Scikit-Learn cheat Sheet

Sklearn is a free machine learning library for Python. It features various classification, regression and clustering algorithms.

Basic

The code below demonstrates the basic steps of using sklearn 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

The data needs to be numeric and stored as Numpy arrays or Scipy spare matrix (numeric arrays, such as Pandas Data Frame's are also ok)

```
>>> import numpy as np
>>> X = np.random.random((10,5))
array([[0.21,0.33],
       [0.23,0.60],
       [0.48,0.62]])
>>> y = np.array(['A','B','A'])
array(['A','B','A'])
```

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)# Splits data into training and test set
```

Preprocessing The Data

Standardization

Standardizes 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 (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

Imputation transformer for completing missing values.

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

Imputing Missing Values

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

Generating Polynomial Features

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

Create your Model

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_cluster =3, random_state = 0)
```

Model Fitting

Fitting supervised and unsupervised learning models onto data.

Supervised Learning

```
lr.fit(x,y) # Fit the model to the data
knn.fit(x_train, y_train)
svc.fit(x_train, y_train)
```

Unsupervised Learning

```
k_means.fit(x_train) # Fit the model to the data
pca_model = pca.fit_transform(x_train) # Fit to data, then transform
```

Prediction

Predict Labels

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

Estimate probability of a label

```
y_pred = knn.predict_proba(x_test)
```

Evaluate your Model's Performance

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)
```

Tune Your Model

Grid Search

```
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)
```