

Machine Learning in Python using Scikit-Learn

17th Nov 2019

About Me



Rohit Walimbe

Data Scientist @ PhonePe

Content

Content

- Background
- Scikit Learn Introduction
- Deep Dive In to Package
- Connecting the dots - Demo
- QA

Duration : 45 Minutes

Source : <https://scikit-learn.org/>

Popularity of Python in ML |

Programming languages

Top Machine Learning Languages on GitHub

- 1 Python
- 2 C++
- 3 JavaScript
- 4 Java
- 5 C#
- 6 Julia
- 7 Shell
- 8 R
- 9 TypeScript
- 10 Scala

Popular machine learning projects

Top Machine Learning Projects on GitHub

- 1 tensorflow/tensorflow
- 2 scikit-learn/scikit-learn
- 3 explosion/spaCy
- 4 JuliaLang/julia
- 5 CMU-Perceptual-Computing-Lab/openpose
- 6 tensorflow/serving
- 7 thtrieu/darkflow
- 8 ageitgey/face_recognition
- 9 RasaHQ/rasa_nlu
- 10 tesseract-ocr/tesseract

Popular machine learning and data science packages

Packages Imported by Machine Learning Projects on GitHub

- | | | |
|----|-----------------|-----|
| 1 | numpy | 74% |
| 2 | scipy | 47% |
| 3 | pandas | 41% |
| 4 | matplotlib | 40% |
| 5 | scikit-learn | 38% |
| 6 | six | 31% |
| 7 | tensorflow | 24% |
| 8 | requests | 23% |
| 9 | python-dateutil | 22% |
| 10 | pytz | 21% |

Scikit-learn

Introduction

About Scikit Learn |

History

This project was started in 2007 as a Google Summer of Code project by David Cournapeau. Later that year, Matthieu Brucher started work on this project as part of his thesis.

In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel of INRIA took leadership of the project and made the first public release, February the 1st 2010. Since then, several releases have appeared following a ~3 month cycle, and a thriving international community has been leading the development.

Installing scikit-learn

Note: If you wish to contribute to the project, it's recommended you [install the latest development version](#).

Installing the latest release

Scikit-learn requires:

- Python (≥ 3.5)
- NumPy ($\geq 1.11.0$)
- SciPy ($\geq 0.17.0$)
- joblib (≥ 0.11)

Scikit-learn plotting capabilities (i.e., functions start with "plot_") require Matplotlib ($\geq 1.5.1$). Some of the scikit-learn examples might require one or more extra dependencies: scikit-image ($\geq 0.12.3$), pandas ($\geq 0.18.0$).

Warning: Scikit-learn 0.20 was the last version to support Python 2.7 and Python 3.4. Scikit-learn now requires Python 3.5 or newer.

If you already have a working installation of numpy and scipy, the easiest way to install scikit-learn is using `pip`

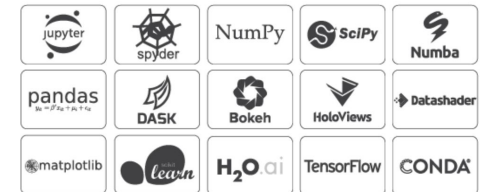
```
pip install -U scikit-learn
```

or `conda`:

```
conda install scikit-learn
```

The open-source [Anaconda Distribution](#) is the easiest way to perform Python/R data science and machine learning on Linux, Windows, and Mac OS X. With over 15 million users worldwide, it is the industry standard for developing, testing, and training on a single machine, enabling *individual data scientists* to:

- Quickly download 1,500+ Python/R data science packages
- Manage libraries, dependencies, and environments with Conda
- Develop and train machine learning and deep learning models with scikit-learn, TensorFlow, and Theano
- Analyze data with scalability and performance with Dask, NumPy, pandas, and Numba
- Visualize results with Matplotlib, Bokeh, Datashader, and Holoviews



