be used by scikit-learn.

例子

```
from sklearn import datasets
iris = datasets.load_iris()
digits = datasets.load_digits()
print(digits.data)
digits.target
digits.images[0]

#Display the first digit
import matplotlib.pyplot as plt
plt.figure(1, figsize=(3, 3))
plt.imshow(digits.images[-1], cmap=plt.cm.gray_r, interpolation
    ='nearest')
plt.show()

digits.images.shape
data = digits.images.reshape((digits.images.shape[0], -1))
```

2 基本概念

- Machine learning
- Datasets
- Estimator and parameters
- Model selection

Estimator

1. the main API implemented by scikit-learn is that of the estimator.
2. estimators: An object which manages the estimation and decoding of a model.
3. An estimator is any object that learns from data; it may be a classification, regression or clustering algorithm or a transformer that extracts/filters useful features from raw data.
4. All estimator objects expose a `fit` method that takes a dataset (usually a 2-d array)
5. We can consider the estimator as a black box.

```
1 estimator.fit(data)
```

parameters

Estimator parameters:

- All the parameters of an estimator can be set when it is instantiated or by modifying the corresponding attribute:

```
estimator = Estimator(param1=1, param2=2)
estimator.param1
```

Estimated parameters:

- When data is fitted with an estimator, parameters are estimated from the data at hand. All the estimated parameters are attributes of the estimator object ending by an underscore:

```
estimator.estimated_param_
```

The problem solved in supervised learning

1. Supervised learning consists in learning the link between two datasets:
    1. the observed data X and an external variable y that we are trying to predict, usually called "target" or "labels".
    2. Most often, y is a 1D array of length n_samples.
2. All supervised estimators in scikit-learn implement a `fit(X, y)` method to fit the model and a `predict(X)` method that, given unlabeled observations X, returns the predicted labels y.
3. If the prediction task is to classify the observations in a set of finite labels, in other words to "name" the objects observed, the task is said to be a classification task. When doing classification in scikit-learn, y is a vector of integers or strings.
4. if the goal is to predict a continuous target variable, it is said to be a regression task.

支持向量机例子

```
from sklearn import datasets
digits = datasets.load_digits()

from sklearn import svm
clf = svm.SVC(gamma=0.001, C=100.)

clf.fit(digits.data[:-1], digits.target[:-1])
clf.predict(digits.data[-1:])
```

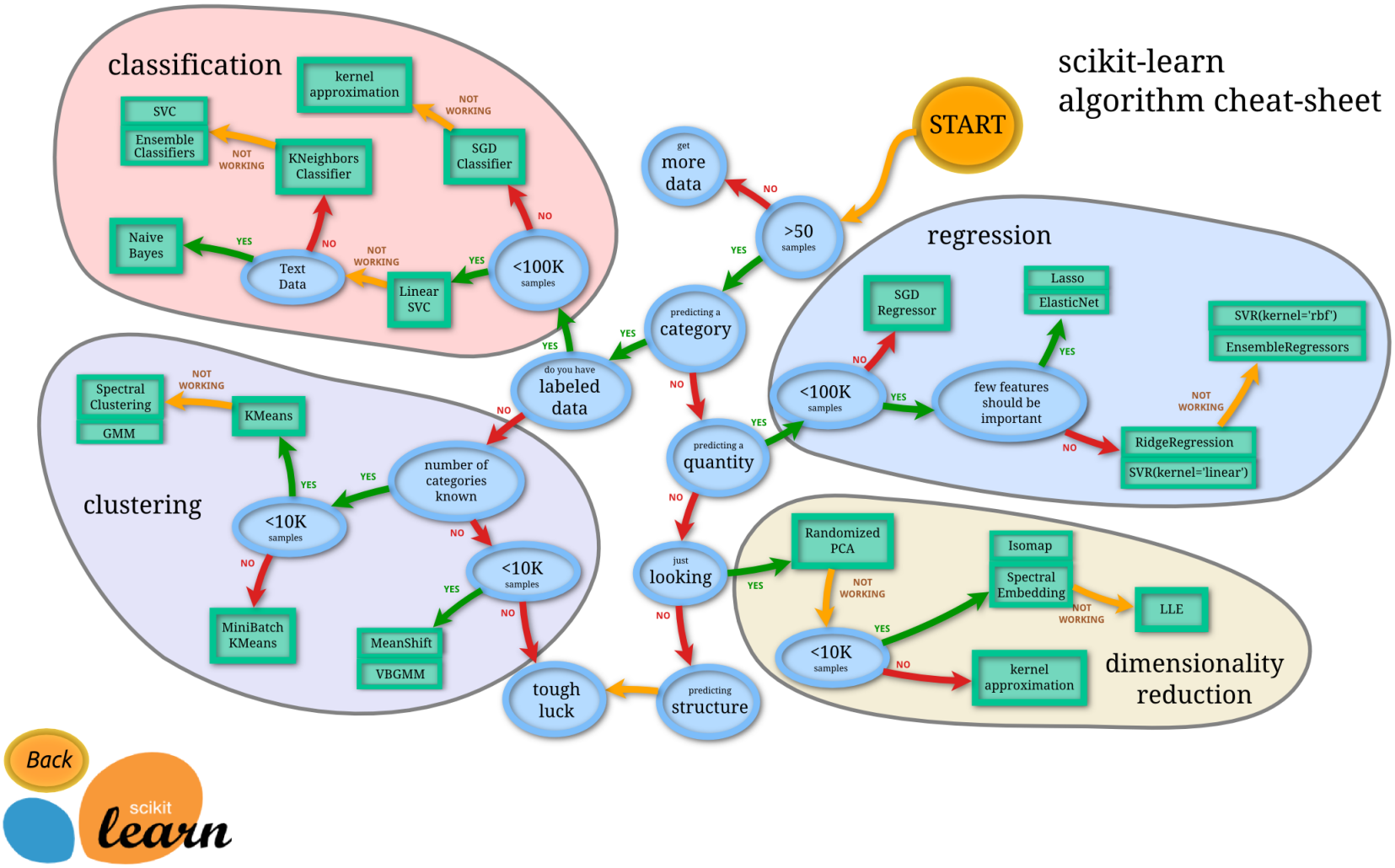
2 基本概念

- Machine learning
- Datasets
- Estimator and parameters
- Model selection

Choosing the right estimator

1. Often the hardest part of solving a machine learning problem can be finding the right estimator for the job.
2. Different estimators are better suited for different types of data and different problems.
3. https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

分类图


scikit-learn
algorithm cheat-sheet
START
get more data
NO
>50 samples
YES
predicting a category
YES
do you have labeled data
YES
<100K samples
NOT WORKING
Linear SVC
NOT WORKING
Text Data
YES
Naive Bayes
NO
KNeighbors Classifier
NOT WORKING
Ensemble Classifiers
NOT WORKING
SVC
NO
SGD Classifier
NOT WORKING
kernel approximation
classification
NO
predicting a quantity
YES
<100K samples
YES
few features should be important
NO
RidgeRegression
SVR(kernel='linear')
NOT WORKING
SVR(kernel='rbf')
EnsembleRegressors
YES
Lasso
ElasticNet
NO
SGD Regressor
regression
NO
just looking
YES
Randomized PCA
NOT WORKING
<10K samples
YES
Isomap
Spectral Embedding
NOT WORKING
LLE
NO
kernel approximation
dimensionality reduction
NO
predicting structure
NO
tough luck
NO
<10K samples
YES
MeanShift
VBGMM
NO
<10K samples
NO
MiniBatch KMeans
YES
number of categories known
YES
<10K samples
YES
KMeans
NOT WORKING
Spectral Clustering
GMM
clustering
Back
scikit learn

Score

1. every estimator exposes a `score` method that can judge the quality of the fit (or the prediction) on new data.
2. To get a better measure of prediction accuracy (which we can use as a proxy for goodness of fit of the model), we can successively split the data in folds that we use for training and testing.
3. Scikit-learn has a collection of classes which can be used to generate lists of train/test indices for popular cross-validation strategies.
4. They expose a `split` method which accepts the input dataset to be split and yields the train/test set indices for each iteration of the chosen cross-validation strategy.

Cross-validated scores

1. The cross-validation score can be directly calculated using the `cross_val_score` helper.
2. Given an estimator, the cross-validation object and the input dataset, the `cross_val_score` splits the data repeatedly into a training and a testing set, trains the estimator using the training set and computes the scores based on the testing set for each iteration of cross-validation.
3. By default the estimator' s score method is used to compute the individual scores.

Grid-search

1. scikit-learn provides an object that, given data, computes the score during the fit of an estimator on a parameter grid and chooses the parameters to maximize the cross-validation score.
2. By default, the GridSearchCV uses a 3-fold cross-validation. However, if it detects that a classifier is passed, rather than a regressor, it uses a stratified 3-fold. The default will change to a 5-fold cross-validation in version 0.22.

3 应用流程

- Fitting and predicting: estimator basics
- Transformers and pre-processors
- Pipelines: chaining pre-processors and estimators
- Model evaluation
- Automatic parameter searches

Fitting

1. Scikit-learn provides dozens of built-in machine learning algorithms and models, called estimators.
2. Each estimator can be fitted to some data using its `fit` method.

example: fit a `RandomForestClassifier` to data

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=0)
X = [[ 1,  2,  3],  # 2 samples, 3 features
     [11, 12, 13]]
y = [0, 1]  # classes of each sample
clf.fit(X, y)
```

Fitting

1. The `fit` method generally accepts 2 inputs.
2. The samples matrix (or design matrix) X. The size of X is typically (n_samples, n_features), which means that samples are represented as rows and features are represented as columns.
3. The target values y which are real numbers for regression tasks, or integers for classification (or any other discrete set of values). For unsupervized learning tasks, y does not need to be specified.
4. y is usually 1d array where the i th entry corresponds to the target of the i th sample (row) of X.
5. Both X and y are usually expected to be numpy arrays or equivalent array-like data types, though some estimators work with other formats such as sparse matrices.

3 应用流程

- Fitting and predicting: estimator basics
- Transformers and pre-processors
- Pipelines: chaining pre-processors and estimators
- Model evaluation
- Automatic parameter searches

Transformers and pre-processors

1. Machine learning workflows are often composed of different parts.
2. A typical pipeline consists of a pre-processing step that transforms or imputes the data, and a final predictor that predicts target values.
3. In scikit-learn, pre-processors and transformers follow the same API as the estimator objects (they actually all inherit from the same `BaseEstimator` class).
4. The transformer objects don' t have a `predict` method but rather a `transform` method that outputs a newly transformed sample matrix X.
5. Apply different transformations to different features: the `ColumnTransformer` is designed for these use-cases.

例子

```
from sklearn.preprocessing import StandardScaler
X = [[0, 15],
     [1, -10]]
StandardScaler().fit(X).transform(X)
```

3 应用流程

- Fitting and predicting: estimator basics
- Transformers and pre-processors
- Pipelines: chaining pre-processors and estimators
- Model evaluation
- Automatic parameter searches

Pipeline

1. Transformers and estimators (predictors) can be combined together into a single unifying object: a `Pipeline`.
2. The pipeline offers the same API as a regular estimator: it can be fitted and used for prediction with fit and predict.
3. As we will see later, using a pipeline will also prevent you from data leakage, i.e. disclosing some testing data in your training data.

例子

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# create a pipeline object
pipe = make_pipeline(StandardScaler(), LogisticRegression(
    random_state=0))

# load the iris dataset and split it into train and test sets
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    random_state=0)

# fit the whole pipeline
pipe.fit(X_train, y_train)
# we can now use it like any other estimator
accuracy_score(pipe.predict(X_test), y_test)
```

3 应用流程

- Fitting and predicting: estimator basics
- Transformers and pre-processors
- Pipelines: chaining pre-processors and estimators
- Model evaluation
- Automatic parameter searches

Model evaluation

1. Fitting a model to some data does not entail that it will predict well on unseen data.
2. This needs to be directly evaluated.
3. We have just seen the `train_test_split` helper that splits a dataset into train and test sets, but scikit-learn provides many other tools for model evaluation, in particular for cross-validation.

例子

- We here briefly show how to perform a 5-fold cross-validation procedure, using the `cross_validate` helper. Note that it is also possible to manually iterate over the folds, use different data splitting strategies, and use custom scoring functions.

```
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_validate

X, y = make_regression(n_samples=1000, random_state=0)
lr = LinearRegression()

result = cross_validate(lr, X, y)  # defaults to 5-fold CV
result['test_score']  # r_squared score is high because dataset is
    easy
```

3 应用流程

- Fitting and predicting: estimator basics
- Transformers and pre-processors
- Pipelines: chaining pre-processors and estimators
- Model evaluation
- Automatic parameter searches

parameter searches

1. All estimators have parameters (often called hyper-parameters in the literature) that can be tuned.
2. The generalization power of an estimator often critically depends on a few parameters.
3. For example a `RandomForestRegressor` has a n_estimators parameter that determines the number of trees in the forest, and a max_depth parameter that determines the maximum depth of each tree.
4. Quite often, it is not clear what the exact values of these parameters should be since they depend on the data at hand.
5. Scikit-learn provides tools to automatically find the best parameter combinations (via cross-validation).

