# **Python For Data Science** *Cheat Sheet*

# Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com



### Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.



### A Basic Example

```
>>> 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, random_state=33)
>>> 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**

#### Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> X[X < 0.7] = 0
```

# **Training And Test Data**

```
>>> from sklearn.model_selection import train_test_split
>>> X train, X test, y train, y test = train test split(X,
                                                  random state=0)
```

## **Create Your Model**

### Supervised Learning Estimators

### 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 Baves

>>> from sklearn.naive bayes import GaussianNB

### >>> gnb = GaussianNB() KNN

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

### Unsupervised Learning Estimators

### 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**

### Supervised learning

>>> lr.fit(X, y) >>> knn.fit(X train, y train) >>> svc.fit(X train, y train)

### Unsupervised Learning

>>> k means.fit(X train)

>>> pca model = pca.fit transform(X train) | Fit to data, then transform it

### Fit the model to the data

Fit the model to the data

# **Prediction**

### **Supervised Estimators**

>>> y pred = svc.predict(np.random.random((2,5))) >>> y pred = lr.predict(X test)

#### >>> y pred = knn.predict proba(X test) Unsupervised Estimators

>>> y pred = k means.predict(X test)

#### Predict labels Predict labels Estimate probability of a label

Predict labels in clustering algos

# **Preprocessing The Data**

#### Standardization

- >>> from sklearn.preprocessing import StandardScaler
- >>> scaler = StandardScaler().fit(X train)
- >>> standardized X = scaler.transform(X train) >>> standardized X test = scaler.transform(X test)

### Normalization

- >>> from sklearn.preprocessing import Normalizer >>> scaler = Normalizer().fit(X train) >>> normalized X = scaler.transform(X train)
- >>> normalized X test = scaler.transform(X test)

#### Binarization

- >>> from sklearn.preprocessing import Binarizer >>> binarizer = Binarizer(threshold=0.0).fit(X)
- >>> binary X = binarizer.transform(X)

# **Encoding Categorical Features**

- >>> from sklearn.preprocessing import LabelEncoder
- >>> enc = LabelEncoder()
- >>> y = enc.fit transform(y)

### Imputing Missing Values

- >>> from sklearn.preprocessing import Imputer
- >>> imp = Imputer(missing values=0, strategy='mean', axis=0) >>> imp.fit transform(X train)

# Generating Polynomial Features

- >>> from sklearn.preprocessing import PolynomialFeatures
- >>> poly = PolynomialFeatures(5) >>> poly.fit transform(X)

### **Accuracy Score**

>>> knn.score(X test, y test)

**Classification Metrics** 

Estimator score method

**Evaluate Your Model's Performance** 

>>> from sklearn.metrics import accuracy score Metric scoring functions >>> accuracy score(y test, y pred)

#### Classification Report

>>> from sklearn.metrics import classification report Precision, recall, fi-score >>> print(classification report(y test, y pred)) and support

#### 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 >>> y true = [3, -0.5, 2]
- >>> mean\_absolute\_error(y\_true, y\_pred)

# Mean Squared Error

- >>> from sklearn.metrics import mean squared error
- >>> mean squared error(y test, y pred)

>>> from sklearn.metrics import r2 score >>> r2 score(y true, y\_pred)

# Clustering Metrics

### **Adjusted Rand Index**

- >>> from sklearn.metrics import adjusted rand score >>> adjusted rand score(y true, y pred)
- Homogeneity
- >>> from sklearn.metrics import homogeneity score
- >>> homogeneity score(y true, y pred)

#### V-measure

>>> from sklearn.metrics import v measure score >>> metrics.v measure score(y true, y pred)

### **Cross-Validation**

- >>> from sklearn.cross validation import cross val score
- >>> print(cross val score(knn, X train, y train, cv=4)) >>> print(cross val score(lr, X, y, cv=2))

# **Tune Your Model**

### **Grid Search**

- >>> from sklearn.grid search 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

- >>> from sklearn.grid search import RandomizedSearchCV >>> params = {"n neighbors": range(1,5),
- n iter=8, random state=5)
- >>> rsearch.fit(X train, y train) >>> print(rsearch.best score )

#### **DataCamp Learn Python for Data Science Interactively**