Windows | macOS | Linux

Anaconda 2019.10 for Linux Installer

Anaconda comes with Scikit-Learn

<https://github.com/scikit-learn/scikit-learn>

Setting Expectations Right |

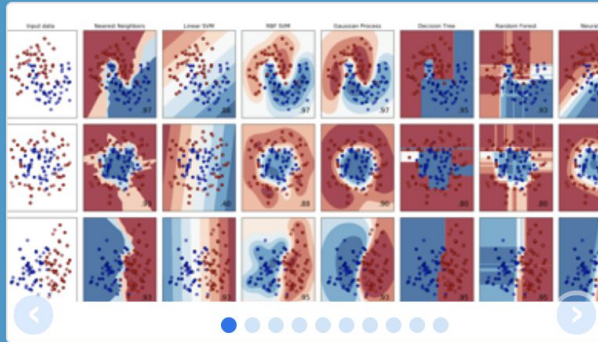
- **NO** GPU Support
- **NO** Deep learning and Reinforcement Learning
 - Only MLP
- Traditional Algorithms
- Inclusion Criteria for new algorithms :
 - Well-established algorithms for inclusion
 - At least 3 years since publication, 200+ citations and wide use and usefulness
 - Only those which fit well within the current API of scikit-learn

About Scikit Learn | <http://scikit-learn.org>



Home Installation Documentation ▾ Examples

Google Custom Search



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.

Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning

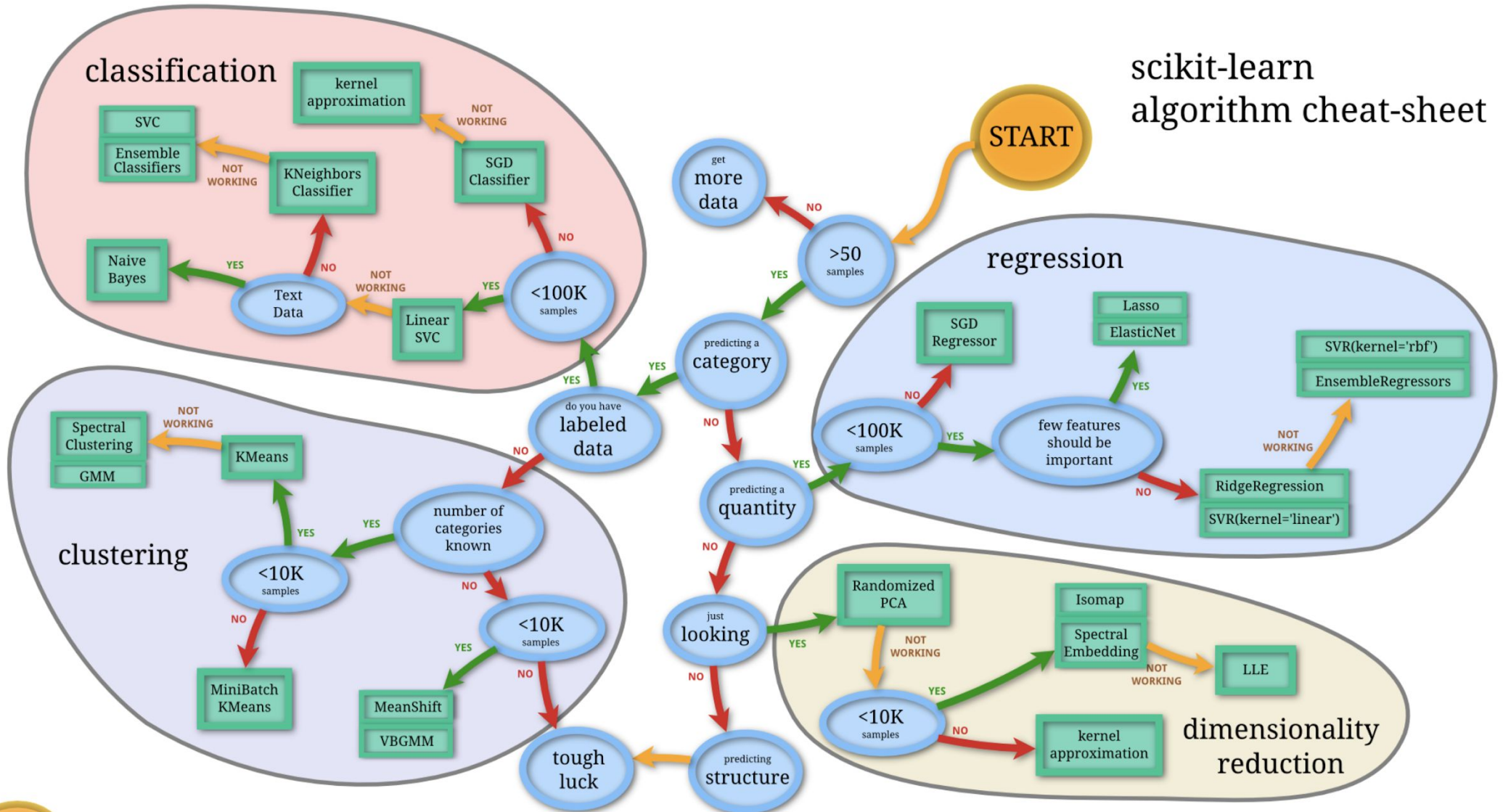
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.

