Chapter 5
Scikit-Learn

5.1 Introduction

SCIKIT-LEARN is an open source machine learning library written in Python [1]. It allows the easy and fast integration of machine learning methods into PYTHON code. The SCIKIT-LEARN library comprises a wide bandwidth of methods for classification, regression, covariance matrix estimation, dimensionality reduction, data pre-processing, and benchmark problem generation. It can be accessed via the URL http://scikit-learn.org. It is available for various operating systems and is easy to install. The library is steadily improved and extended. SCIKIT-LEARN is widely used in many commercial applications and is also part of many research projects and publications. The SCIKIT-LEARN implementations are the basis of many methods introduced in this book. To improve efficiency, some algorithms are implemented in C and integrated via CYTHON, which is an optimizing static compiler for PYTHON and allows easy extensions in C. The SVM variants are based on LIBSVM and integrated into SCIKIT-LEARN with a CYTHON wrapper. LIBLINEAR is a library containing methods for linear classification. It is used to import linear SVMs and logistic regression.

In Sect. 5.2, we will introduce methods for data management, i.e., loading and generation of benchmark data sets. For demonstration purposes, we will employ the solution candidates generated by an exemplary run of a (1+1)-ES on the constrained benchmark function, i.e., the Tangent problem, see Appendix A. The use of supervised learning methods like nearest neighbors with model training and prediction is introduced in Sect. 5.3. Pre-processing methods like scaling, normalization, imputation, and feature selection are presented in Sect. 5.4, while model evaluation methods are sketched in Sect. 5.5. Section 5.6 introduces cross-validation methods. Unsupervised learning is presented in Sect. 5.7, and conclusions are finally drawn in Sect. 5.8.

O. Kramer, *Machine Learning for Evolution Strategies*,
Studies in Big Data 20, DOI 10.1007/978-3-319-33383-0_5

5.2 Data Management

An important part of SCIKIT-LEARN is data management, which allows loading typical machine learning data sets, e.g., from the UCI machine learning repository. An example is the Iris data set that contains 150 4-dimensional patterns with three classes that are types of Iris flowers. Data set modules are imported via `from sklearn import datasets`, while `iris = datasets.load_iris()` loads the Iris data set.

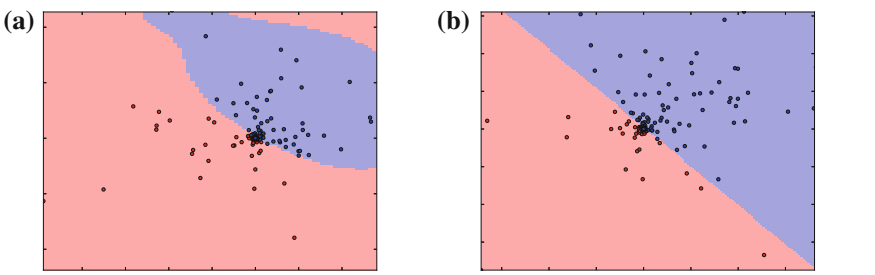
Another example is the California housing data set. The data set is represented by a dictionary-like data structure with the attributes `dataset.data`, which is a 20,640 times 8-shape `numpy` array. The array `dataset.target` contains the labels corresponding to the average house value in units of 100000. Numerous further data sets are available. The list is continuously extended.

For a convenient way of splitting the data set into training and test set, the method `train_test_split` from `sklearn.cross_validation` can be used. With parameter `test_size` the fraction of the test set size can be specified with a value between 0 and 1. For example, for a 80/20 split of training and test data, the setting `test_size = 0.2` must be used.

Further, SCIKIT-LEARN employs methods that allow generating data sets, so called sample generators. There exist regression and classification data set generators that allow the specification of interesting data set properties like sparsity, etc. Examples are `make_classification`, `make_circle`, `make_regression`, and `make_hastie`. For example, the latter generates a data set for binary classification. The output consists of 10 features with standard independent Gaussian labels, i.e.,

```
y[i] = 1 if np.sum(X[i] ** 2) > 9.34 else -1.
```

In Fig. 5.1, we will analyze a data set that has been generated during optimization of a (1+1)-ES on the Tangent problem with $N = 10$.