例子

1. In the following example, we randomly search over the parameter space of a random forest with a `RandomizedSearchCV` object.
2. When the search is over, the `RandomizedSearchCV` behaves as a `RandomForestRegressor` that has been fitted with the best set of parameters.

例子：代码

```
from sklearn.datasets import fetch_california_housing
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from scipy.stats import randint
X, y = fetch_california_housing(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    random_state=0)
# define the parameter space that will be searched over
param_distributions = {'n_estimators': randint(1, 5),
^^I^^I        'max_depth': randint(5, 10)}
# now create a searchCV object and fit it to the data
search = RandomizedSearchCV(estimator=RandomForestRegressor(
    random_state=0), n_iter=5,
^^I^^I^^I    param_distributions=param_distributions, random_
    state=0)
search.fit(X_train, y_train)
search.best_params_
# the search object now acts like a normal random forest estimator
# with max_depth=9 and n_estimators=4
search.score(X_test, y_test)
```

scikit-learn 有监督学习算法

1. Linear Models
2. Linear and Quadratic Discriminant Analysis
3. Kernel ridge regression
4. Support Vector Machines
5. Stochastic Gradient Descent
6. Nearest Neighbors
7. Gaussian Processes
8. Cross decomposition
9. Naive Bayes

scikit-learn 有监督学习算法

10. Decision Trees
11. Ensemble methods
12. Multiclass and multilabel algorithms
13. Feature selection
14. Semi-Supervised
15. Isotonic regression
16. Probability calibration
17. Neural network models (supervised)

4 有监督学习

- 种类
- 例子：线型回归

sklearn.linear_model.LinearRegression

1. LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.
2. From the implementation point of view, this is just plain Ordinary Least Squares (`scipy.linalg.lstsq`) wrapped as a predictor object.

Methods

`fit(self, X, y)`	Fit linear model.
`get_params(self[, deep])`	Get parameters for this estimator.
`predict(self, X)`	Predict using the linear model.
`score(self, X, y)`	Return the coefficient of determination
`set_params(self, **params)`	Set the parameters of this estimator.

例子 1

```
import numpy as np
from sklearn.linear_model import LinearRegression
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
# y = 1 * x_0 + 2 * x_1 + 3
y = np.dot(X, np.array([1, 2])) + 3
reg = LinearRegression().fit(X, y)

reg.score(X, y)
reg.coef_
reg.intercept_
reg.predict(np.array([[3, 5]]))
```

例子 2

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True
    )
# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]
```

例子 2

```
# Create linear regression object
regr = linear_model.LinearRegression()
# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(diabetes_y_test, diabetes_y_pred))
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue',
    linewidth=3)
plt.xticks(());plt.yticks(())
plt.show()
```

5 无监督学习

- 种类
- 例子：K 均值聚类

scikit-learn 中无监督学习算法

1. Gaussian mixture models
2. Manifold learning
3. Clustering
4. Biclustering
5. Decomposing signals in components (matrix factorization problems)
6. Covariance estimation
7. Novelty and Outlier Detection
8. Density Estimation
9. Neural network models (unsupervised)

5 无监督学习

- 种类
- 例子：K 均值聚类

sklearn.cluster.KMeans

1. The `KMeans` algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares.
2. This algorithm requires the number of clusters to be specified.
3. It scales well to large number of samples and has been used across a large range of application areas in many different fields.

Methods

fit(self, X[, y])	Compute k-means clustering.
fit_predict(self, X[, y])	Compute cluster centers and predict cluster in
fit_transform(self, X[, y])	Compute clustering and transform X to cluste
get_params(self[, deep])	Get parameters for this estimator.
predict(self, X)	Predict the closest cluster each sample in X b
score(self, X[, y])	Opposite of the value of X on the K-means ob
set_params(self, **params)	Set the parameters of this estimator.
transform(self, X)	Transform X to a cluster-distance space.

例子 1

```
from sklearn.cluster import KMeans
import numpy as np
X = np.array([[1, 2], [1, 4], [1, 0],
^^I       [10, 2], [10, 4], [10, 0]])
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
kmeans.labels_

kmeans.predict([[0, 0], [12, 3]])

kmeans.cluster_centers_
```

例子 2

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  # Though the following import is not directly being used, it is
       required
4  # for 3D projection to work
5  from mpl_toolkits.mplot3d import Axes3D
6
7  from sklearn.cluster import KMeans
8  from sklearn import datasets
9
10 np.random.seed(5)
11
12 iris = datasets.load_iris()
13 X = iris.data
14 y = iris.target
15
16 estimators = [('k_means_iris_8', KMeans(n_clusters=8)),
17 ^^I       ('k_means_iris_3', KMeans(n_clusters=3)),
18 ^^I       ('k_means_iris_bad_init', KMeans(n_clusters=3,
19 ^^I^^I^^I^^I^^I         n_init=1, init='random'))]
```

例子 2

```
fignum = 1
titles = ['8 clusters', '3 clusters', '3 clusters, bad
    initialization']
for name, est in estimators:
    fig = plt.figure(fignum, figsize=(4, 3))
    ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
    est.fit(X)
    labels = est.labels_

    ax.scatter(X[:, 3], X[:, 0], X[:, 2],
^^I       c=labels.astype(np.float), edgecolor='k')

    ax.w_xaxis.set_ticklabels([])
    ax.w_yaxis.set_ticklabels([])
    ax.w_zaxis.set_ticklabels([])
    ax.set_xlabel('Petal width')
    ax.set_ylabel('Sepal length')
    ax.set_zlabel('Petal length')
    ax.set_title(titles[fignum - 1])
    ax.dist = 12
    fignum = fignum + 1
```

例子 2

```
# Plot the ground truth
fig = plt.figure(fignum, figsize=(4, 3))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
for name, label in [('Setosa', 0),
^^I^^I    ('Versicolour', 1),
^^I^^I    ('Virginica', 2)]:
    ax.text3D(X[y == label, 3].mean(),
^^I       X[y == label, 0].mean(),
^^I       X[y == label, 2].mean() + 2, name,
^^I       horizontalalignment='center',
^^I       bbox=dict(alpha=.2, edgecolor='w', facecolor='w'))
# Reorder the labels to have colors matching the cluster results
y = np.choose(y, [1, 2, 0]).astype(np.float)
ax.scatter(X[:, 3], X[:, 0], X[:, 2], c=y, edgecolor='k')

ax.w_xaxis.set_ticklabels([]);ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])
ax.set_xlabel('Petal width');ax.set_ylabel('Sepal length')
ax.set_zlabel('Petal length');ax.set_title('Ground Truth')
ax.dist = 12
fig.show()
```

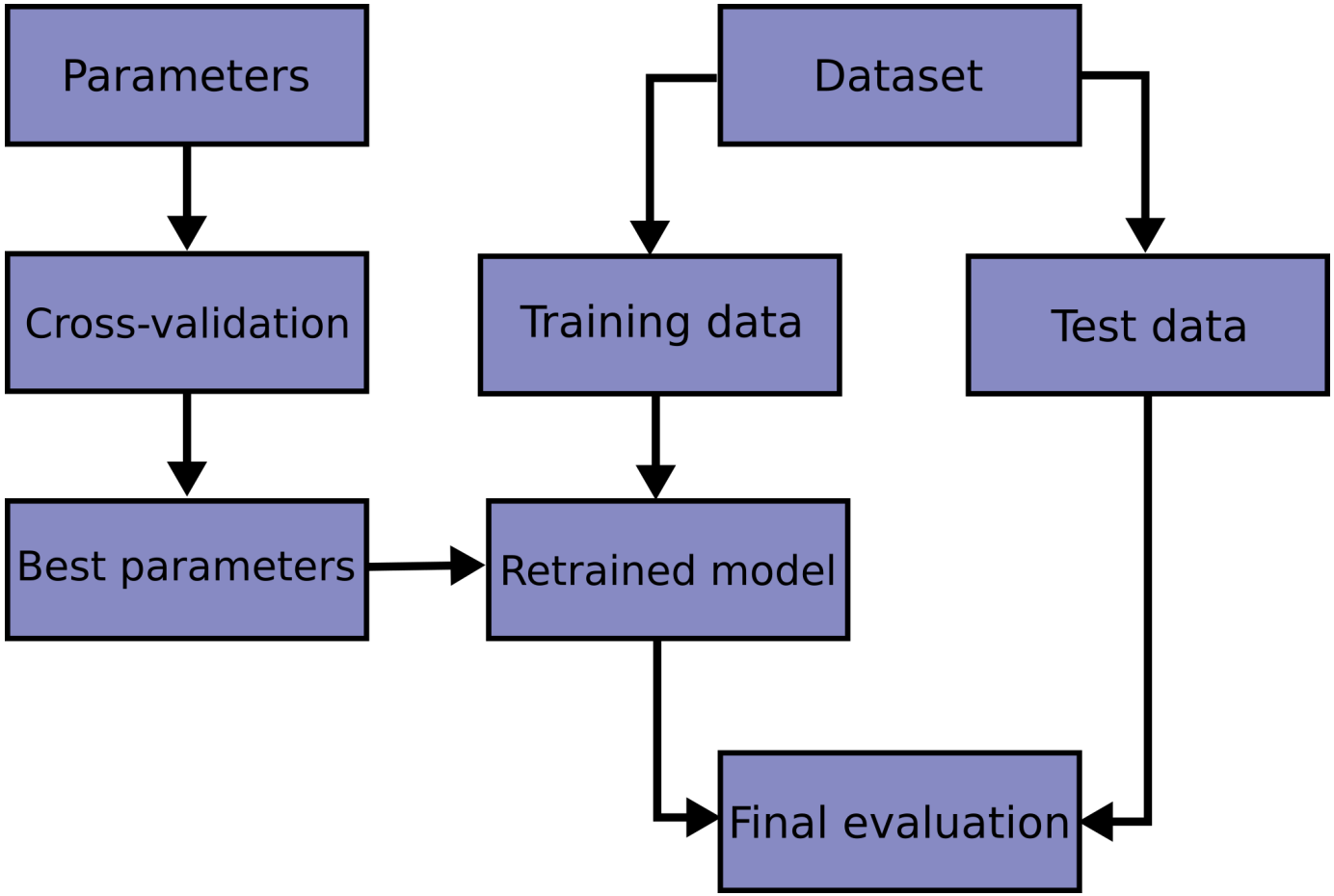
6 模型选择和评价

- 训练集，验证集和测试集
- 交叉验证
- 交叉验证循环方式
- Tuning the hyper-parameters of an estimator
- Metrics and scoring: quantifying the quality of predictions
- Model persistence
- Validation curves: plotting scores to evaluate models

背景

1. Learning the parameters of a prediction function and testing it on the same data is a methodological mistake:
2. a model that 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.
3. 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 `X_test, y_test`.
4. In scikit-learn a random split into training and test sets can be quickly computed with the `train_test_split` helper function.

机器学习流程图


Parameters
Dataset
Cross-validation
Training data
Test data
Best parameters
Retrained model
Final evaluation

为什么需要验证集

1. When evaluating different settings ("hyperparameters") for estimators, such as the C setting that must be manually set for an SVM, there is still a risk of overfitting on the test set because the parameters can be tweaked until the estimator performs optimally.
2. This way, knowledge about the test set can "leak" into the model and evaluation metrics no longer report on generalization performance.
3. To solve this problem, yet another part of the dataset can be held out as a so-called "validation set "

训练集，验证集和测试集定义

Training Dataset:

- The sample of data used to fit the model.

Validation Dataset:

- The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.

Test Dataset:

- The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

训练集

1. The training set is used to fit the models;use the training set to find the "optimal" weights. used to fit the parameters (e.g. weights of connections between neurons in artificial neural networks) of the model.
2. The model (e.g. a neural net or a naive Bayes classifier) is trained on the training dataset using a supervised learning method (e.g. gradient descent or stochastic gradient descent).
3. The current model is run with the training dataset and produces a result, which is then compared with the target, for each input vector in the training dataset.
4. Based on the result of the comparison and the specific learning algorithm being used, the parameters of the model are adjusted.
5. The model fitting can include both variable selection and parameter estimation.