Modules: preprocessing, feature extraction. — Examples



Deep Dive in to package

1- Data Preprocessing

- Standardization , Mean Removal , Variance Scaling : Scaling to range, Scaling Sparse Data

```
from sklearn import preprocessing
StandardScaler, MinMaxScaler(), Scale()
```

- Non Linear Transformations

```
preprocessing.QuantileTransformer()
```

- Normalisation

```
X_normalized =
preprocessing.normalize(X, norm='l2')
```

- Categorical Encoding

```
enc = preprocessing.OneHotEncoder()
```

- Discretization

- K-Bins

```
preprocessing.KBinsDiscretizer(n_bins=[
3, 2, 2], encode='ordinal')
```

- Feature Binarization

```
binarizer = preprocessing.Binarizer()
```

- Imputation of Missing Values

- Univariate

```
from sklearn.impute import
SimpleImputer
```

```
SimpleImputer(missing_values=np.nan,
strategy='mean')
```

```
SimpleImputer(strategy="most_frequent
")
```

- Multivariate

```
from sklearn.impute import
IterativeImputer
```

- Generating Polynomial Features

```
from sklearn.preprocessing import
PolynomialFeatures
```

2- Feature Extraction

- Feature Hashing
 - Scaling to range

```
FeatureHasher(input_type='string')
```

- Text Feature Extraction
 - Bag of Words - Count Vectorizer
 - TFIDF
 - Vectorizing a large text corpus with the hashing trick

```
>>> corpus = [  
...     'This is the first document.',  
...     'This is the second second document.',  
...     'And the third one.',  
...     'Is this the first document?',  
... ]  
  
bigram_vectorizer = CountVectorizer(ngram_range=(1, 2),  
                                  token_pattern=r'\b\w+\b', min_df=1)  
analyze = bigram_vectorizer.build_analyzer()  
analyze('Bi-grams are cool!') == (  
    ['bi', 'grams', 'are', 'cool', 'bi grams', 'grams are', 'are cool'])  
e
```

```
>>> transformer = TfidfTransformer()  
>>> transformer.fit_transform(counts).toarray()  
array([[0.85151335, 0.          , 0.52433293],  
       [1.          , 0.          , 0.          ],  
       [1.          , 0.          , 0.          ],  
       [1.          , 0.          , 0.          ],  
       [0.55422893, 0.83236428, 0.          ],  
       [0.63035731, 0.          , 0.77630514]])
```

- Image Feature Extraction
 - Patch Extraction

Caveat :
Better packages are available for
text and image feature extractors

3- Dimension Reduction

- Principal Component Analysis

```
from sklearn.decomposition import PCA
```

- Incremental PCA
- Kernel PCA

- Truncated SVD

```
from sklearn.decomposition import TruncatedSVD
```

- Feature Selection

```
from sklearn.feature_selection import VarianceThreshold
```

```
from sklearn.feature_selection import SelectFromModel
```

- Non-Negative Matrix Factorisation

```
from sklearn.decomposition import NMF
```