SVM on Tangent problem, RBF-kernel, standard settings, $C = 10^3$

SVM on Tangent problem tuned with cross-validation, resulting in RBF-kernel, $C = 10^7$, and $\gamma = 10^{-3}$

Fig. 5.1 SVM on the Tangent problem for $d = 2$ dimensions **a** with standard settings and **b** with optimized settings determined by cross-validation

5.3 Supervised Learning

In Chap. 4, the basis of machine learning is introduced. In Chap. 6, we will employ the first supervised learning algorithm, i.e., nearest neighbor regression. Here, we will show how to use the kNN implementation of SCIKIT-LEARN. Training, fitting, and prediction with the nearest neighbor method kNN works as follows.

1. The command `from sklearn import neighbors` imports the nearest neighbor package of SCIKIT-LEARN that contains the kNN classifier.
2. `clf = neighbors.KNeighborsClassifier(n_neighbors = 2, algorithm = 'ball_tree')` instantiates a kNN model with neighborhood size $k = 2$ employing a ball tree. Ball trees allows efficient neighborhood requests in $O(\log N)$ for low-dimensional data sets.
3. `clf.fit(X, y)` trains the classifier with patterns collected in `X` (which can be a list or a NUMPY array) and corresponding labels in `y`.
4. `clf.predict(X_)` finally applies to model to the patterns in `X_` yielding a corresponding list of labels.

Other classification and regression methods are used similarly, i.e., with the methods `fit` and `predict`. A famous method for supervised learning with continuous labels is linear regression that fits a linear model to the data. The command `regr = linear_model.LinearRegression()` creates a linear regression object. The method `regr.fit(X, y)` trains the linear regression model with training patterns `X` and labels `y`. Variants of linear models exist like ridge regression (`linear_model.Ridge`) and kernel ridge regression (`linear_model.KernelRidge`). A further prominent method is Lasso (`linear_model.Lasso`) that improves conditioning of the data by mitigating the curse of dimensionality. It selects only the informative features and sets the non-informative ones to zero. Lasso penalizes coefficients with L1 prior as regularizer. Decision trees are very successful methods and also part of the SCIKIT-LEARN library. With `tree.DecisionTreeClassifier()` a decision tree is available.

For support vector classification, the class `sklearn.svm.SVC` is implemented. It is based on the libraries LIBSVM and LIBLINEAR with the following interesting parameters. For example, parameter `C` is the regularization parameter that controls the penalty term of the SVM. With `kernel`, a kernel function can be specified. A default choice is a radial-basis function (RBF) kernel. In case of polynomial kernels, `degree` specifies its degree. Due to numerical reasons, the SVM training process may get stuck. Parameter `max_iter` allows the specification of a maximum number of training steps of the optimizer and forces the training to terminate. The standard setting (`max_iter = -1`) does not set a limit on the number of optimization steps.

Figure 5.1 shows the results when employing SVMs for classification of the feasibility of the constrained solution space when optimizing the Tangent problem with the (1+1)-ES as introduced in Chap. 2 for 200 generations.

5.4 Pre-processing Methods

Pre-processing of patterns is often required to allow successful learning. An essential part is scaling. For the successful application of many methods, data must look standard normally distributed with zero mean and unit variance. The pre-processing library is imported with the expression `from sklearn import preprocessing`. A numpy array `X` is scaled with

```
X_scaled = preprocessing.scale(X).
```

The result has zero mean and unit variance. Another scaling variant is the min-max scaler. Instantiated with

```
min_max_scaler = preprocessing.MinMaxScaler(),
```

the data can be scaled to values between 0 and 1 by

```
X_minmax = min_max_scaler.fit_transform(X)
```

for a training set `X`. Normalization is a variant that maps patterns so that they have unit norm. The assumption is useful for text classification and clustering and also supports pipelining. Feature transformation is an important issue. A useful technique is the binarization of features. Instantiated with

```
binarizer = preprocessing.Binarizer().fit(X),
```

the binarizer's method `binarizer.transform(X)` allows thresholding numerical features to get boolean features. This may be useful for many domains, e.g., text classification.