验证集

1. validation dataset provides an unbiased evaluation of a model fit on the training dataset while tuning the model's hyperparameters(e.g. the number of hidden units in a neural network).
2. the validation set is used to estimate prediction error for model selection;
3. Validation datasets can be used for regularization by early stopping: stop training when the error on the validation dataset increases, as this is a sign of overfitting to the training dataset.
4. The validation dataset functions as a hybrid: it is training data used by testing, but neither as part of the low-level training nor as part of the final testing.
5. The validation dataset may also play a role in other forms of model preparation, such as feature selection.

测试集

1. used to provide an unbiased evaluation of a final model fit on the training dataset. If the data in the test dataset has never been used in training (for example in cross-validation), the test dataset is also called a holdout dataset.
2. the test set is used for assessment of the generalization error of the final chosen model.
3. Ideally, the test set should be kept in a "vault," and be brought out only at the end of the data analysis.
4. a test set, by the standard definition in the NN [neural net] literature, is never used to choose among two or more networks, so that the error on the test set provides an unbiased estimate of the generalization error.

说明

1. The final model could be fit on the aggregate of the training and validation datasets.
2. the "validation dataset" is predominately used to describe the evaluation of models when tuning hyperparameters and data preparation, and the "test dataset" is predominately used to describe the evaluation of a final tuned model when comparing it to other final models.
3. That the notions of "validation dataset" and "test dataset" may disappear when adopting alternate resampling methods like k-fold cross validation.
4. 参考：
   - https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets
   - https://machinelearningmastery.com/difference-test-validation-datasets/

伪代码

```
# split data
data = ...
train, validation, test = split(data)

# tune model hyperparameters
parameters = ...
for params in parameters:
    model = fit(train, params)
    skill = evaluate(model, validation)

# evaluate final model for comparison with other models
model = fit(train)
skill = evaluate(model, test)
```

例子

```
### 注意：该例子没有验证集
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn import svm

X, y = datasets.load_iris(return_X_y=True)
X.shape, y.shape

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=0)

X_train.shape, y_train.shape

X_test.shape, y_test.shape

clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
clf.score(X_test, y_test)
```

6 模型选择和评价

- 训练集，验证集和测试集
- 交叉验证
- 交叉验证循环方式
- Tuning the hyper-parameters of an estimator
- Metrics and scoring: quantifying the quality of predictions
- Model persistence
- Validation curves: plotting scores to evaluate models

为什么需要交叉验证

1. partitioning the available data into three sets, we drastically reduce the number of samples which can be used for learning the model,
2. and the results can depend on a particular random choice for the pair of (train, validation) sets.
3. A validation set is a single evaluation of the model and has limited ability to characterize the uncertainty in the results.

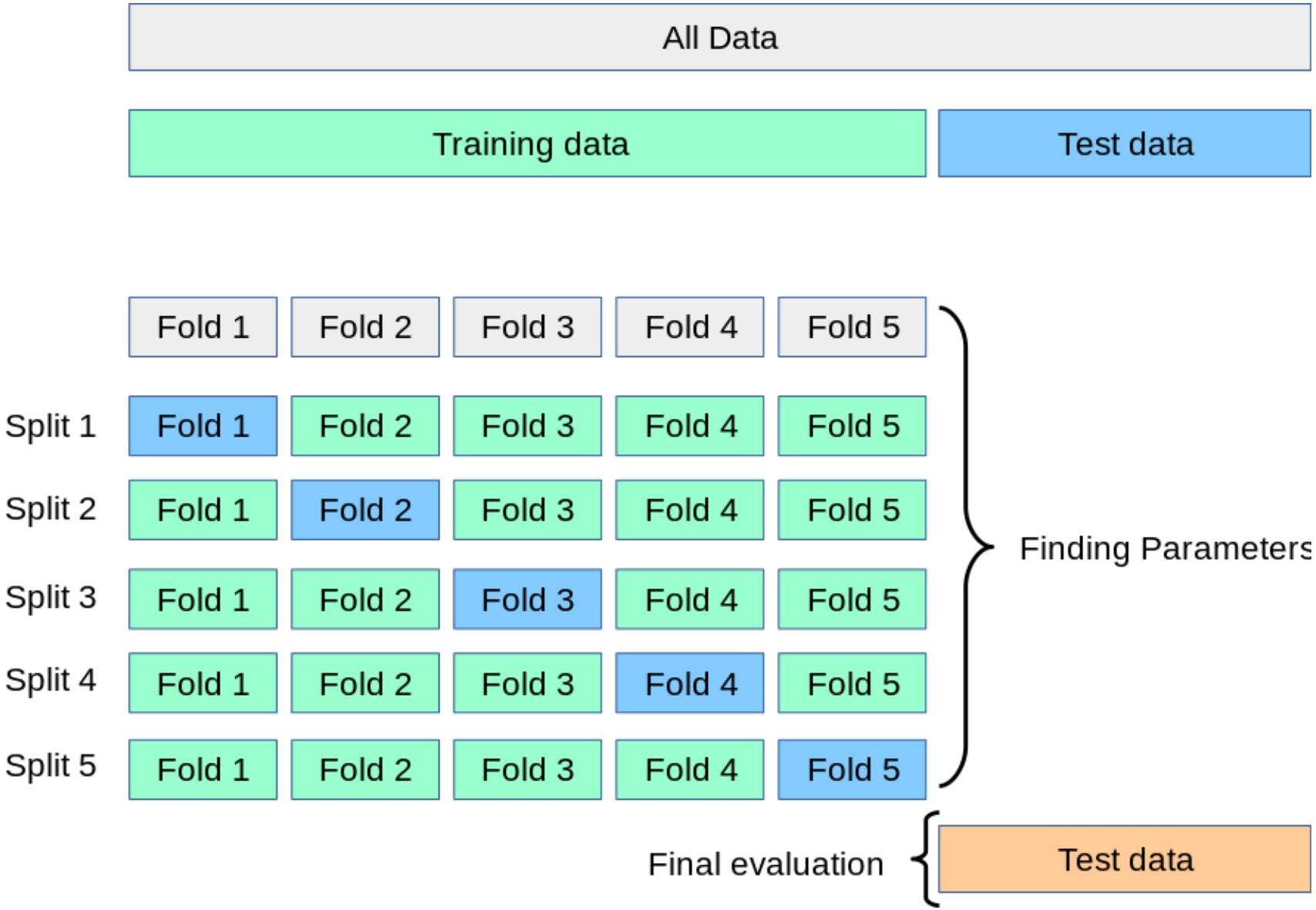
交叉验证 (ross validation)

1. A solution to this problem is a procedure called cross-validation (CV for short). A test set should still be held out for final evaluation, but the validation set is no longer needed when doing CV.
2. It is more than likely that you will not see references to training, validation, and test datasets in modern applied machine learning.
3. Reference to a "validation dataset" disappears if the practitioner is choosing to tune model hyperparameters using k-fold cross-validation with the training dataset.
4. Reference to the "test dataset" too may disappear if the cross-validation of model hyperparameters using the training dataset is nested within a broader cross-validation of the model.
5. Cross validation iterators can be used to directly perform model selection using Grid Search for the optimal hyperparameters of the model.

k-fold 交叉验证

1. the training set is split into k smaller sets.
2. The following procedure is followed for each of the k "folds" :
   1. A model is trained using $k-1$ of the folds as training data;
   2. 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).
3. The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop.
4. This approach can be computationally expensive, but does not waste too much data (as is the case when fixing an arbitrary validation set), which is a major advantage in problems where the number of samples is very small.

k-fold 交叉验证示意图



交叉验证伪代码

```
# split data
data = ...
train, test = split(data)

# tune model hyperparameters
parameters = ...
k = ...
for params in parameters:
    skills = list()
    for i in k:
        fold_train, fold_val = cv_split(i, k, train)
        model = fit(fold_train, params)
        skill_estimate = evaluate(model, fold_val)
        skills.append(skill_estimate)
    skill = summarize(skills)

# evaluate final model for comparison with other models
model = fit(train)
skill = evaluate(model, test)
```

Computing cross-validated metrics

1. The simplest way to use cross-validation is to call the cross_val_score helper function on the estimator and the dataset.
2. The following example demonstrates how to estimate the accuracy of a linear kernel support vector machine on the iris dataset by splitting the data, fitting a model and computing the score 5 consecutive times.
3. When the `cv` argument is an integer, cross_val_score uses the KFold strategies by default.
4. By default, the score computed at each CV iteration is the score method of the estimator. It is possible to change this by using the `scoring` parameter.

例子

```
from sklearn.model_selection import cross_val_score
clf = svm.SVC(kernel='linear', C=1)
scores = cross_val_score(clf, X, y, cv=5)
scores

from sklearn import metrics
scores = cross_val_score(
    clf, X, y, cv=5, scoring='f1_macro')
scores
```

6 模型选择和评价

- 训练集，验证集和测试集
- 交叉验证
- 交叉验证循环方式
- Tuning the hyper-parameters of an estimator
- Metrics and scoring: quantifying the quality of predictions
- Model persistence
- Validation curves: plotting scores to evaluate models

不同数据类型交叉验证

1. Cross-validation iterators for i.i.d. data
2. Cross-validation iterators with stratification based on class labels.
3. Cross-validation iterators for grouped data
4. Cross validation of time series data

5 种交叉验证方式

1. K-fold: `sklearn.model_selection.KFold`
2. Repeated K-Fold: `sklearn.model_selection.RepeatedKFold`
3. Leave One Out (LOO): `sklearn.model_selection.LeaveOneOut`
4. Leave P Out (LPO): `sklearn.model_selectionLeavePOut.`
5. Random permutations cross-validation a.k.a. Shuffle & Split: `sklearn.model_selection.ShuffleSplit`

例子

```python
import numpy as np

## K-fold
from sklearn.model_selection import KFold

X = ["a", "b", "c", "d"]
kf = KFold(n_splits=2)
for train, test in kf.split(X):
    print("%s %s" % (train, test))

# Repeated K-Fold
from sklearn.model_selection import RepeatedKFold
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
random_state = 12883823
rkf = RepeatedKFold(n_splits=2, n_repeats=2, random_state=
    random_state)
for train, test in rkf.split(X):
    print("%s %s" % (train, test))
```

例子

```
from sklearn.model_selection import LeaveOneOut
X = [1, 2, 3, 4]
loo = LeaveOneOut()
for train, test in loo.split(X):
    print("%s %s" % (train, test))

from sklearn.model_selection import LeavePOut
X = np.ones(4)
lpo = LeavePOut(p=2)
for train, test in lpo.split(X):
    print("%s %s" % (train, test))

from sklearn.model_selection import ShuffleSplit
X = np.arange(10)
ss = ShuffleSplit(n_splits=5, test_size=0.25, random_state=0)
for train_index, test_index in ss.split(X):
    print("%s %s" % (train_index, test_index))
```

6 模型选择和评价

- 训练集，验证集和测试集
- 交叉验证
- 交叉验证循环方式
- Tuning the hyper-parameters of an estimator
- Metrics and scoring: quantifying the quality of predictions
- Model persistence
- Validation curves: plotting scores to evaluate models

hyper-parameters

1. Hyper-parameters are parameters that are not directly learnt within estimators. In scikit-learn they are passed as arguments to the constructor of the estimator classes.
2. Typical examples include `C`, `kernel` and `gamma` for Support Vector Classifier, `alpha` for Lasso, etc.
3. It is possible and recommended to search the hyper-parameter space for the best cross validation score.
4. Any parameter provided when constructing an estimator may be optimized in this manner. Specifically, to find the names and current values for all parameters for a given estimator, use:

```
1 estimator.get_params()
```

How to search

1. A search consists of:
   1. an estimator (regressor or classifier such as `sklearn.svm.SVC()`);
   2. a parameter space;
   3. a method for searching or sampling candidates;
   4. a cross-validation scheme; and
   5. a score function.

Two generic approaches

1. Two generic approaches to sampling search candidates are provided in scikit-learn:
   1. for given values, `GridSearchCV` exhaustively considers all parameter combinations,
   2. while `RandomizedSearchCV` can sample a given number of candidates from a parameter space with a specified distribution.
2. Note that it is common that a small subset of those parameters can have a large impact on the predictive or computation performance of the model while others can be left to their default values.
3. It is recommended to read the docstring of the estimator class to get a finer understanding of their expected behavior

Exhaustive Grid Search

1. The grid search provided by `GridSearchCV` exhaustively generates candidates from a grid of parameter values specified with the param_grid parameter.
2. The `GridSearchCV` instance implements the usual estimator API: when "fitting" it on a dataset all the possible combinations of parameter values are evaluated and the best combination is retained.

```
param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel':
     ['rbf']},
 ]
```

Randomized Parameter Optimization

1. While using a grid of parameter settings is currently the most widely used method for parameter optimization, other search methods have more favourable properties.
2. `RandomizedSearchCV` implements a randomized search over parameters, where each setting is sampled from a distribution over possible parameter values.
3. This has two main benefits over an exhaustive search:
   1. A budget can be chosen independent of the number of parameters and possible values.
   2. Adding parameters that do not influence the performance does not decrease efficiency.