- LDA

```
from sklearn.decomposition import LatentDirichletAllocation
```

4- Classification & Regression

1.1. Generalized Linear Models

- 1.1.1. Ordinary Least Squares
 - 1.1.1.1. Ordinary Least Squares Complexity
- 1.1.2. Ridge Regression
 - 1.1.2.1. Ridge Complexity
 - 1.1.2.2. Setting the regularization parameter: generalized Cross-Validation
- 1.1.3. Lasso
 - 1.1.3.1. Setting regularization parameter
 - 1.1.3.1.1. Using cross-validation
 - 1.1.3.1.2. Information-criteria based model selection
 - 1.1.3.1.3. Comparison with the regularization parameter of SVM
- 1.1.4. Multi-task Lasso
- 1.1.5. Elastic-Net
- 1.1.6. Multi-task Elastic-Net
- 1.1.7. Least Angle Regression
- 1.1.8. LARS Lasso
 - 1.1.8.1. Mathematical formulation
- 1.1.9. Orthogonal Matching Pursuit (OMP)
- 1.1.10. Bayesian Regression
 - 1.1.10.1. Bayesian Ridge Regression
 - 1.1.10.2. Automatic Relevance Determination - ARD
- 1.1.11. Logistic regression
- 1.1.12. Stochastic Gradient Descent - SGD
- 1.1.13. Perceptron
- 1.1.14. Passive Aggressive Algorithms
- 1.1.15. Robustness regression: outliers and modeling errors
 - 1.1.15.1. Different scenario and useful concepts
 - 1.1.15.2. RANSAC: RANdom SAmple Consensus
 - 1.1.15.2.1. Details of the algorithm
 - 1.1.15.3. Theil-Sen estimator: generalized-median-based estimator
 - 1.1.15.3.1. Theoretical considerations
 - 1.1.15.4. Huber Regression
 - 1.1.15.5. Notes
- 1.1.16. Polynomial regression: extending linear models with basis functions

1.3. Kernel ridge regression

1.4. Support Vector Machines

- 1.4.1. Classification
 - 1.4.1.1. Multi-class classification
 - 1.4.1.2. Scores and probabilities
 - 1.4.1.3. Unbalanced problems
- 1.4.2. Regression
- 1.4.3. Density estimation, novelty detection
- 1.4.4. Complexity
- 1.4.5. Tips on Practical Use
- 1.4.6. Kernel functions
 - 1.4.6.1. Custom Kernels
 - 1.4.6.1.1. Using Python functions as kernels
 - 1.4.6.1.2. Using the Gram matrix
 - 1.4.6.1.3. Parameters of the RBF Kernel
- 1.4.7. Mathematical formulation
 - 1.4.7.1. SVC
 - 1.4.7.2. NuSVC
 - 1.4.7.3. SVR
- 1.4.8. Implementation details

4- Classification & Regression

1.6. Nearest Neighbors

- 1.6.1. Unsupervised Nearest Neighbors
 - 1.6.1.1. Finding the Nearest Neighbors
 - 1.6.1.2. KDTree and BallTree Classes
- 1.6.2. Nearest Neighbors Classification
- 1.6.3. Nearest Neighbors Regression
- 1.6.4. Nearest Neighbor Algorithms
 - 1.6.4.1. Brute Force
 - 1.6.4.2. K-D Tree
 - 1.6.4.3. Ball Tree
 - 1.6.4.4. Choice of Nearest Neighbors Algorithm
 - 1.6.4.5. Effect of `leaf_size`
- 1.6.5. Nearest Centroid Classifier
 - 1.6.5.1. Nearest Shrunken Centroid
- 1.6.6. Neighborhood Components Analysis
 - 1.6.6.1. Classification
 - 1.6.6.2. Dimensionality reduction
 - 1.6.6.3. Mathematical formulation
 - 1.6.6.3.1. Mahalanobis distance
 - 1.6.6.4. Implementation
 - 1.6.6.5. Complexity
 - 1.6.6.5.1. Training
 - 1.6.6.5.2. Transform

1.9. Naive Bayes

- 1.9.1. Gaussian Naive Bayes
- 1.9.2. Multinomial Naive Bayes
- 1.9.3. Complement Naive Bayes
- 1.9.4. Bernoulli Naive Bayes
- 1.9.5. Out-of-core naive Bayes model