Besides binary or numerical features, we can have categorical features. SCIKIT-LEARN features the one-hot encoder available via `preprocessing.OneHotEncoder()`. It encodes categorical integer features employing a one-of-K scheme, i.e., outputs a sparse matrix, where each column corresponds to a value of a feature. An input matrix of integers is required within a range based on the number of features.

A further class of pre-processing methods cares for imputation. In practical machine learning problems, patterns are often incomplete. Methods can only cope with complete patterns. If the removal of incomplete patterns is not possible, e.g., due small training sets, missing elements have to be filled with appropriate values. This is also possible with regression methods. SCIKIT-LEARN features the imputer that completes missing data (`preprocessing.Imputer`). A further feature transformation method is the method for polynomial features. It adds complexity to the model by considering nonlinear features similar to kernel functions.

Feature selection methods can also be seen as a kind of pre-processing step. The task is to select a subset of features to improve the classifier accuracy. Feature selection will also improve the speed of most classifiers. The method `feature_selection.SelectKBest` selects the `K` best features w.r.t. measures like `f_classif` for the ANOVA F-value between labels for classification tasks and `chi2` for Chi-squared statistics of non-negative features. Further measures are available. An interesting methods for feature selection is recursive feature elimination (`feature_selection.RFE`). It step by step removes features that are assigned by a classifier with a minimal weight. This process is continued until the desired number is reached. Besides feature selection, the concatenation of features is a useful method and easily possible via `feature_selection.FeatureUnion(X1, X2)`, where `X1` and `X2` are patterns.

SCIKIT-LEARN allows pipelining, i.e., the combination of multiple methods successively. An exemplary pipeline combines normalization and classification. With `pipeline.Pipeline` the pipeline object is loaded. For example, for instantiated components `normalization` and `KNNClassifier` the command

```
pipe = Pipeline(steps = [('norm',normalization),('KNN',
KNNClassifier)])
```

implements a pipeline with two steps. To be applicable to be a pipeline member in the middle of a pipeline, the method must implement the `transform` command. Pipelines are efficient expression for such successive pattern-wise processing steps.

5.5 Model Evaluation

Model evaluation has an important part to play in machine learning. The quality measure depends on the problem class. In classification, the precision score is a reasonable measure. It is the ratio of true positives (correct classification for pattern of class *positive*) and all positives (i.e., the sum of true positive and false positives). Intuitively, precision is the ability of a classifier not to label a negative pattern as positive. The precision score is available via `metrics.precision_score` with variants that globally count the total number of true positives, false negatives, and false positives (`average = 'micro'`). A further variant computes the matrix for each label with unweighted mean (`average = 'macro'`), or weighed by the support mean (`average = 'weighted'`) for taking into account imbalanced data.

Another important measure is recall (`metrics.recall_score`) that is defined as the ratio between the true positives and the sum of the true positives and the false negatives. Intuitively, recall is a measure for the ability of the classifier to find all the positive samples. A combination of the precision score and recall is

Table 5.1 Classification report for constraints

Domain	Precision	Recall	F1	Support
0	0.92	1.00	0.96	68
1	1.00	0.14	0.25	7
Avg/total	0.93	0.92	0.89	75

the F1 score (`metrics.f1_score`) that is a weighted average of both measures. All three measures can be generated at once with the classification report (`metrics.classification_report`). An example is presented in Table 5.1, which shows the corresponding measures for a data set generated by a (1+1)-ES, which runs on the Tangent problem. The (1+1)-ES uses Gaussian mutation and Rechenberg's step size control. The optimization runs results in a 10-dimensional data set comprising 200 patterns. The `classification_report` will also be used in Chap. 7. For regression, there are also metrics available like the R2 score (`metrics.r2_score`), whose best value is 1.0 and that can also return negative values for bad regression methods.