Randomized Parameter Optimization

1. Specifying how parameters should be sampled is done using a dictionary, very similar to specifying parameters for `GridSearchCV` .
2. Additionally, a computation budget, being the number of sampled candidates or sampling iterations, is specified using the n_iter parameter.
3. For each parameter, either a distribution over possible values or a list of discrete choices (which will be sampled uniformly) can be specified.

```
{'C': scipy.stats.expon(scale=100), 'gamma': scipy.stats.expon(
    scale=.1),
  'kernel': ['rbf'], 'class_weight':['balanced', None]}
```

Randomized Parameter Optimization

1. In principle, any function can be passed that provides a `rvs` (random variate sample) method to sample a value. A call to the rvs function should provide independent random samples from possible parameter values on consecutive calls.
2. For continuous parameters, such as `C` above, it is important to specify a continuous distribution to take full advantage of the randomization. This way, increasing n_iter will always lead to a finer search.

Specifying an objective metric

1. By default, parameter search uses the `score` function of the estimator to evaluate a parameter setting.
2. These are the sklearn.metrics.accuracy_score for classification and sklearn.metrics.r2_score for regression.
3. For some applications, other scoring functions are better suited (for example in unbalanced classification, the accuracy score is often uninformative).
4. An alternative `scoring` function can be specified via the scoring parameter to `GridSearchCV`, `RandomizedSearchCV` and many of the specialized cross-validation tools described below.

6 模型选择和评价

- 训练集，验证集和测试集
- 交叉验证
- 交叉验证循环方式
- Tuning the hyper-parameters of an estimator
- Metrics and scoring: quantifying the quality of predictions
- Model persistence
- Validation curves: plotting scores to evaluate models

3 different APIs

1. There are 3 different APIs for evaluating the quality of a model's predictions:
2. `Dummy` estimators are useful to get a baseline value of those metrics for random predictions.

Estimator score method:

- Estimators have a score method providing a default evaluation criterion for the problem they are designed to solve.
- This is not discussed on this page, but in each estimator's documentation.

3 different APIs

Scoring parameter:

- Model-evaluation tools using cross-validation (such as `model_selection.cross_val_score` and `model_selection.GridSearchCV`) rely on an internal scoring strategy.

Metric functions:

- The `metrics` module implements functions assessing prediction error for specific purposes.
- These metrics are detailed in sections on Classification metrics, Multilabel ranking metrics, Regression metrics and Clustering metrics.

The scoring parameter

1. Model selection and evaluation using tools, such as `model_selection.GridSearchCV` and `model_selection.cross_val_score`, take a `scoring` parameter that controls what metric they apply to the estimators evaluated.
2. For the most common use cases, you can designate a scorer object with the `scoring` parameter.
3. All scorer objects follow the convention that higher return values are better than lower return values.
4. Thus metrics which measure the distance between the model and the data, like metrics.mean_squared_error, are available as neg_mean_squared_error which return the negated value of the metric.

Classification metrics

1. Some of these are restricted to the binary classification case:
   1. precision_recall_curve(y_true, probas_pred): Compute precision-recall pairs for different probability thresholds
   2. roc_curve(y_true, y_score[, pos_label, ...]): Compute Receiver operating characteristic (ROC)
2. Others also work in the multiclass case:
   1. balanced_accuracy_score(y_true, y_pred[, ...]): Compute the balanced accuracy
   2. cohen_kappa_score(y1, y2[, labels, weights, ...]): Cohen' s kappa: a statistic that measures inter-annotator agreement.
   3. confusion_matrix(y_true, y_pred[, labels, ...]): Compute confusion matrix to evaluate the accuracy of a classification.
   4. hinge_loss(y_true, pred_decision[, labels, ...]): Average hinge loss (non-regularized)
   5. matthews_corrcoef(y_true, y_pred[, ...]): Compute the Matthews correlation coefficient (MCC)
   6. roc_auc_score(y_true, y_score[, average, ...]): Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.

Classification metrics

3. Some also work in the multilabel case:
   1. accuracy_score(y_true, y_pred[, normalize, ...]): Accuracy classification score.
   2. classification_report(y_true, y_pred[, ...]): Build a text report showing the main classification metrics
   3. f1_score(y_true, y_pred[, labels, ...]): Compute the F1 score, also known as balanced F-score or F-measure
   4. fbeta_score(y_true, y_pred, beta[, labels, ...]): Compute the F-beta score
   5. hamming_loss(y_true, y_pred[, labels, ...]): Compute the average Hamming loss.
   6. jaccard_score(y_true, y_pred[, labels, ...]): Jaccard similarity coefficient score
   7. log_loss(y_true, y_pred[, eps, normalize, ...]): Log loss, aka logistic loss or cross-entropy loss.
   8. multilabel_confusion_matrix(y_true, y_pred): Compute a confusion matrix for each class or sample

Classification metrics

9. precision_recall_fscore_support(y_true, y_pred): Compute precision, recall, F-measure and support for each class
10. precision_score(y_true, y_pred[, labels, ...]): Compute the precision
11. recall_score(y_true, y_pred[, labels, ...]): Compute the recall
12. roc_auc_score(y_true, y_score[, average, ...]): Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
13. zero_one_loss(y_true, y_pred[, normalize, ...]): Zero-one classification loss.
14. And some work with binary and multilabel (but not multiclass) problems:average_precision_score(y_true, y_score[, ...]): Compute average precision (AP) from prediction scores

Multilabel ranking metrics

1. Coverage error: The `coverage_error`
2. Label ranking average precision: The `label_ranking_average_precision_score`
3. Ranking loss: The `label_ranking_loss`
4. Normalized Discounted Cumulative Gain

Regression metrics

1. Explained variance score: The `explained_variance_score`
2. Max error: The `max_error`
3. Mean absolute error: The `mean_absolute_error`
4. Mean squared error: The `mean_squared_error`
5. Mean squared logarithmic error: The `mean_squared_log_error`
6. Median absolute error: The `median_absolute_error`
7. R^2 score, the coefficient of determination: The `r2_score`
8. Mean Poisson, Gamma, and Tweedie deviances: The `mean_tweedie_deviance`

Clustering metrics

1. Adjusted Rand index: `adjusted_rand_score`
2. Mutual Information based scores: `adjusted_mutual_info_score`
3. Homogeneity, completeness and V-measure: `homogeneity_score`, `completeness_score`, `v_measure_score`
4. Fowlkes-Mallows scores: `fowlkes_mallows_score`
5. Silhouette Coefficient: `silhouette_score`
6. Calinski-Harabasz Index: `calinski_harabasz_score`
7. Davies-Bouldin Index: `davies_bouldin_score`
8. Contingency Matrix: `sklearn.metrics.cluster.contingency_matrix`

Dummy estimators

1. When doing supervised learning, a simple sanity check consists of comparing one′ s estimator against simple rules of thumb.

`DummyClassifier` implements several such simple strategies for classification:

1. stratified generates random predictions by respecting the training set class distribution.
2. most_frequent always predicts the most frequent label in the training set.
3. prior always predicts the class that maximizes the class prior (like `most_frequent`) and `predict_proba` returns the class prior.
4. uniform generates predictions uniformly at random.
5. constant always predicts a constant label that is provided by the user.

Dummy estimators

1. Note that with all these strategies, the predict method completely ignores the input data!
2. More generally, when the accuracy of a classifier is too close to random, it probably means that something went wrong: features are not helpful, a hyperparameter is not correctly tuned, the classifier is suffering from class imbalance, etc.

例子

```
### create an imbalanced dataset
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
X, y = load_iris(return_X_y=True)
y[y != 1] = -1
X_train, X_test, y_train, y_test = train_test_split(X, y,
    random_state=0)

### compare the accuracy of SVC and most_frequent
from sklearn.dummy import DummyClassifier
from sklearn.svm import SVC
clf = SVC(kernel='linear', C=1).fit(X_train, y_train)
clf.score(X_test, y_test)
clf = DummyClassifier(strategy='most_frequent', random_state=0)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)

### change the kernel
clf = SVC(kernel='rbf', C=1).fit(X_train, y_train)
clf.score(X_test, y_test)
```

Dummy estimators

`DummyRegressor` also implements four simple rules of thumb for regression:

1. mean always predicts the mean of the training targets.
2. median always predicts the median of the training targets.
3. quantile always predicts a user provided quantile of the training targets.
4. constant always predicts a constant value that is provided by the user.
5. In all these strategies, the predict method completely ignores the input data.

6 模型选择和评价

- 训练集，验证集和测试集
- 交叉验证
- 交叉验证循环方式
- Tuning the hyper-parameters of an estimator
- Metrics and scoring: quantifying the quality of predictions
- Model persistence
- Validation curves: plotting scores to evaluate models

Model persistence

1. After training a scikit-learn model, it is desirable to have a way to persist the model for future use without having to retrain.
2. It is possible to save a model in scikit-learn by using Python' s built-in persistence model, namely `pickle` .
3. In the specific case of scikit-learn, it may be better to use `joblib` ' s replacement of pickle (`dump` & `load`), which is more efficient on objects that carry large numpy arrays internally as is often the case for fitted scikit-learn estimators, but can only pickle to the disk and not to a string.

例子

```
from sklearn import svm
from sklearn import datasets
clf = svm.SVC()
X, y= datasets.load_iris(return_X_y=True)
clf.fit(X, y)

import pickle
s = pickle.dumps(clf)
clf2 = pickle.loads(s)
clf2.predict(X[0:1])

y[0]

from joblib import dump, load
dump(clf, 'filename.joblib')
clf = load('filename.joblib')
```

6 模型选择和评价

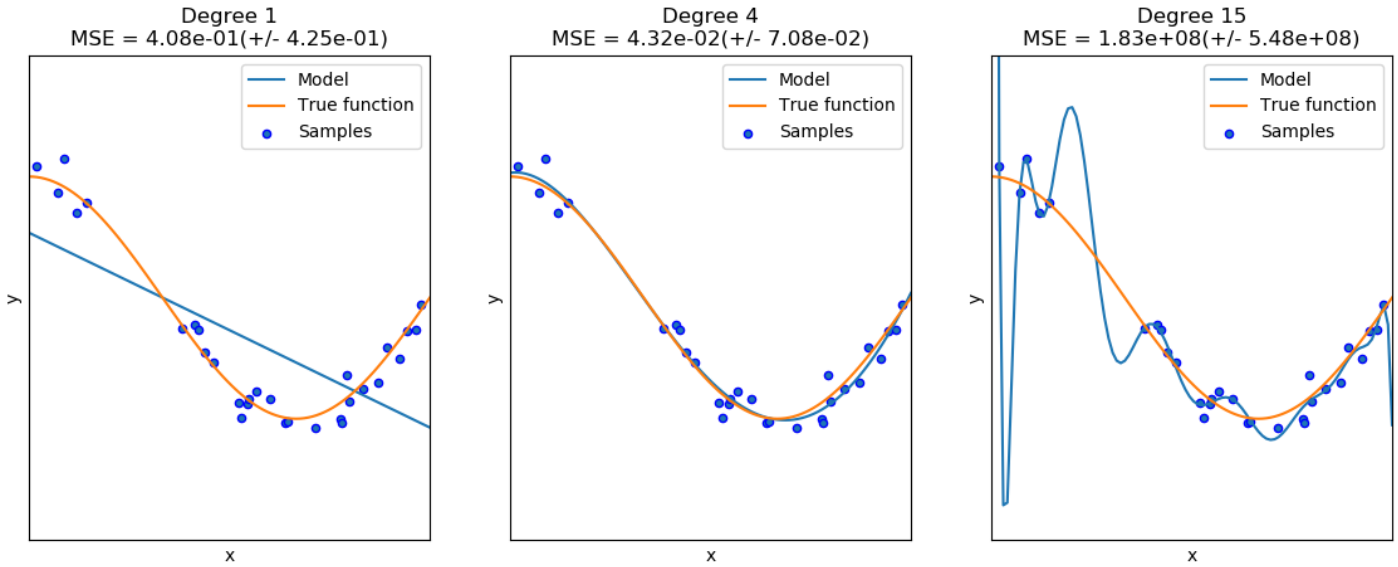
- 训练集，验证集和测试集
- 交叉验证
- 交叉验证循环方式
- Tuning the hyper-parameters of an estimator
- Metrics and scoring: quantifying the quality of predictions
- Model persistence
- Validation curves: plotting scores to evaluate models

简介

1. Every estimator has its advantages and drawbacks.
2. Its generalization error can be decomposed in terms of `bias`, `variance` and `noise`.
3. The `bias` of an estimator is its average error for different training sets.
4. The `variance` of an estimator indicates how sensitive it is to varying training sets.
5. `Noise` is a property of the data.
6. Bias and variance are inherent properties of estimators and we usually have to select learning algorithms and hyperparameters so that both bias and variance are as low as possible (Bias-variance dilemma).
7. Another way to reduce the variance of a model is to use more training data.