1.10. Decision Trees

- 1.10.1. Classification
- 1.10.2. Regression
- 1.10.3. Multi-output problems
- 1.10.4. Complexity
- 1.10.5. Tips on practical use
- 1.10.6. Tree algorithms: ID3, C4.5, C5.0 and CART
- 1.10.7. Mathematical formulation
 - 1.10.7.1. Classification criteria
 - 1.10.7.2. Regression criteria

1.11. Ensemble methods

- 1.11.1. Bagging meta-estimator
- 1.11.2. Forests of randomized trees
 - 1.11.2.1. Random Forests
 - 1.11.2.2. Extremely Randomized Trees
 - 1.11.2.3. Parameters
 - 1.11.2.4. Parallelization
 - 1.11.2.5. Feature importance evaluation
 - 1.11.2.6. Totally Random Trees Embedding
- 1.11.3. AdaBoost
 - 1.11.3.1. Usage
- 1.11.4. Gradient Tree Boosting
 - 1.11.4.1. Classification
 - 1.11.4.2. Regression
 - 1.11.4.3. Fitting additional weak-learners
 - 1.11.4.4. Controlling the tree size
 - 1.11.4.5. Mathematical formulation
 - 1.11.4.5.1. Loss Functions
 - 1.11.4.6. Regularization
 - 1.11.4.6.1. Shrinkage
 - 1.11.4.6.2. Subsampling
 - 1.11.4.7. Interpretation
 - 1.11.4.7.1. Feature importance
- 1.11.5. Voting Classifier
 - 1.11.5.1. Majority Class Labels (Majority/Hard Voting)
 - 1.11.5.1.1. Usage
 - 1.11.5.2. Weighted Average Probabilities (Soft Voting)
 - 1.11.5.3. Using the `VotingClassifier` with `GridSearchCV`
 - 1.11.5.3.1. Usage
- 1.11.6. Voting Regressor

4- Classification & Regression

1.12. Multiclass and multilabel algorithms

- 1.12.1. Multilabel classification format
- 1.12.2. One-Vs-The-Rest
 - 1.12.2.1. Multiclass learning
 - 1.12.2.2. Multilabel learning
- 1.12.3. One-Vs-One
 - 1.12.3.1. Multiclass learning
- 1.12.4. Error-Correcting Output-Codes
 - 1.12.4.1. Multiclass learning
- 1.12.5. Multioutput regression
- 1.12.6. Multioutput classification
- 1.12.7. Classifier Chain
- 1.12.8. Regressor Chain

1.17. Neural network models (supervised)

- 1.17.1. Multi-layer Perceptron
- 1.17.2. Classification
- 1.17.3. Regression
- 1.17.4. Regularization
- 1.17.5. Algorithms
- 1.17.6. Complexity
- 1.17.7. Mathematical formulation
- 1.17.8. Tips on Practical Use
- 1.17.9. More control with warm_start

ONLY MLP

6- Clustering

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large n_{samples} , medium n_{clusters} with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_{samples}	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with n_{samples}	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium n_{samples} , small n_{clusters}	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large n_{samples} and n_{clusters}	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large n_{samples} and n_{clusters}	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large n_{samples} , medium n_{clusters}	Non-flat geometry, uneven cluster sizes	Distances between nearest points
OPTICS	minimum cluster membership	Very large n_{samples} , large n_{clusters}	Non-flat geometry, uneven cluster sizes, variable cluster density	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large n_{clusters} and n_{samples}	Large dataset, outlier removal, data reduction.	Euclidean distance between points