5.6 Model Selection

In Chap. 4, the importance of model selection has been emphasized. In SCIKIT-LEARN, numerous methods are implemented for model selection. Cross-validation is a method to avoid overfitting and underfitting. It is combined with grid search or other optimization methods for tuning the models' parameters. For many supervised methods, grid search is sufficient, e.g., for the hyper-parameters of SVMs. Among the cross-validation methods implemented in SCIKIT-LEARN is n-fold cross-validation (`cross_validation.kFold`), which splits the data set into n folds, trains on $n-1$ folds and tests on the left-out fold. As a variant, stratified n-fold cross-validation (`cross_validation.StratifiedKFold`) preserves the class ratios within each fold. If few training patterns are available, leave-one-out cross-validation should be used, available as `cross_validation.LeaveOneOut`. All cross-validation variants employ the parameter `shuffle` that is a boolean variable stating, if the data set should be shuffled (`shuffle = True`) or not `shuffle = False` before splitting the data set into folds. A useful object in SCIKIT-LEARN is the grid search object `GridSearchCV` that implements an n-fold cross-validation and gets a classifier, a dictionary of test parameters, the number of folds and a quality measure as arguments. It performs the cross-validation with the employed method and returns a classifier that achieves an optimal score as well as scores for all parameter combinations.

5.7 Unsupervised Learning

Unsupervised learning is learning without label information. Various methods for unsupervised learning are part of SCIKIT-LEARN. One important unsupervised problem is clustering, which is the task of grouping data sets w.r.t. their intrinsic properties. It has numerous applications. A famous clustering method is k-means, which is also implemented in SCIKIT-LEARN. Given the number k of clusters, k-means iteratively places the k clusters in data space by successively assigning all patterns to the closest cluster center and computing the mean of these clusters. With `cluster.KMeans`, k-means can be applied stating the desired number of cluster centers k. Fitting k-means to a set of patterns and getting the cluster labels is possible with `y = KMeans(n_clusters=2).fit_predict(X)`. To estimate an appropriate number k, i.e., `n_clusters`, the percentage of inner cluster variance can be plotted for different k. It is recommendable to choose k from the area of the largest change of slope of this variance curve. Various other methods for clustering are available in SCIKIT-LEARN like DBSCAN. DBSCAN, which can be accessed with `cluster.DBSCAN`, determines core samples of high density defined via the number `min_samples` of neighboring patterns within a radius `eps`. It expands clusters from the core samples to find further corse samples. Corner samples have less patterns than `min_samples` in their radius, but are located in the radius of core samples. The method is used with command `DBSCAN(eps = 0.3, min_samples = 10).fit(X)`. DBSCAN will be introduced in Chap. 10, where it is used for clustering-based niching.

To give a further demonstration of PYTHON code, the following steps show the generation of a list of clusters, which contain their assigned patterns.

```
label_list = k_means.fit_predict(X)
labels = list(set(labellist))
clusters = [[] for i in range(len(labels))]
for i in xrange(len(X)):
>>> clusters[labellist[i]].append(np.array(X[i]))
```

First, the list of labels are accessed from the trained k-means method. With the `set`-method, this list is cast to a set that contains each label only once. The third step generates a list of empty cluster lists, which is filled with the corresponding patterns in the `for`-loop.