简介例子

1. In the following plot, we see a function $f(x) = \cos(\frac{3}{2}\pi x)$ and some noisy samples from that function.
2. We use three different estimators to fit the function: linear regression with polynomial features of degree 1, 4 and 15.



简介例子

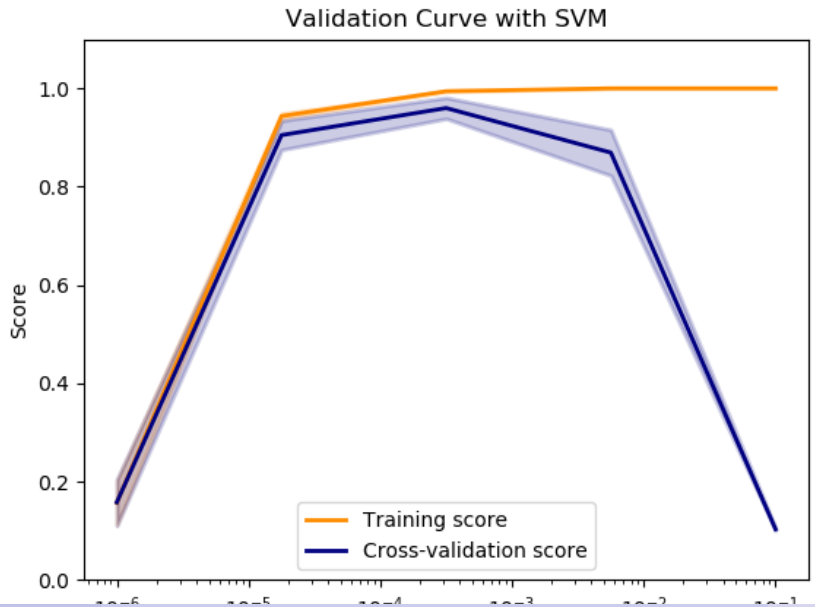
1. We see that the first estimator can at best provide only a poor fit to the samples and the true function because it is too simple (high bias, underfiting),
2. the second estimator approximates it almost perfectly
3. and the last estimator approximates the training data perfectly but does not fit the true function very well, i.e. it is very sensitive to varying training data (high variance, overfitting).
4. In the simple one-dimensional problem that we have seen in the example it is easy to see whether the estimator suffers from bias or variance.
5. However, in high-dimensional spaces, models can become very difficult to visualize.

Validation curve

1. To validate a model we need a scoring function, for example accuracy for classifiers.
2. The proper way of choosing multiple hyperparameters of an estimator are grid search or similar methods that select the hyperparameter with the maximum score on a validation set or multiple validation sets.
3. Note that if we optimized the hyperparameters based on a validation score the validation score is biased and not a good estimate of the generalization any longer.
4. To get a proper estimate of the generalization we have to compute the score on another test set.
5. However, it is sometimes helpful to plot the influence of a single hyperparameter on the training score and the validation score to find out whether the estimator is overfitting or underfitting for some hyperparameter values.
6. The function validation_curve can help in this case.

Validation curve

1. If the training score and the validation score are both low, the estimator will be underfitting.
2. If the training score is high and the validation score is low, the estimator is overfitting
3. and otherwise it is working very well.
4. A low training score and a high validation score is usually not possible.
5. All three cases can be found in the plot below where we vary the parameter γ of an SVM on the digits dataset.



例子

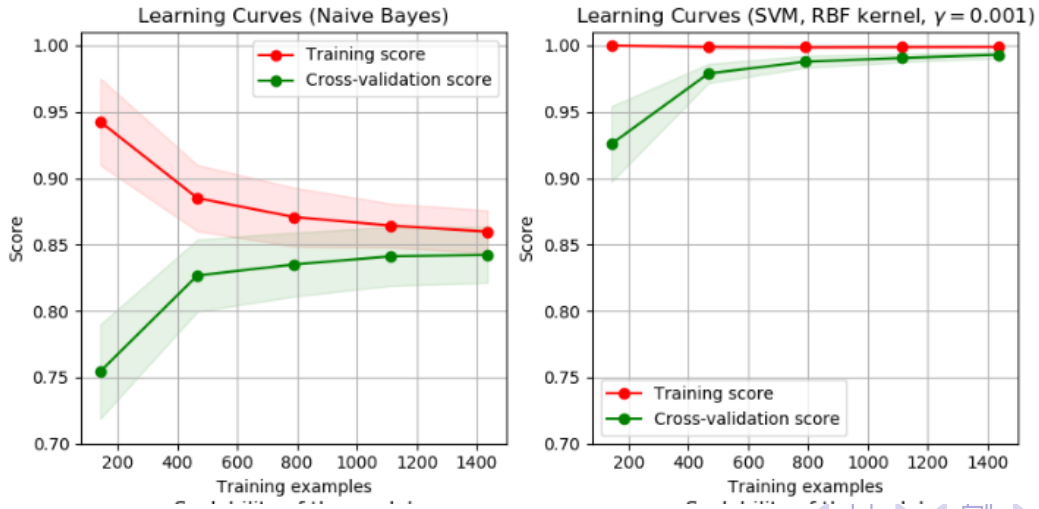
```
import numpy as np
from sklearn.model_selection import validation_curve
from sklearn.datasets import load_iris
from sklearn.linear_model import Ridge

np.random.seed(0)
X, y = load_iris(return_X_y=True)
indices = np.arange(y.shape[0])
np.random.shuffle(indices)
X, y = X[indices], y[indices]

train_scores, valid_scores = validation_curve(Ridge(), X, y, "
    alpha",
^^I^^I^^I^^I^^I      np.logspace(-7, 3, 3),
^^I^^I^^I^^I^^I      cv=5)
train_scores
valid_scores
```

Learning curve

1. A learning curve shows the validation and training score of an estimator for varying numbers of training samples.
2. It is a tool to find out how much we benefit from adding more training data and whether the estimator suffers more from a variance error or a bias error.
3. We can use the function learning_curve to generate the values that are required to plot such a learning curve (number of samples that have been used, the average scores on the training sets and the average scores on the validation sets).

Learning curve

1. Consider the following example where we plot the learning curve of a naive Bayes classifier and an SVM.
2. For the naive Bayes, both the validation score and the training score converge to a value that is quite low with increasing size of the training set. Thus, we will probably not benefit much from more training data.
3. In contrast, for small amounts of data, the training score of the SVM is much greater than the validation score. Adding more training samples will most likely increase generalization.



例子

```
from sklearn.model_selection import learning_curve
from sklearn.svm import SVC

train_sizes, train_scores, valid_scores = learning_curve(
    SVC(kernel='linear'), X, y, train_sizes=[50, 80, 110], cv
    =5)

train_sizes
train_scores
valid_scores
```

7 审查和可视化

- Inspection
- Visualizations

简介

1. Predictive performance is often the main goal of developing machine learning models.
2. Yet summarising performance with an evaluation metric is often insufficient: it assumes that the evaluation metric and test dataset perfectly reflect the target domain, which is rarely true.
3. In certain domains, a model needs a certain level of interpretability before it can be deployed.
4. A model that is exhibiting performance issues needs to be debugged for one to understand the model' s underlying issue.
5. The `sklearn.inspection` module provides tools to help understand the predictions from a model and what affects them.
6. This can be used to evaluate assumptions and biases of a model, design a better model, or to diagnose issues with model performance.

Partial dependence plots

1. Partial dependence plots (PDP) show the dependence between the target response and a set of 'target' features, marginalizing over the values of all other features (the 'complement' features). Intuitively, we can interpret the partial dependence as the expected target response as a function of the 'target' features.
2. Due to the limits of human perception the size of the target feature set must be small (usually, one or two) thus the target features are usually chosen among the most important features.
3. The `sklearn.inspection` module provides a convenience function plot_partial_dependence to create one-way and two-way partial dependence plots.

Permutation feature importance

1. Permutation feature importance is a model inspection technique that can be used for any fitted estimator when the data is rectangular.
2. This is especially useful for non-linear or opaque estimators.
3. The permutation feature importance is defined to be the decrease in a model score when a single feature value is randomly shuffled.
4. This procedure breaks the relationship between the feature and the target, thus the drop in the model score is indicative of how much the model depends on the feature.
5. This technique benefits from being model agnostic and can be calculated many times with different permutations of the feature.
6. The permutation_importance function calculates the feature importance of estimators for a given dataset.

7 审查和可视化

- Inspection
- Visualizations

简介

1. Scikit-learn defines a simple API for creating visualizations for machine learning.
2. The key feature of this API is to allow for quick plotting and visual adjustments without recalculation.

Functions

- inspection.plot_partial_dependence(...[, ...]): Partial dependence plots.
- metrics.plot_confusion_matrix(estimator, X, ...): Plot Confusion Matrix.
- metrics.plot_precision_recall_curve(...[, ...]): Plot Precision Recall Curve for binary classifiers.
- metrics.plot_roc_curve(estimator, X, y[, ...]): Plot Receiver operating characteristic (ROC) curve.

简介

Display Objects

- inspection.PartialDependenceDisplay(...): Partial Dependence Plot (PDP) visualization.
- metrics.ConfusionMatrixDisplay(...): Confusion Matrix visualization.
- metrics.PrecisionRecallDisplay(precision, ...): Precision Recall visualization.
- metrics.RocCurveDisplay(fpr, tpr, roc_auc, ...): ROC Curve visualization.

例子

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import plot_roc_curve
from sklearn.datasets import load_wine

X,y=load_wine(return_X_y=True)
y = y == 2
X_train, X_test, y_train, y_test = train_test_split(X, y,
    random_state=42)
svc = SVC(random_state=42)
svc.fit(X_train, y_train)
svc_disp = plot_roc_curve(svc, X_test, y_test)

import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(random_state=42)
rfc.fit(X_train, y_train)
ax = plt.gca()
rfc_disp = plot_roc_curve(rfc, X_test, y_test, ax=ax, alpha
    =0.8)
```

简介

1. scikit-learn provides a library of transformers, which may clean (see Preprocessing data), reduce (see Unsupervised dimensionality reduction), expand (see Kernel Approximation) or generate (see Feature extraction) feature representations.
2. Like other estimators, these are represented by classes with a `fit` method, which learns model parameters (e.g. mean and standard deviation for normalization) from a training set, and a `transform` method which applies this transformation model to unseen data.

简介

1. fit_transform may be more convenient and efficient for modelling and transforming the training data simultaneously.
2. Combining such transformers, either in parallel or series is covered in Pipelines and composite estimators.
3. Pairwise metrics, Affinities and Kernels covers transforming feature spaces into affinity matrices, while Transforming the prediction target (y) considers transformations of the target space (e.g. categorical labels) for use in scikit-learn.

8 数据集转换

- Pipelines and composite estimators
- Feature extraction
- Preprocessing data
- Imputation of missing values
- Unsupervised dimensionality reduction
- Random Projection
- 其他数据集转换方法

管道 (pipeline)

1. Transformers are usually combined with classifiers, regressors or other estimators to build a composite estimator.
2. The most common tool is a Pipeline.
3. Pipeline is often used in combination with `FeatureUnion` which concatenates the output of transformers into a composite feature space.
4. `TransformedTargetRegressor` deals with transforming the target (i.e. log-transform y).
5. In contrast, Pipelines only transform the observed data (X).

Pipeline: chaining estimators

1. Pipeline can be used to chain multiple estimators into one. This is useful as there is often a fixed sequence of steps in processing the data, for example feature selection, normalization and classification.
2. All estimators in a pipeline, except the last one, must be transformers (i.e. must have a transform method). The last estimator may be any type (transformer, classifier, etc.).
3. Calling `fit` on the pipeline is the same as calling `fit` on each estimator in turn, transform the input and pass it on to the next step.
4. The pipeline has all the methods that the last estimator in the pipeline has, i.e. if the last estimator is a classifier, the Pipeline can be used as a classifier. If the last estimator is a transformer, again, so is the pipeline.

管道作用

- Pipeline serves multiple purposes:

Convenience and encapsulation

- You only have to call fit and predict once on your data to fit a whole sequence of estimators.

Joint parameter selection

- You can grid search over parameters of all estimators in the pipeline at once.

Safety

- Pipelines help avoid leaking statistics from your test data into the trained model in cross-validation, by ensuring that the same samples are used to train the transformers and predictors.

管道构建方法

1. The `Pipeline` is built using a `list` of `(key, value)` pairs, where the key is a string containing the name you want to give this step and value is an estimator object.
2. The utility function make_pipeline is a shorthand for constructing pipelines; it takes a variable number of estimators and returns a pipeline, filling in the names automatically.

管道构建例子