7- Model Selection and Performance

3.1. Cross-validation: evaluating estimator performance

- 3.1.1. Computing cross-validated metrics
 - 3.1.1.1. The `cross_validate` function and multiple metric evaluation
 - 3.1.1.2. Obtaining predictions by cross-validation
- 3.1.2. Cross validation iterators
 - 3.1.2.1. Cross-validation iterators for i.i.d. data
 - 3.1.2.1.1. K-fold
 - 3.1.2.1.2. Repeated K-Fold
 - 3.1.2.1.3. Leave One Out (LOO)
 - 3.1.2.1.4. Leave P Out (LPO)
 - 3.1.2.1.5. Random permutations cross-validation a.k.a. Shuffle & Split
 - 3.1.2.2. Cross-validation iterators with stratification based on class labels.
 - 3.1.2.2.1. Stratified k-fold
 - 3.1.2.2.2. Stratified Shuffle Split
 - 3.1.2.3. Cross-validation iterators for grouped data.
 - 3.1.2.3.1. Group k-fold
 - 3.1.2.3.2. Leave One Group Out
 - 3.1.2.3.3. Leave P Groups Out
 - 3.1.2.3.4. Group Shuffle Split
 - 3.1.2.4. Predefined Fold-Splits / Validation-Sets
 - 3.1.2.5. Cross validation of time series data
 - 3.1.2.5.1. Time Series Split
- 3.1.3. A note on shuffling
- 3.1.4. Cross validation and model selection

7- Model Selection and Performance

3.3. Model evaluation: quantifying the quality of predictions

- 3.3.1. The `scoring` parameter: defining model evaluation rules
 - 3.3.1.1. Common cases: predefined values
 - 3.3.1.2. Defining your scoring strategy from metric functions
 - 3.3.1.3. Implementing your own scoring object
 - 3.3.1.4. Using multiple metric evaluation
- 3.3.2. Classification metrics
 - 3.3.2.1. From binary to multiclass and multilabel
 - 3.3.2.2. Accuracy score
 - 3.3.2.3. Balanced accuracy score
 - 3.3.2.4. Cohen's kappa
 - 3.3.2.5. Confusion matrix
 - 3.3.2.6. Classification report
 - 3.3.2.7. Hamming loss
 - 3.3.2.8. Precision, recall and F-measures
 - 3.3.2.8.1. Binary classification
 - 3.3.2.8.2. Multiclass and multilabel classification
 - 3.3.2.9. Jaccard similarity coefficient score
 - 3.3.2.10. Hinge loss
 - 3.3.2.11. Log loss
 - 3.3.2.12. Matthews correlation coefficient
 - 3.3.2.13. Multi-label confusion matrix
 - 3.3.2.14. Receiver operating characteristic (ROC)
 - 3.3.2.15. Zero one loss
 - 3.3.2.16. Brier score loss
- 3.3.3. Multilabel ranking metrics
 - 3.3.3.1. Coverage error
 - 3.3.3.2. Label ranking average precision
 - 3.3.3.3. Ranking loss
- 3.3.4. Regression metrics
 - 3.3.4.1. Explained variance score
 - 3.3.4.2. Max error
 - 3.3.4.3. Mean absolute error
 - 3.3.4.4. Mean squared error
 - 3.3.4.5. Mean squared logarithmic error
 - 3.3.4.6. Median absolute error
 - 3.3.4.7. R^2 score, the coefficient of determination
- 3.3.5. Clustering metrics
- 3.3.6. Dummy estimators

