Scikit-Learn Cheat Sheet

MLtrends

Workflow of a basic example

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

iris = datasets.load_iris() #1.Preprocessing
X, y = iris.data[:,:3], iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
X_train_std = StandardScaler().fit_transform(X_train)
X_test_std = StandardScaler().fit_transform(X_test)
knc = KNeighborsClassifier(n_neighbors=3) #2.Model creation
knc.fit(X_train_std,y_train) #3.Model fitting
y_pred = knc.predict(X_test_std) #4.Prediction
accuracy_score(y_test, y_pred) #5.Performance evaluation
```

1. Data Preprocessing

Loading data

```
# Import data
from sklearn import datasets
iris = datasets.load_iris()
X, y = iris.data[:,:4], iris.target
```

Training and test data

```
# Split training and test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
```

Data preparation

```
# Standardization (z score = (x – μ) / σ )
scaler = StandardScaler().fit(X_train)
X_train_standardized = scaler.transform(X_train)
X_test_standardized = scaler.transform(X_test)
```

```
# Normalization ((X - X_min)/(X_max - X_min))
from sklearn.preprocessing import Normalizer
scaler = Normalizer().fit(X_train)
X_train_normalized = scaler.transform(X_train)
X_test_normalized = scaler.transform(X_test)
```

```
# Binarization (numerical features to boolean values)
from sklearn.preprocessing import Binarizer
binarizer = Binarizer(threshold = 3).fit(X_train)
X_binarized = binarizer.transform(X_test)
```

```
# Imputation of Missing Values
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp.fit_transform(X_train)
```

```
# Generating polynomial features
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(2)
X_poly = poly.fit_transform(X_train)
```

```
# Custom transformers
from sklearn.preprocessing import FunctionTransformer
transformer = FunctionTransformer(np.log1p, validate=True)
X_transformed = transformer.transform(X_train)
```

```
# Binning
import numpy as np
from sklearn.preprocessing import KBinsDiscretizer
k_bins = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='kmeans').fit(X)
est = k_bins.transform(X_train)
```

```
# One-Hot Encoder
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
enc.fit(y.reshape(-1,1))
enc.transform(y.reshape(-1,1)).toarray()
```

```
# Label Encoder
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
y_encoded = enc.fit_transform(y)
```

2. Model Creation

Supervised learning - regression

```
# Supervised Learning Estimators - regression
# 1. Linear Regression
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
# 2. K Nearest Neighbor
from sklearn.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor(n_neighbors=2)
```

Supervised learning - classification

```
# Supervised Learning Estimators - classification
# 1. Support Vector Machines (SVM)
from sklearn.svm import SVC
svc = SVC()
# 2. Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
# 3. K Nearest Neighbor
from sklearn.neighbors import KNeighborsClassifier
knc = KNeighborsClassifier(n_neighbors=3)
```

```
# Unsupervised Learning Estimators
# K means
from sklearn.cluster import KMeans
k_means = KMeans(n_clusters = 3, random_state= 0)
# PCA
# Reduce number of attributes, while presrving as much info as possible
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X_train)
```

3. Model Fitting

```
# Model fitting
# Supervised learning
clf = svc.fit(X_train, y_train) # Fit the model to data
clf = knc.fit(X_train, y_train) # Fit the model to data
clf = gnb.fit(X_train, y_train) # Fit the model to data
# Unsupervised learning
reg = k_means.fit(X_train)
pca_model = pca.fit_transform(X_train)  # Fit to the data, and trannsform it
```

4. Prediction

```
# Prediction
# Supervised learning
y_pred = svc.predict(X_test) # Predict labels
y_pred = lr.predict(X_test) # Predict labels
y_pred = knc.predict(X_test) # Predict labels
# Unsupervised learning
y_pred = k_means.predict(X_test) # Predict labels clustering
```

5. Performance Evaluation

Classification Metrics

```
# Classifcation Metrics
# 1. Accuracy score
from sklearn.metrics import accuracy_score
knc.score(X_test,y_test)
accuracy_score(y_test,y_pred)
# 2. Classfication Report
from sklearn.metrics import classification_report
classification_report(y_test,y_pred)
# 3. Confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Cross Validation

```
# Cross-validation
from sklearn.model_selection import cross_val_score
clf = svc.fit(X_train, y_train)
scores = cross_val_score(clf, X, y, cv=5, scoring='f1_macro')
```

Regression Metrics

```
# Regression Metrics
# 1. Mean Absolute Error
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test, y_pred)
# 2.Mean Squared Error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
# 3. R^2 score
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

Clustering Metrics

```
# Clustering Metrics
# Adjusted Rand Index
from sklearn.metrics import adjusted_rand_score
adjusted_rand_score(y, y_pred)
# Homogeneity
from sklearn.metrics import homogeneity_score
homogeneity_score(y_pred, y_pred)
# V-measure
from sklearn.metrics import v_measure_score
v_measure_score(y, y_pred)
```

Model Tuning

Grid Search

```
# Grid search
from sklearn.model_selection import GridSearchCV
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
clf = GridSearchCV(svc, parameters)
clf.fit(X_train, y_train)
# To check the results
clf.cv_results_
```

Randomized Search

```
# Randomized Parameter Optimization
from sklearn.model_selection import RandomizedSearchCV
params ={'n_neighbors': [2,3,4], 'weights':['uniform','distance']}
clf = RandomizedSearchCV(estimator=knc,
                         param_distributions=params,
                         cv=4,
                         n_iter=8,
                         random_state=5)
clf.fit(X_train, y_train)
clf.cv_results_
```

Pipeline

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
# Load data
iris = datasets.load_iris()
X, y = iris.data, iris.target
# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
# Making the Pipeline: PCA -> Scaling the data -> Classfication
pipe = Pipeline([('pca', PCA()),
                 ('scaler', StandardScaler()),
                 ('classifer', DecisionTreeClassifier())])
# Fitting the model
parameters = {'pca__n_components': [2, 3, 4],
              'classifer__max_depth': [5, 10, 20]}
grid = GridSearchCV(pipe, parameters).fit(X_train, y_train)
# Stores the optimum model in best_pipe
best_pipe = grid.best_estimator_
print(best_pipe)
print('Test set score: ' + str(best_pipe.score(X_test,y_test)))
```

Scikit-Learn Cheat Sheet

MLtrends