The second important class of unsupervised learning is dimensionality reduction. PCA is a prominent example and very appropriate to linear data. In SCIKIT-LEARN, PCA is implemented with singular value decomposition from the linear algebra package `scipy.linalg`. It keeps only the most significant singular vectors for the projection of the data to the low-dimensional space. Available in `decomposition.PCA` with a specification of the target dimensionality `PCA(n_components = 2)`, again the method `fit(X)` fits the PCA to the data, while `transform(X)` delivers the low-dimensional points. A combination of both

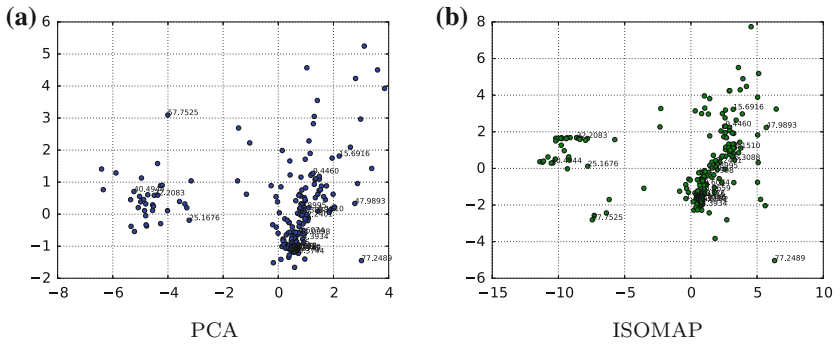


Fig. 5.2 **a** PCA embedding of 10-dimensional patterns of a (1+1)-ES for 200 generations. **b** ISOMAP embedding of the corresponding data set with $k = 10$

methods is `fit_transform(X)`, which directly delivers the low-dimensional pendants of the high-dimensional patterns. With `sklearn.manifold`, various manifold learning methods for non-linear data are available. For example, ISOMAP can be accessed via `manifold.Isomap` with parameters `n_neighbors` specifying the number of neighbors employed for ISOMAP's neighborhood graph. The target dimensionality can again be set with `n_components`. With `fit_transform(X)`, ISOMAP is fit to the data `X`, and the low-dimensional representations are computed.

Figure 5.2 shows an application of PCA and ISOMAP optimizing the Tangent problem. The corresponding data set is generated as follows. The (1+1)-ES runs for 200 generations resulting in 200 patterns. For dimensionality reduction, no labels are employed, i.e., the fitness values are ignored. PCA and ISOMAP map into a two-dimensional space for visualization. ISOMAP uses the neighborhood size $k = 10$. Both figures show that the optimization process converges towards a region in solution space.

5.8 Conclusions

PYTHON is a modern programming language that allows fast prototyping of methods. It is a universal programming language, which is in most cases interpreted and not compiled. PYTHON supports various programming paradigm like functional and objective programming. PYTHON code is usually easy to read and shorter than code in other high-level programming languages. This is achieved with a reduced number of code words and a reduced syntax that allows short expressions. PYTHON supports dynamic typing of variables. Numerous libraries make PYTHON an attractive programming language for a broad field of applications and domains.

SCIKIT-LEARN is a strong machine learning library written in PYTHON that developed fast in the past years. It allows an easy use of known machine learning concepts employing various important algorithms for pre-processing, feature selection, pipelining, model evaluation, and many other tasks. Numerous methods for supervised and unsupervised learning are implemented. This list of supported methods comprises nearest neighbor methods, SVMs, support vector regression (SVR), decision trees, random forests. Also unsupervised methods are available like k-means and DBSCAN for clustering and PCA, ISOMAP, and LEE for dimensionality reduction.

This chapter gives a short introduction and comprehensive overview over most of the concepts that are supported. SCIKIT-LEARN advertises that it is used by numerous companies and research institutes, e.g., Infria, Evernote, Spotify, and DataRobot. Further, SCIKIT-LEARN offers a very comprehensive documentation of the whole library and an excellent support by the developer and research community. In the course of this book, various techniques that are part of SCIKIT-LEARN are combined with the (1+1)-ES. Each chapter will give a short introduction to the relevant SCIKIT-LEARN method.

Reference

1. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)