3.2. Tuning the hyper-parameters of an estimator

- 3.2.1. Exhaustive Grid Search
- 3.2.2. Randomized Parameter Optimization
- 3.2.3. Tips for parameter search
 - 3.2.3.1. Specifying an objective metric
 - 3.2.3.2. Specifying multiple metrics for evaluation
 - 3.2.3.3. Composite estimators and parameter spaces
 - 3.2.3.4. Model selection: development and evaluation
 - 3.2.3.5. Parallelism
 - 3.2.3.6. Robustness to failure
- 3.2.4. Alternatives to brute force parameter search
 - 3.2.4.1. Model specific cross-validation
 - 3.2.4.1.1. `sklearn.linear_model.ElasticNetCV`
 - 3.2.4.1.2. `sklearn.linear_model.LarsCV`
 - 3.2.4.1.3. `sklearn.linear_model.LassoCV`
 - 3.2.4.1.3.1. Examples using `sklearn.linear_model.LassoCV`
 - 3.2.4.1.4. `sklearn.linear_model.LassoLarsCV`
 - 3.2.4.1.4.1. Examples using `sklearn.linear_model.LassoLarsCV`
 - 3.2.4.1.5. `sklearn.linear_model.LogisticRegressionCV`
 - 3.2.4.1.6. `sklearn.linear_model.MultiTaskElasticNetCV`
 - 3.2.4.1.7. `sklearn.linear_model.MultiTaskLassoCV`
 - 3.2.4.1.8. `sklearn.linear_model.OrthogonalMatchingPursuitCV`
 - 3.2.4.1.8.1. Examples using `sklearn.linear_model.OrthogonalMatchingPursuitCV`
 - 3.2.4.1.9. `sklearn.linear_model.RidgeCV`
 - 3.2.4.1.9.1. Examples using `sklearn.linear_model.RidgeCV`
 - 3.2.4.1.10. `sklearn.linear_model.RidgeClassifierCV`
 - 3.2.4.2. Information Criterion
 - 3.2.4.2.1. `sklearn.linear_model.LassoLarsIC`
 - 3.2.4.2.1.1. Examples using `sklearn.linear_model.LassoLarsIC`
 - 3.2.4.3. Out of Bag Estimates
 - 3.2.4.3.1. `sklearn.ensemble.RandomForestClassifier`
 - 3.2.4.3.1.1. Examples using `sklearn.ensemble.RandomForestClassifier`
 - 3.2.4.3.2. `sklearn.ensemble.RandomForestRegressor`
 - 3.2.4.3.2.1. Examples using `sklearn.ensemble.RandomForestRegressor`
 - 3.2.4.3.3. `sklearn.ensemble.ExtraTreesClassifier`
 - 3.2.4.3.3.1. Examples using `sklearn.ensemble.ExtraTreesClassifier`
 - 3.2.4.3.4. `sklearn.ensemble.ExtraTreesRegressor`
 - 3.2.4.3.4.1. Examples using `sklearn.ensemble.ExtraTreesRegressor`
 - 3.2.4.3.5. `sklearn.ensemble.GradientBoostingClassifier`
 - 3.2.4.3.5.1. Examples using `sklearn.ensemble.GradientBoostingClassifier`
 - 3.2.4.3.6. `sklearn.ensemble.GradientBoostingRegressor`
 - 3.2.4.3.6.1. Examples using `sklearn.ensemble.GradientBoostingRegressor`

7- Model Selection and Performance

Clustering Performance Metrics :

1- Adjusted Rand Index

2- Mutual Information Based Scores

3- Homogeneity Completeness and V Score

4- Fowlkes Mallows Score

5- Silhouette Coefficient

6- Calinski-Harabasz Index

7- Davies-Bouldin Index

Computing with Scikit-learn

Computing With Scikit-Learn

1- Scaling with instances using out-of-core learning

2- Incremental Learning

3- Configuring Scikit-learn for reduced validation overhead

```
sklearn.config_context
```

Model Compression

Model Reshaping

Limiting Memory

4- Parallelism, resource management, and configuration



Navigation

[Why joblib: project goals](#)

[Installing joblib](#)

[On demand recomputing:](#)

[the *Memory* class](#)

[Embarrassingly parallel for loops](#)

[Persistence](#)

[Examples](#)

[Development](#)

[joblib.Memory](#)

[joblib.Parallel](#)

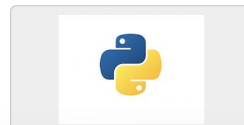
[joblib.dump](#)

[joblib.load](#)

[joblib.hash](#)

[joblib.register_compress](#)

or



Joblib: running Python functions as pipeline jobs

Introduction

Joblib is a set of tools to provide **lightweight pipelining in Python**. In particular:

1. transparent disk-caching of functions and lazy re-evaluation (memoize pattern)
2. easy simple parallel computing

Joblib is optimized to be **fast** and **robust** on large data in particular and has specific optimizations for *numpy* arrays. It is **BSD-licensed**.

Documentation:	https://joblib.readthedocs.io
Download:	https://pypi.python.org/pypi/joblib#downloads
Source code:	https://github.com/joblib/joblib
Report issues:	https://github.com/joblib/joblib/issues

Vision

The vision is to provide tools to easily achieve better performance and reproducibility when working with long running jobs.

