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.

### Preprocessing The Data

#### Standardization

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train)
standardized_X = scaler.transform(X_train)
```

#### Normalization

```python
from sklearn.preprocessing import Normalizer
normalizer = Normalizer().fit(X_train)
normalized_X = normalizer.transform(X_train)
```

#### Binarization

```python
from sklearn.preprocessing import Binarizer
binaryizer = Binarizer(threshold=0.5).fit(X_train)
binary_X = binaryizer.transform(X_train)
```

### Generating Polynomial Features

```python
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
poly_X = poly.fit_transform(X)
```

### Create Your Model

#### Supervised Learning Estimators

**Linear Regression**

```python
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg.fit(X, y)
```

**Support Vector Machines (SVM)**

```python
from sklearn.svm import SVC
svm = SVC(kernel='linear')
svm.fit(X, y)
```

**Naive Bayes**

```python
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
bayes.fit(X, y)
```

### Model Fitting

```python
model.fit(X_train, y_train)
model.predict(X_test)
```

### Prediction

```python
y_pred = model.predict(X_test)
```

### Evaluate Your Model’s Performance

#### Classification Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>accuracy_score(y_true, y_pred)</td>
</tr>
<tr>
<td>Precision</td>
<td>precision_score(y_true, y_pred)</td>
</tr>
<tr>
<td>Recall</td>
<td>recall_score(y_true, y_pred)</td>
</tr>
<tr>
<td>F1 Score</td>
<td>f1_score(y_true, y_pred)</td>
</tr>
</tbody>
</table>

#### Regression Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Absolute Error</td>
<td>mean_absolute_error(y_true, y_pred)</td>
</tr>
<tr>
<td>Mean Squared Error</td>
<td>mean_squared_error(y_true, y_pred)</td>
</tr>
<tr>
<td>R2 Score</td>
<td>r2_score(y_true, y_pred)</td>
</tr>
</tbody>
</table>

#### Clustering Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homogeneity</td>
<td>homogeneity_score(y_true, y_pred)</td>
</tr>
<tr>
<td>V-Measure</td>
<td>v_measure(y_true, y_pred)</td>
</tr>
</tbody>
</table>

### Cross-Validation

```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=5)
```

### Tune Your Model

#### Grid Search

```python
from sklearn.model_selection import GridSearchCV
gscv = GridSearchCV(estimator=model, param_grid=params, cv=5)
gscv.fit(X, y)
gscv.best_estimator_  # the best model
```

#### Randomized Parameter Optimization

```python
from sklearn.model_selection import RandomizedSearchCV
rsrv = RandomizedSearchCV(estimator=model, param_distributions=params, cv=5)
rsv.fit(X, y)
rsv.best_estimator_  # the best model
```

---

**Training And Test Data**

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

**Creating Model**

```python
reg = LinearRegression()
reg.fit(X_train, y_train)
reg.score(X_test, y_test)
```

**Predicting**

```python
y_pred = reg.predict(X_test)
```

**Scikit-learn Cheat Sheet**

- **Python For Data Science**
- **Learn Scikit-Learn online at DataCamp**

---

**Loading The Data**

- `X` **dataframe with features**
- `y` **target variable**

**Supervised Learning**

- **Linear Regression**
- **Support Vector Machines (SVM)**
- **Naive Bayes**

**Unsupervised Learning**

- **Principal Component Analysis (PCA)**

**Cross-Validation**

- **Grid Search**
- **Randomized Parameter Optimization**

---

**A Basic Example**

```python
# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Standardize the features
scaler = StandardScaler().fit(X)
X = scaler.transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

# Create the model
knn = KNeighborsClassifier(n_neighbors=5)

# Fit the model to the data
knn.fit(X_train, y_train)

# Predict the labels
y_pred = knn.predict(X_test)
```

---

**Recommendations**

- Use `StandardScaler` for standardizing the features.
- Use `train_test_split` to create training and testing sets.
- Use `KNeighborsClassifier` for a simple classification model.
- Explore `GridSearchCV` for hyperparameter tuning.

---

**Also see**

- NumPy
- Pandas

---

**Learn Data Skills Online at DataCamp.com**