```
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.decomposition import PCA
estimators = [('reduce_dim', PCA()), ('clf', SVC())]
pipe = Pipeline(estimators)
pipe

from sklearn.pipeline import make_pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import Binarizer
make_pipeline(Binarizer(), MultinomialNB())
```

获取中间处理步骤: Accessing steps

1. The estimators of a pipeline are stored as a list in the steps attribute, but can be accessed by index or name by indexing (with [idx]) the Pipeline.
2. Pipeline′ s named_steps attribute allows accessing steps by name with tab completion in interactive environments.
3. A sub-pipeline can also be extracted using the slicing notation commonly used for Python Sequences such as lists or strings (although only a step of 1 is permitted). This is convenient for performing only some of the transformations (or their inverse).
4. 管道中参数设置方法: Parameters of the estimators in the pipeline can be accessed using the <estimator>__<parameter> syntax.

例子

```
pipe.steps[0]
pipe[0]

pipe['reduce_dim']
pipe.named_steps.reduce_dim is pipe['reduce_dim']

pipe[:1]
pipe[-1:]

pipe.set_params(clf__C=10)

from sklearn.model_selection import GridSearchCV
param_grid = dict(reduce_dim__n_components=[2, 5, 10],
^^I^^I  clf__C=[0.1, 10, 100])
grid_search = GridSearchCV(pipe, param_grid=param_grid)
```

Caching transformers: avoid repeated computation

1. Fitting transformers may be computationally expensive. With its `memory` parameter set, Pipeline will cache each transformer after calling fit.
2. This feature is used to avoid computing the fit transformers within a pipeline if the parameters and input data are identical.
3. A typical example is the case of a grid search in which the transformers can be fitted only once and reused for each configuration.
4. The parameter `memory` is needed in order to cache the transformers. memory can be either a string containing the directory where to cache the transformers or a `joblib.Memory` object.

例子

```
from tempfile import mkdtemp
from shutil import rmtree
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
estimators = [('reduce_dim', PCA()), ('clf', SVC())]
cachedir = mkdtemp()
pipe = Pipeline(estimators, memory=cachedir)
pipe

# Clear the cache directory when you don't need it anymore
rmtree(cachedir)
```

Transforming target in regression

1. `compose.TransformedTargetRegressor` transforms the targets y before fitting a regression model.
2. The predictions are mapped back to the original space via an inverse transform.
3. It takes as an argument the regressor that will be used for prediction, and the transformer that will be applied to the target variable.

例子

```
import numpy as np
from sklearn.datasets import load_boston
from sklearn.compose import TransformedTargetRegressor
from sklearn.preprocessing import QuantileTransformer
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
X, y = load_boston(return_X_y=True)
transformer = QuantileTransformer(output_distribution='normal')
regressor = LinearRegression()
regr = TransformedTargetRegressor(regressor=regressor,
^^I^^I^^I^^I  transformer=transformer)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    random_state=0)
regr.fit(X_train, y_train)

print('R2 score: {0:.2f}'.format(regr.score(X_test, y_test)))

raw_target_regr = LinearRegression().fit(X_train, y_train)
print('R2 score: {0:.2f}'.format(raw_target_regr.score(X_test,
    y_test)))
```

FeatureUnion: composite feature spaces

1. `FeatureUnion` combines several transformer objects into a new transformer that combines their output.
2. A FeatureUnion takes a list of transformer objects. During fitting, each of these is fit to the data independently.
3. The transformers are applied in parallel, and the feature matrices they output are concatenated side-by-side into a larger matrix.
4. FeatureUnion serves the same purposes as Pipeline - convenience and joint parameter estimation and validation.
5. FeatureUnion and Pipeline can be combined to create complex models.

用法

1. A FeatureUnion is built using a list of `(key, value)` pairs, where the key is the name you want to give to a given transformation (an arbitrary string; it only serves as an identifier) and value is an estimator object.
2. Like pipelines, feature unions have a shorthand constructor called make_union that does not require explicit naming of the components.
3. Like Pipeline, individual steps may be replaced using set_params, and ignored by setting to 'drop'.

例子

```
from sklearn.pipeline import FeatureUnion
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
estimators = [('linear_pca', PCA()), ('kernel_pca', KernelPCA()
    )]
combined = FeatureUnion(estimators)
combined

combined.set_params(kernel_pca='drop')
```

ColumnTransformer for heterogeneous data

1. Many datasets contain features of different types, say text, floats, and dates, where each type of feature requires separate preprocessing or feature extraction steps.
2. Often it is easiest to preprocess data before applying scikit-learn methods, for example using pandas.
3. Processing your data before passing it to scikit-learn might be problematic for one of the following reasons:
   1. Incorporating statistics from test data into the preprocessors makes cross-validation scores unreliable (known as data leakage), for example in the case of scalers or imputing missing values.
   2. You may want to include the parameters of the preprocessors in a parameter search.

ColumnTransformer for heterogeneous data

1. The `ColumnTransformer` helps performing different transformations for different columns of the data, within a Pipeline that is safe from data leakage and that can be parametrized. `ColumnTransformer` works on arrays, sparse matrices, and pandas DataFrames.
2. To each column, a different transformation can be applied, such as preprocessing or a specific feature extraction method.
3. The make_column_transformer function is available to more easily create a `ColumnTransformer` object. Specifically, the names will be given automatically.
4. 例子
   1. Column Transformer with Heterogeneous Data Sources
   2. Column Transformer with Mixed Types

8 数据集转换

- Pipelines and composite estimators
- Feature extraction
- Preprocessing data
- Imputation of missing values
- Unsupervised dimensionality reduction
- Random Projection
- 其他数据集转换方法

简介

1. The sklearn.feature_extraction module can be used to extract features in a format supported by machine learning algorithms from datasets consisting of formats such as text and image.
2. Feature extraction is very different from Feature selection: the former consists in transforming arbitrary data, such as text or images, into numerical features usable for machine learning. The latter is a machine learning technique applied on these features.

Loading features from dicts

1. The class `DictVectorizer` can be used to convert feature arrays represented as lists of standard Python dict objects to the NumPy/SciPy representation used by scikit-learn estimators.
2. While not particularly fast to process, Python′ s dict has the advantages of being convenient to use, being sparse (absent features need not be stored) and storing feature names in addition to values.
3. `DictVectorizer` implements what is called one-of-K or "one-hot" coding for categorical (aka nominal, discrete) features.
4. Categorical features are "attribute-value" pairs where the value is restricted to a list of discrete of possibilities without ordering (e.g. topic identifiers, types of objects, tags, names...).

例子

- In the following, "city" is a categorical attribute while "temperature" is a traditional numerical feature.

```
measurements = [
    {'city': 'Dubai', 'temperature': 33.},
    {'city': 'London', 'temperature': 12.},
    {'city': 'San Francisco', 'temperature': 18.},
]

from sklearn.feature_extraction import DictVectorizer
vec = DictVectorizer()

vec.fit_transform(measurements).toarray()
vec.get_feature_names()
```

Text feature extraction

1. Text Analysis is a major application field for machine learning algorithms.
2. However the raw data, a sequence of symbols cannot be fed directly to the algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.
3. In order to address this, scikit-learn provides utilities(bag) for the most common ways to extract numerical features from text content.

Text feature extraction

bag(multiset)

1. In mathematics, a multiset (aka bag or mset) is a modification of the concept of a set that, unlike a set, allows for multiple instances for each of its elements.
2. The positive integer number of instances, given for each element is called the multiplicity of this element in the multiset.

bag utilities

1. tokenizing strings and giving an integer id for each possible token, for instance by using white-spaces and punctuation as token separators.
2. counting the occurrences of tokens in each document.
3. normalizing and weighting with diminishing importance tokens that occur in the majority of samples / documents.

features and samples

1. each individual token occurrence frequency (normalized or not) is treated as a feature.
2. the vector of all the token frequencies for a given document is considered a multivariate sample.
3. A corpus of documents can thus be represented by a matrix with one row per document and one column per token (e.g. word) occurring in the corpus.

Bag of Words

1. We call vectorization the general process of turning a collection of text documents into numerical feature vectors.
2. This specific strategy (tokenization, counting and normalization) is called the Bag of Words representation.
3. Documents are described by word occurrences while completely ignoring the relative position information of the words in the document.

Sparsity

1. As most documents will typically use a very small subset of the words used in the corpus, the resulting matrix will have many feature values that are zeros (typically more than 99% of them).
2. For instance a collection of 10,000 short text documents (such as emails) will use a vocabulary with a size in the order of 100,000 unique words in total while each document will use 100 to 1000 unique words individually.
3. In order to be able to store such a matrix in memory but also to speed up algebraic operations matrix/vector, implementations will typically use a sparse representation such as the implementations available in the `scipy.sparse` package.

Stop words

1. Stop words are words like “and”, “the”, “him”, which are presumed to be uninformative in representing the content of a text, and which may be removed to avoid them being construed as signal for prediction.
2. Sometimes, however, similar words are useful for prediction, such as in classifying writing style or personality.
3. take care in choosing a stop word list. Popular stop word lists may include words that are highly informative to some tasks, such as computer.
4. You should also make sure that the stop word list has had the same preprocessing and tokenization applied as the one used in the vectorizer.

Image feature extraction

Patch extraction

1. The extract_patches_2d function extracts patches from an image stored as a two-dimensional array, or three-dimensional with color information along the third axis.
2. For rebuilding an image from all its patches, use reconstruct_from_patches_2d.

8 数据集转换

- Pipelines and composite estimators
- Feature extraction
- Preprocessing data
- Imputation of missing values
- Unsupervised dimensionality reduction
- Random Projection
- 其他数据集转换方法

简介

1. The `sklearn.preprocessing` package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.
2. In general, learning algorithms benefit from standardization of the data set. If some outliers are present in the set, robust scalers or transformers are more appropriate.

标准化 (Standardization)

1. Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn;
2. they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance.
3. In practice we often ignore the shape of the distribution and just transform the data to center it by removing the mean value of each feature, then scale it by dividing non-constant features by their standard deviation.

标准化实施方法

1. The function `scale` provides a quick and easy way to perform this operation on a single array-like dataset.
2. Scaled data has zero mean and unit variance.
3. The `preprocessing` module further provides a utility class `StandardScaler` that implements the Transformer API to compute the mean and standard deviation on a training set so as to be able to later reapply the same transformation on the testing set.
4. It is possible to disable either centering or scaling by either passing with_mean=False or with_std=False to the constructor of `StandardScaler`.

标准化例子

```
from sklearn import preprocessing
import numpy as np
X_train = np.array([[ 1., -1.,  2.],
^^I^^I    [ 2.,  0.,  0.],
^^I^^I    [ 0.,  1., -1.]])
X_scaled = preprocessing.scale(X_train)

X_scaled
X_scaled.mean(axis=0)
X_scaled.std(axis=0)

scaler = preprocessing.StandardScaler().fit(X_train)
scaler
scaler.mean_
scaler.scale_

scaler.transform(X_train)

X_test = [[-1., 1., 0.]]
scaler.transform(X_test)
```

Scaling features to a range

1. An alternative standardization is scaling features to lie between a given minimum and maximum value, often between zero and one, or so that the maximum absolute value of each feature is scaled to unit size.
2. This can be achieved using `MinMaxScaler` or `MaxAbsScaler`, respectively.
3. As with `scale`, the module further provides convenience functions minmax_scale and maxabs_scale if you don' t want to create an object.
4. If MinMaxScaler is given an explicit feature_range=(min, max) the full formula is:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

例子

```
X_train = np.array([[ 1., -1.,  2.],
^^I^^I    [ 2.,  0.,  0.],
^^I^^I    [ 0.,  1., -1.]])

min_max_scaler = preprocessing.MinMaxScaler()
X_train_minmax = min_max_scaler.fit_transform(X_train)
X_train_minmax

X_test = np.array([[-3., -1.,  4.]])
X_test_minmax = min_max_scaler.transform(X_test)
X_test_minmax

min_max_scaler.scale_
min_max_scaler.min_
```

Non-linear transformation

1. Two types of transformations are available: quantile transforms and power transforms.
2. Both quantile and power transforms are based on monotonic transformations of the features and thus preserve the rank of the values along each feature.
3. Quantile transforms put all features into the same desired distribution based on the formula $G^{-1}(F(X))$ where F is the cumulative distribution function of the feature and G^{-1} the quantile function of the desired output distribution G.
4. By performing a rank transformation, a quantile transform smooths out unusual distributions and is less influenced by outliers than scaling methods. It does, however, distort correlations and distances within and across features.
5. Power transforms are a family of parametric transformations that aim to map data from any distribution to as close to a Gaussian distribution.

Mapping to a Uniform distribution

1. `QuantileTransformer` and quantile_transform provide a non-parametric transformation to map the data to a uniform distribution with values between 0 and 1.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    random_state=0)
quantile_transformer = preprocessing.QuantileTransformer(random
    _state=0)
X_train_trans = quantile_transformer.fit_transform(X_train)
X_test_trans = quantile_transformer.transform(X_test)
np.percentile(X_train[:, 0], [0, 25, 50, 75, 100])
```

Mapping to a Gaussian distribution

1. In many modeling scenarios, normality of the features in a dataset is desirable. Power transforms are a family of parametric, monotonic transformations that aim to map data from any distribution to as close to a Gaussian distribution as possible in order to stabilize variance and minimize skewness.
2. PowerTransformer currently provides two such power transformations, the Yeo-Johnson transform and the Box-Cox transform.
3. Box-Cox can only be applied to strictly positive data. In both methods, the transformation is parameterized by λ , which is determined through maximum likelihood estimation.
4. It is also possible to map data to a normal distribution using QuantileTransformer by setting output_distribution='normal'.