- **Avoid computing the same thing twice:** code is often rerun again and again, for instance when prototyping computational-heavy jobs (as in scientific development), but hand-crafted solutions to alleviate this issue are error-prone and often lead to unreproducible results.
- **Persist to disk transparently:** efficiently persisting arbitrary objects containing large data is hard. Using joblib's caching mechanism avoids hand-written persistence and implicitly links the file on disk to the execution context of the original Python object. As a result, joblib's persistence is good for resuming an application status or computational job, eg after a crash.

Other Related Projects

scikit-learn-contrib

scikit-learn-contrib

scikit-learn-contrib is a github organization for gathering high-quality [scikit-learn](#) compatible projects. It also provides a [template](#) for establishing new scikit-learn compatible projects.


Vision

With the explosion of the number of machine learning papers, it becomes increasingly difficult for users and researchers to implement and compare algorithms. Even when authors release their software, it takes time to learn how to use it and how to apply it to one's own purposes. The goal of scikit-learn-contrib is to provide **easy-to-install** and **easy-to-use** high-quality machine learning software. With scikit-learn-contrib, users can install a project by `pip install sklearn-contrib-project-name` and immediately try it on their data with the usual `fit`, `predict` and `transform` methods. In addition, projects are compatible with scikit-learn tools such as grid search, pipelines, etc.

Projects

If you would like to include your own project in scikit-learn-contrib, take a look at the [workflow](#).

lightning
Large-scale linear classification, regression and ranking. Maintained by Mathieu Blondel and Fabian Pedregosa .
py-earth
A Python implementation of Jerome Friedman's Multivariate Adaptive Regression Splines. Maintained by Jason Rudy and Mehdi .
imbalanced-learn
Python module to perform under sampling and over sampling with various techniques. Maintained by Guillaume Lemaitre , Fernando Nogueira , Dayvid Oliveira and Christos Aridas .
polylearn
Factorization machines and polynomial networks for classification and regression in Python. Maintained by Vlad Niculae .
forest-confidence-interval
Confidence intervals for scikit-learn forest algorithms. Maintained by Ariel Rokem , Kivan Polimis and Bryna Hazelton .

hdbscan
A high performance implementation of HDBSCAN clustering. Maintained by Leland McInnes , jc-healy , c-north and Steve Astels .
categorical-encoding
A library of sklearn compatible categorical variable encoders. Maintained by Will McGinnis
boruta_py
Python implementations of the Boruta all-relevant feature selection method. Maintained by Daniel Homola
sklearn-pandas
Pandas integration with sklearn. Maintained by Israel Saeta Pérez
 skope-rules
Machine learning with logical rules in Python. Maintained by Florian Gardin , Ronan Gautier , Nicolas Goix and Jean-Matthieu Schertzer .
stability-selection
A Python implementation of the stability selection feature selection algorithm. Maintained by Thomas Huijskens

Spark scikit-learn

Scikit-learn integration package for Apache Spark

This package contains some tools to integrate the [Spark computing framework](#) with the popular [scikit-learn machine library](#). Among other things, it can:

- train and evaluate multiple scikit-learn models in parallel. It is a distributed analog to the [multicore implementation](#) included by default in `scikit-learn`
- convert Spark's Dataframes seamlessly into numpy `ndarray` or sparse matrices
- (experimental) distribute Scipy's sparse matrices as a dataset of sparse vectors

It focuses on problems that have a small amount of data and that can be run in parallel. For small datasets, it distributes the search for estimator parameters (`GridSearchCV` in scikit-learn), using Spark. For datasets that do not fit in memory, we recommend using the [distributed implementation in `Spark MLlib](#).

This package distributes simple tasks like grid-search cross-validation. It does not distribute individual learning algorithms (unlike Spark MLlib).

Installation

This package is available on PYPI:

```
pip install spark-sklearn
```

This project is also available as [Spark package](#).

The developer version has the following requirements:

- scikit-learn 0.18 or 0.19. Later versions may work, but tests currently are incompatible with 0.20.
- Spark >= 2.1.1. Spark may be downloaded from the [Spark website](#). In order to use this package, you need to use the `pyspark` interpreter or another Spark-compliant python interpreter. See the [Spark guide](#) for more details.
- `nose` (testing dependency only)
- `pandas`, if using the `pandas` integration or testing. `pandas==0.18` has been tested.

DEMO

THANK YOU