Scikit-Learn Cheat Sheet (2021), Python for Data Science

The absolute basics for beginners learning Scikit-Learn in 2021

Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression, clustering algorithms, and efficient tools for data mining and data analysis. It's built on NumPy, SciPy, and Matplotlib.

Sections:

Basic Example:

The code below demonstrates the basic steps of using scikit-learn to create and run a model on a set of data.

The steps in the code include: loading the data, splitting into train and test sets, scaling the sets, creating the model, fitting the model on the data, using the trained model to make predictions on the test set, and finally evaluating the performance of the model.

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X,y = iris.data[:,:2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X,y)
>>> scaler = preprocessing_StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors = 5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading the Data

Your data needs to be numeric and stored as NumPy arrays or SciPy spare matrix. Other types that convert to numeric arrays, such as Pandas DataFrame's are also acceptable.

```
>>> import numpy as np>>> X =
np.random.random((10,5))array([[0.21069686, 0.33457064],
       [0.23887117, 0.6093155 ],
       [0.48848537, 0.62649292]])>>> y =
np.array(['A','B','A'])array(['A', 'B', 'A'])
```

Training and Test Data

Splits the dataset into training and test sets for both the X and y variables.

```
>>> from sklearn.model_selection import train_test_split
>>> X_train,X_test,y_train,y_test = train_test_split(X,y,
random_state = 0)
```

Preprocessing The Data

Getting the data ready before the model is fitted.

Standardization

Standardize's the features by removing the mean and scaling to unit variance.

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standarized_X = scaler.transform(X_train)
>>> standarized_X_test = scaler.transform(X_test)
```

Normalization

Each sample (i.e. each row of the data matrix) with at least one non-zero component is rescaled independently of other samples so that its norm equals one.

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

Binarize data (set feature values to 0 or 1) according to a threshold.

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold = 0.0).fit(X)
>>> binary_X = binarizer.transform(X_test)
```

Encoding Categorical Features

Encode’s target labels with values between 0 and n_classes-1.

```
>>> from sklearn import preprocessing
>>> le = preprocessing.LabelEncoder()
>>> le.fit_transform(X_train)
```

Imputing Missing Values

Imputation transformer for completing missing values.

```
>>> from sklearn.impute import SimpleImputer
>>> imp = SimpleImputer(missing_values = 0, strategy = 'mean')
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

Generate a new feature matrix consisting of all polynomial combinations of the features with degrees less than or equal to the specified degree.

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Create Your Model

Creation of various supervised and unsupervised learning models.

Supervised Learning Models

- Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr  = LinearRegression(normalize = True)
```

- Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel = 'linear')
```

- Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

- KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors = 5)
```

Unsupervised Learning Models

- Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components = 0.95)
```

- K means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters = 3, random_state = 0)
```

Model Fitting

Fitting supervised and unsupervised learning models onto data.

Supervised learning

- Fit the model to the data

```
>>> lr.fit(X, y)
>>> knn.fit(X_train,y_train)
>>> svc.fit(X_train,y_train)
```

Unsupervised learning

- Fit the model to the data

```
>>> k_means.fit(X_train)
```

- Fit to data, then transform it

```
>>> pca_model = pca.fit_transform(X_train)
```

Prediction

Predicting test sets using trained models.

- Predict labels

```
#Supervised Estimators
>>> y_pred = lr.predict(X_test)#Unsupervised Estimators
>>> y_pred = k_means.predict(X_test)
```

- Estimate probability of a label

```
>>> y_pred = knn.predict_proba(X_test)
```

Evaluate Your Model’s Performance

Various regression and classification metrics that determine how well a model performed on a test set.

Classification Metrics

- Accuracy Score

```
>>> knn.score(X_test,y_test)
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test,y_pred)
```

- Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test,y_pred))
```

- Confusion Matrix

```
>>> from sklearn .metrics import confusion_matrix
>>> print(confusion_matrix(y_test,y_pred))
```

Regression Metrics

- Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> mean_absolute_error(y_test,y_pred)
```

- Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test,y_pred)
```

- R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_test, y_pred)
```

Clustering Metrics

- Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_test,y_pred)
```

- Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_test,y_pred)
```

- V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> v_measure_score(y_test,y_pred)
```

Cross-Validation

- Evaluate a score by cross-validation

```
>>> from sklearn.model_selection import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
```

Tune Your Model

Finding correct parameter values that will maximize a model’s prediction accuracy.

Grid Search

Exhaustive search over specified parameter values for an estimator. The example below attempts to find the right amount of clusters to specify for knn to maximize the model’s accuracy.

```
>>> from sklearn.model_selection import GridSearchCV
>>> params = {'n_neighbors': np.arange(1,3),
'metric':['euclidean','cityblock']}
>>> grid = GridSearchCV(estimator = knn, param_grid = params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

Randomized search on hyperparameters. In contrast to Grid Search, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions. The number of parameter settings that are tried is given by n_iter.

```
>>> from sklearn.model_selection import RandomizedSearchCV
>>> params = {'n_neighbors':range(1,5),
'weights':['uniform','distance']}
>>> rsearch = RandomizedSearchCV(estimator = knn,
param_distributions = params, cv = 4, n_iter = 8, random_state = 5)
>>> rseach.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Scikit-learn is an extremely useful library for various machine learning models. The sections above provided a basic step-by-step process to perform an analysis on different models. If you want to learn more however check out the documentation for Scikit-Learn, as there are still plenty of useful functions you could learn.