例子

```
pt = preprocessing.PowerTransformer(method='box-cox',
    standardize=False)
X_lognormal = np.random.RandomState(616).lognormal(size=(3, 3))
X_lognormal
pt.fit_transform(X_lognormal)

quantile_transformer = preprocessing.QuantileTransformer(
    output_distribution='normal', random_state=0)
X_trans = quantile_transformer.fit_transform(X)
quantile_transformer.quantiles_
```

正规化 (Normalization)

1. Normalization is the process of scaling individual samples to have unit norm. This process can be useful if you plan to use a quadratic form such as the dot-product or any other kernel to quantify the similarity of any pair of samples.
2. The function normalize provides a quick and easy way to perform this operation on a single array-like dataset, either using the l1 or l2 norms.
3. The `preprocessing` module further provides a utility class `Normalizer` that implements the same operation using the Transformer API (even though the `fit` method is useless in this case: the class is stateless as this operation treats samples independently).

例子

```
from sklearn import preprocessing
import numpy as np

X = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.]]
X_normalized_L2 = preprocessing.normalize(X, norm='l2')
X_normalized_L1 = preprocessing.normalize(X, norm='l1')


X_normalized_L1
X_normalized_L2

normalizer = preprocessing.Normalizer().fit(X)  # fit does
    nothing
normalizer

normalizer.transform(X)
normalizer.transform([[-1.,  1., 0.]])
```

分类属性编码 Encoding categorical features

1. Often features are not given as continuous values but categorical.
2. To convert categorical features to integer codes, we can use the `OrdinalEncoder` .
3. This estimator transforms each categorical feature to one new feature of integers (0 to n_categories - 1)
4. Such integer representation can, however, not be used directly with all scikit-learn estimators, as these expect continuous input, and would interpret the categories as being ordered, which is often not desired.

分类属性编码 Encoding categorical features

1. Another possibility to convert categorical features to features that can be used with scikit-learn estimators is to use a one-of-K, also known as one-hot or dummy encoding.
2. This type of encoding can be obtained with the `OneHotEncoder`, which transforms each categorical feature with n_categories possible values into n_categories binary features, with one of them 1, and all others 0.
3. By default, the values each feature can take is inferred automatically from the dataset and can be found in the categories_ attribute.
4. It is possible to specify this explicitly using the `parameter` categories.

分类属性编码 Encoding categorical features

1. It is also possible to encode each column into n_categories - 1 columns instead of n_categories columns by using the `drop` parameter.
2. This parameter allows the user to specify a category for each feature to be dropped. This is useful to avoid co-linearity in the input matrix in some classifiers.

例子

```
enc = preprocessing.OrdinalEncoder()
X = [['male', 'from US', 'uses Safari'], ['female', 'from
    Europe', 'uses Firefox']]
enc.fit(X)
enc.transform([['female', 'from US', 'uses Safari']])

enc = preprocessing.OneHotEncoder()
X = [['male', 'from US', 'uses Safari'], ['female', 'from
    Europe', 'uses Firefox']]
enc.fit(X)
enc.transform([['female', 'from US', 'uses Safari'],
^^I       ['male', 'from Europe', 'uses Safari']]).toarray()

enc.categories_
```

例子

```
genders = ['female', 'male']
locations = ['from Africa', 'from Asia', 'from Europe', 'from
    US']
browsers = ['uses Chrome', 'uses Firefox', 'uses IE', 'uses
    Safari']
enc = preprocessing.OneHotEncoder(categories=[genders,
    locations, browsers])
# Note that for there are missing categorical values
# for the 2nd and 3rd feature
X = [['male', 'from US', 'uses Safari'], ['female', 'from
    Europe', 'uses Firefox']]
enc.fit(X)
enc.transform([['female', 'from Asia', 'uses Chrome']]).toarray
    ()

X = [['male', 'from US', 'uses Safari'], ['female', 'from
    Europe', 'uses Firefox']]
drop_enc = preprocessing.OneHotEncoder(drop='first').fit(X)
drop_enc.categories_

drop_enc.transform(X).toarray()
```

离散化 Discretization

1. Discretization (otherwise known as quantization or binning) provides a way to partition continuous features into discrete values.
2. Certain datasets with continuous features may benefit from discretization, because discretization can transform the dataset of continuous attributes to one with only nominal attributes.

K-bins discretization

1. K-bins discretization: `KBinsDiscretizer` discretizes features into k bins.
2. By default the output is one-hot encoded into a sparse matrix and this can be configured with the `encode` parameter.
3. For each feature, the bin edges are computed during fit and together with the number of bins, they will define the intervals.
4. Discretization is similar to constructing histograms for continuous data. However, histograms focus on counting features which fall into particular bins, whereas discretization focuses on assigning feature values to these bins.
5. `KBinsDiscretizer` implements different binning strategies, which can be selected with the `strategy` parameter. The 'uniform' strategy uses constant-width bins.

例子

```
from sklearn import preprocessing
import numpy as np

X = np.array([[ -3., 5., 15 ],
^^I       [  0., 6., 14 ],
^^I       [  6., 3., 11 ]])
est = preprocessing.KBinsDiscretizer(n_bins=[3, 2, 2], encode='
    ordinal').fit(X)

est.transform(X)
```

Feature binarization

1. Feature binarization is the process of thresholding numerical features to get boolean values.
2. This can be useful for downstream probabilistic estimators that make assumption that the input data is distributed according to a multi-variate Bernoulli distribution.
3. As for the `Normalizer`, the utility class `Binarizer` is meant to be used in the early stages of `sklearn.pipeline.Pipeline`. The fit method does nothing as each sample is treated independently of others.
4. It is possible to adjust the threshold of the binarizer by using `threshold` parameter.
5. the preprocessing module provides a companion function binarize to be used when the transformer API is not necessary.
6. Note that the `Binarizer` is similar to the `KBinsDiscretizer` when k = 2, and when the bin edge is at the value threshold.

例子

```
from sklearn import preprocessing
import numpy as np

X = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.]]

binarizer = preprocessing.Binarizer().fit(X)  # fit does nothing
binarizer
binarizer.transform(X)

binarizer = preprocessing.Binarizer(threshold=1.1)
binarizer.transform(X)
```

Generating polynomial features

1. Often it′ s useful to add complexity to the model by considering nonlinear features of the input data.
2. A simple and common method to use is polynomial features, which can get features′ high-order and interaction terms.
3. It is implemented in `PolynomialFeatures` .
4. In some cases, only interaction terms among features are required, and it can be gotten with the setting interaction_only=True

例子

1. The features of X have been transformed (X_1, X_2) from to $(1, X_1, X_2, X_1^2, X_1X_2, X_2^2)$.
2. The features of X have been transformed (X_1, X_2, X_3) from to $(1, X_1, X_2, X_3, X_1X_2, X_1X_3, X_2X_3, X_1X_2X_3)$.

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
X = np.arange(6).reshape(3, 2)
X

poly = PolynomialFeatures(2)
poly.fit_transform(X)

X = np.arange(9).reshape(3, 3)
X

poly = PolynomialFeatures(degree=3, interaction_only=True)
poly.fit_transform(X)
```

Custom transformers

1. Often, you will want to convert an existing Python function into a transformer to assist in data cleaning or processing.
2. You can implement a transformer from an arbitrary function with `FunctionTransformer` .
3. For example, to build a transformer that applies a log transformation in a pipeline, do:

```
import numpy as np
from sklearn.preprocessing import FunctionTransformer

transformer = FunctionTransformer(np.log1p, validate=True)
X = np.array([[0, 1], [2, 3]])
transformer.transform(X)
```

8 数据集转换

- Pipelines and composite estimators
- Feature extraction
- Preprocessing data
- Imputation of missing values
- Unsupervised dimensionality reduction
- Random Projection
- 其他数据集转换方法

简介

1. For various reasons, many real world datasets contain missing values, often encoded as blanks, NaNs or other placeholders.
2. Such datasets however are incompatible with scikit-learn estimators which assume that all values in an array are numerical, and that all have and hold meaning.
3. A basic strategy to use incomplete datasets is to discard entire rows and/or columns containing missing values. However, this comes at the price of losing data which may be valuable (even though incomplete).
4. A better strategy is to impute the missing values, i.e., to infer them from the known part of the data.

Univariate vs. Multivariate Imputation

1. One type of imputation algorithm is univariate, which imputes values in the i-th feature dimension using only non-missing values in that feature dimension (e.g. `impute.SimpleImputer`).
2. By contrast, multivariate imputation algorithms use the entire set of available feature dimensions to estimate the missing values (e.g. `impute.IterativeImputer`).

Univariate feature imputation

1. The `SimpleImputer` class provides basic strategies for imputing missing values.
2. Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located.
3. This class also allows for different missing values encodings.
4. The `SimpleImputer` class also supports categorical data represented as string values or pandas categoricals when using the 'most_frequent' or 'constant' strategy.

例子

```
# replace missing values, encoded as np.nan, using the mean
import numpy as np
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp.fit([[1, 2], [np.nan, 3], [7, 6]])

X = [[np.nan, 2], [6, np.nan], [7, 6]]
print(imp.transform(X))


import pandas as pd
df = pd.DataFrame([["a", "x"],
^^I^^I   [np.nan, "y"],
^^I^^I   ["a", np.nan],
^^I^^I   ["b", "y"]], dtype="category")

imp = SimpleImputer(strategy="most_frequent")
print(imp.fit_transform(df))
```

Multivariate feature imputation

1. A more sophisticated approach is to use the `IterativeImputer` class, which models each feature with missing values as a function of other features, and uses that estimate for imputation.
2. It does so in an iterated round-robin fashion: at each step, a feature column is designated as output y and the other feature columns are treated as inputs X. A regressor is fit on (X, y) for known y. Then, the regressor is used to predict the missing values of y.
3. This is done for each feature in an iterative fashion, and then is repeated for max_iter imputation rounds. The results of the final imputation round are returned.

Flexibility of IterativeImputer

1. There are many well-established imputation packages in the R data science ecosystem: Amelia, mi, mice, missForest, etc.
2. missForest is popular, and turns out to be a particular instance of different sequential imputation algorithms that can all be implemented with `IterativeImputer` by passing in different regressors to be used for predicting missing feature values.
3. In the case of missForest, this regressor is a Random Forest.

Multiple vs. Single Imputation

1. In the statistics community, it is common practice to perform multiple imputations, generating, for example, `m` separate imputations for a single feature matrix.
2. Each of these `m` imputations is then put through the subsequent analysis pipeline (e.g. feature engineering, clustering, regression, classification).
3. The `m` final analysis results (e.g. held-out validation errors) allow the data scientist to obtain understanding of how analytic results may differ as a consequence of the inherent uncertainty caused by the missing values.
4. The above practice is called multiple imputation.

IterativeImputer: Single Imputation

1. Our implementation of `IterativeImputer` was inspired by the R MICE package (Multivariate Imputation by Chained Equations),
2. but differs from it by returning a single imputation instead of multiple imputations.
3. However, `IterativeImputer` can also be used for multiple imputations by applying it repeatedly to the same dataset with different random seeds when sample_posterior=True.

Nearest neighbors imputation

1. The `KNNImputer` class provides imputation for filling in missing values using the k-Nearest Neighbors approach.
2. By default, a euclidean distance metric that supports missing values, nan_euclidean_distances, is used to find the nearest neighbors.
3. Each missing feature is imputed using values from n_neighbors nearest neighbors that have a value for the feature.
4. The feature of the neighbors are averaged uniformly or weighted by distance to each neighbor.
5. If a sample has more than one feature missing, then the neighbors for that sample can be different depending on the particular feature being imputed.

例子

1. The following snippet demonstrates how to replace missing values, encoded as `np.nan`, using the mean feature value of the two nearest neighbors of samples with missing values:

```
import numpy as np
from sklearn.impute import KNNImputer

nan = np.nan
X = [[1, 2, nan], [3, 4, 3], [nan, 6, 5], [8, 8, 7]]
imputer = KNNImputer(n_neighbors=2, weights="uniform")
imputer.fit_transform(X)
```

Marking imputed values

1. The `MissingIndicator` transformer is useful to transform a dataset into corresponding binary matrix indicating the presence of missing values in the dataset.
2. This transformation is useful in conjunction with imputation.
3. When using imputation, preserving the information about which values had been missing can be informative.
4. Note that both the `SimpleImputer` and `IterativeImputer` have the boolean parameter add_indicator (False by default) which when set to True provides a convenient way of stacking the output of the `MissingIndicator` transformer with the output of the imputer.

Marking imputed values

5. `NaN` is usually used as the placeholder for missing values. However, it enforces the data type to be float. The parameter missing_values allows to specify other placeholder such as integer. In the following example, we will use -1 as missing values.
6. The `features` parameter is used to choose the features for which the mask is constructed. By default, it is 'missing-only' which returns the imputer mask of the features containing missing values at fit time.

例子

```
from sklearn.impute import MissingIndicator
X = np.array([[-1, -1, 1, 3],
^^I       [4, -1, 0, -1],
^^I       [8, -1, 1, 0]])
indicator = MissingIndicator(missing_values=-1)
mask_missing_values_only = indicator.fit_transform(X)
mask_missing_values_only

indicator.features_
```

8 数据集转换

- Pipelines and composite estimators
- Feature extraction
- Preprocessing data
- Imputation of missing values
- Unsupervised dimensionality reduction
- Random Projection
- 其他数据集转换方法

Unsupervised dimensionality reduction

1. If your number of features is high, it may be useful to reduce it with an unsupervised step prior to supervised steps.
2. Many of the Unsupervised learning methods implement a transform method that can be used to reduce the dimensionality.

PCA: principal component analysis

- `decomposition.PCA` looks for a combination of features that capture well the variance of the original features.

Random projections

- The module: random_projection provides several tools for data reduction by random projections.

Feature agglomeration

- `cluster.FeatureAgglomeration` applies Hierarchical clustering to group together features that behave similarly.

8 数据集转换

- Pipelines and composite estimators
- Feature extraction
- Preprocessing data
- Imputation of missing values
- Unsupervised dimensionality reduction
- Random Projection
- 其他数据集转换方法

简介

1. The sklearn.random_projection module implements a simple and computationally efficient way to reduce the dimensionality of the data by trading a controlled amount of accuracy (as additional variance) for faster processing times and smaller model sizes.
2. This module implements two types of unstructured random matrix: Gaussian random matrix and sparse random matrix.
3. The dimensions and distribution of random projections matrices are controlled so as to preserve the pairwise distances between any two samples of the dataset. Thus random projection is a suitable approximation technique for distance based method.

Gaussian random projection

1. Let's say we have a numeric dataset with n examples, each of which is represented by d features (where d is presumably relatively large, maybe on the order of hundreds or thousands).
2. In other words, our data is a matrix X, with n rows and d columns.
3. Suppose we want to reduce the dimensionality of our data so that each example is represented by only k features, where k is small, like 2 or 10.

1. For Gaussian random projection we construct a projection matrix R with d rows and k columns.
2. Each entry is independently sampled from a standard Gaussian distribution

$$R_{ij} \sim N(0, 1)$$

Gaussian random projection

- The projection is done by multiplying our data matrix by the projection matrix:

$$Y = \frac{1}{\sqrt{k}} XR$$

so that our output dataset Y has n rows with only k columns.

The scalar $1/\sqrt{k}$ ensures that the Euclidean distance between any two points in the new low-dimensional space is very close to the distance between the same points in the original high-dimensional space, with high probability.

- The sklearn.random_projection.GaussianRandomProjection reduces the dimensionality by projecting the original input space on a randomly generated matrix where components are drawn from the following distribution $N(0, \frac{1}{n_{components}})$.

例子

```
import numpy as np
from sklearn import random_projection

X = np.random.rand(100, 10000)
transformer = random_projection.GaussianRandomProjection()
X_new = transformer.fit_transform(X)
X_new.shape
```

8 数据集转换

- Pipelines and composite estimators
- Feature extraction
- Preprocessing data
- Imputation of missing values
- Unsupervised dimensionality reduction
- Random Projection
- 其他数据集转换方法

Kernel Approximation

- This submodule contains functions that approximate the feature mappings that correspond to certain kernels, as they are used for example in support vector machines.
- Nystroem Method for Kernel Approximation
- Radial Basis Function Kernel
- Additive Chi Squared Kernel
- Skewed Chi Squared Kernel

Pairwise metrics, Affinities and Kernels

1. The `sklearn.metrics.pairwise` submodule implements utilities to evaluate pairwise distances or affinity of sets of samples.
2. This module contains both distance metrics and kernels.
3. Cosine similarity
4. Linear kernel
5. Polynomial kernel
6. Sigmoid kernel
7. RBF kernel
8. Laplacian kernel
9. Chi-squared kernel

Transforming the prediction target (y)

1. These are transformers that are not intended to be used on features, only on supervised learning targets.
2. `LabelBinarizer` is a utility class to help create a label indicator matrix from a list of multi-class labels.
3. `LabelEncoder` is a utility class to help normalize labels such that they contain only values between 0 and n_classes-1.

简介

1. The sklearn.datasets package embeds some small toy datasets.
2. This package also features helpers to fetch larger datasets commonly used by the machine learning community to benchmark algorithms on data that comes from the 'real world'.
3. To evaluate the impact of the scale of the dataset (n_samples and n_features) while controlling the statistical properties of the data (typically the correlation and informativeness of the features), it is also possible to generate synthetic data.

9 数据集导入

- General dataset API
- Generated datasets
- Loading other datasets
- Loading from external datasets

loader and fetcher

1. The dataset loaders. They can be used to load small standard datasets, described in the Toy datasets section.
2. The dataset fetchers. They can be used to download and load larger datasets, described in the Real world datasets section.
3. Both loaders and fetchers functions return a dictionary-like object holding at least two items: an array of shape n_samples * n_features with key `data` (except for 20newsgroups).
4. and a numpy array of length n_samples, containing the target values, with key `target`.

loader and fetcher

1. It′ s also possible for almost all of these function to constrain the output to be a tuple containing only the data and the target, by setting the return_X_y parameter to True.
2. The datasets also contain a full description in their DESCR attribute and some contain feature_names and target_names. See the dataset descriptions below for details.

The dataset generation functions

1. The dataset generation functions. They can be used to generate controlled synthetic datasets, described in the Generated datasets section.
2. These functions return a `tuple (X, y)` consisting of a n_samples * n_features numpy array X and an array of length n_samples containing the targets y.

Toy datasets

1. scikit-learn comes with a few small standard datasets that do not require to download any file from some external website.
2. 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.
3. They can be loaded using the following functions:

load_boston()	Load and return the boston house-prices dataset (regression).
load_iris()	Load and return the iris dataset (classification).
load_diabetes()	Load and return the diabetes dataset (regression).
load_digits()	Load and return the digits dataset (classification).
load_linnerud()	Load and return the linnerud dataset (multivariate regression).
load_wine()	Load and return the wine dataset (classification).
load_breast_cancer()	Load and return the breast cancer wisconsin dataset (classification).

Real world datasets

1. scikit-learn provides tools to load larger datasets, downloading them if necessary.
2. They can be loaded using the following functions:

fetch_olivetti_faces	Load the Olivetti faces data-set from AT&T (classification).
fetch_20newsgroups	Load the filenames and data from the 20 newsgroups dataset (classification).
fetch_20newsgroups_vectorized	Load the 20 newsgroups dataset and vectorize it into token counts (classification).
fetch_lfw_people	Load the Labeled Faces in the Wild (LFW) people dataset (classification).
fetch_lfw_pairs	Load the Labeled Faces in the Wild (LFW) pairs dataset (classification).
fetch_covtype([data_home, ...])	Load the covertype dataset (classification).
fetch_rcv1	Load the RCV1 multilabel dataset (classification).
fetch_kddcup99	Load the kddcup99 dataset (classification).
fetch_california_housing	Load the California housing dataset (regression).

9 数据集导入

- General dataset API
- Generated datasets
- Loading other datasets
- Loading from external datasets

Generators for classification and clustering

1. scikit-learn includes various random sample generators that can be used to build artificial datasets of controlled size and complexity.
2. These generators produce a matrix of features and corresponding discrete targets.
3. Single label: Both make_blobs and make_classification create multiclass datasets by allocating each class one or more normally-distributed clusters of points.
4. Multilabel: make_multilabel_classification generates random samples with multiple labels, reflecting a bag of words drawn from a mixture of topics.
5. Biclustering:

make_biclusters(shape, n_clusters[, noise, ...])	Generate an array with constant block diagonal structure for biclustering.
make_checkerboard(shape, n_clusters[, ...])	Generate an array with block checkerboard structure for biclustering.

Generators for regression

1. make_regression produces regression targets as an optionally-sparse random linear combination of random features, with noise.
2. Its informative features may be uncorrelated, or low rank (few features account for most of the variance).

Generators for manifold learning

make_s_curve([n_samples, noise, random_state])	Generate an S curve dataset.
make_swiss_roll([n_samples, noise, random_state])	Generate a swiss roll dataset

Generators for decomposition

make_low_rank_matrix([n_samples, ...])	Generate a mostly low rank matrix with bell-shaped singular values
make_sparse_coded_signal(n_samples, ...[, ...])	Generate a signal as a sparse combination of dictionary elements.
make_spd_matrix(n_dim[, random_state])	Generate a random symmetric, positive-definite matrix.
make_sparse_spd_matrix([dim, alpha, ...])	Generate a sparse symmetric definite positive matrix.

9 数据集导入

- General dataset API
- Generated datasets
- Loading other datasets
- Loading from external datasets

Sample images

1. Scikit-learn also embed a couple of sample JPEG images published under Creative Commons license by their authors.
2. Those images can be useful to test algorithms and pipeline on 2D data.

load_sample_images()	Load sample images for image manipulation.
load_sample_image(image_name)	Load the numpy array of a single sample image

Datasets in svmlight / libsvm format

1. scikit-learn includes utility functions for loading datasets in the svmlight/libsvm format.
2. In this format, each line takes the form <label> <feature-id>:<feature-value> <feature-id>:<feature-value> ….
3. This format is especially suitable for sparse datasets.
4. In this module, scipy sparse CSR matrices are used for X and numpy arrays are used for y.
5. Public datasets in svmlight / libsvm format: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets

Downloading datasets from the openml.org repository

1. openml.org is a public repository for machine learning data and experiments, that allows everybody to upload open datasets.
2. The sklearn.datasets package is able to download datasets from the repository using the function sklearn.datasets.fetch_openml.
3. To fully specify a dataset, you need to provide a name and a version, though the version is optional.
4. You can get more information on the dataset by looking at the `DESCR` and `details` attributes.

```
1 import numpy as np
2 from sklearn.datasets import fetch_openml
3 mice = fetch_openml(name='miceprotein', version=4)
4
5 mice.data.shape
6 mice.target.shape
7 np.unique(mice.target)
8
9 print(mice.DESCR)
```

9 数据集导入

- General dataset API
- Generated datasets
- Loading other datasets
- Loading from external datasets

简介

1. scikit-learn works on any numeric data stored as numpy arrays or scipy sparse matrices.
2. Other types that are convertible to numeric arrays such as pandas DataFrame are also acceptable.
3. Here are some recommended ways to load standard columnar data into a format usable by scikit-learn:
   1. `pandas.io` provides tools to read data from common formats including CSV, Excel, JSON and SQL.
   2. `scipy.io` specializes in binary formats often used in scientific computing context such as .mat and .arff
   3. `numpy/routines.io` for standard loading of columnar data into numpy arrays
   4. scikit-learn′ s datasets.load_svmlight_file for the svmlight or libSVM sparse format
   5. scikit-learn′ s datasets.load_files for directories of text files where the name of each directory is the name of each category and each file inside of each directory corresponds to one sample from that category

images, videos, and audio file

1. For some miscellaneous data such as images, videos, and audio, you may wish to refer to:
   1. `skimage.io` or Imageio for loading images and videos into numpy arrays
   2. `scipy.io.wavfile.read` for reading WAV files into a numpy array.
2. Categorical (or nominal) features stored as strings (common in pandas DataFrames) will need converting to numerical features using `sklearn.preprocessing.OneHotEncoder` or `sklearn.preprocessing.OrdinalEncoder` or similar.