New Package Introduction - Unsupervised Learning: Scikit-Learn

Scikit-Learn

Scikit-learn is an open-source Python library that is built upon NumPy, SciPy, and Matplotlib which is typically used in machine learning projects. Scikit-learn is focused on machine learning tools including mathematical, statistical, and general purpose algorithms that form the basis for many machine learning technologies by providing functionality for dimensionality reduction, feature selection, feature extraction, ensemble techniques, and inbuilt datasets. Scikit-learn is a top choice of academic and industrial organizations for carrying out a variety of tasks because of its effectiveness and adaptability.

The essential features of scikit-learn to simplify machine learning includes:

. **Unsupervised Learning Algorithms:** This group of algorithms includes unsupervised neural networks, principal component analysis, cluster analysis, and factoring.
. **Feature Extraction:** Scikit-Learn allows you to extract features from both text and images.
. **Dimensionality Reduction:** With the help of this feature, the number of attributes in the data can be minimized for later feature selection, visualization, and summarization.
. **Clustering:** This feature allows the grouping of unlabeled data.
. **Algorithms for Supervised Learning:** There is an extremely high chance that any supervised machine learning algorithm you have heard of is included in the scikit-learn library. Such supervised learning algorithms are available in the scikit-learn toolkit and include generalized linear models, such as linear regression, decision trees, support vector machines, and Bayesian techniques. **You will learn supervised learning techniques in the coming weeks.**
. **Cross-validation:** Scikit-learn can be used to test the accuracy and validity of supervised models using unseen data.
. **Ensemble methods:** The predictions of various supervised models can be integrated. by using this feature.

Library Installation

The library usually comes **pre-installed with Anaconda**. So, you can directly import the library to your notebook. However, if you face any issue while importing the library, you can try to install it by running the below command in your Jupyter Notebook:

```
!pip install scikit-learn
```

Once the installation is completed successfully, restart Jupyter and try import the library again.

Now, Let’s go through some of the common functions used by the Scikit-learn library:

1) **sklearn.preprocessing.StandardScaler**

Each feature or variable is scaled to unit variance once the mean has been removed by StandardScaler. This process is carried out independently based on features. Since StandardScaler involves the estimation of the empirical mean and standard deviation of each feature, outliers can have an impact on it (if they are present in the dataset).

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = pd.DataFrame(scaler.fit_transform(data))
```

In the above code, we are importing Standard Scaler (first line of code), initializing it (second line of code), applying it on my data and storing the output in another variable called data_scaled (third line of code).

You can refer to the standard scalar sklearn documentation for a better understanding of the parameters and attributes here.

2) **sklearn.cluster.KMeans**

The K-Means algorithm clusters the data by trying to separate samples into n groups by minimizing a criterion known as the inertia or within-cluster sum-of-squares (see below). This algorithm requires the number of clusters to be specified. It scales well to large numbers of samples and has been used across a large range of application areas in many different fields.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(data_scaled)
```

The above code is performing the K-means clustering with having 3 clusters on the data frame data_scaled.

You can refer to the K-Means sklearn documentation for a better understanding of the parameters and attributes here.

3) **sklearn.metrics.silhouette_score**

The silhouette score measures how effectively samples are clustered with other samples that are similar to them in order to assess the quality of clusters produced by clustering algorithms like K-Means. Each sample of various clusters gets a Silhouette score.

To compute Silhouette scores we import this metric from sklearn by using the below code line:

```
from sklearn.metrics import silhouette_score
 # Predicting on the data_scaled
 preds = kmeans.fit_predict((data_scaled))

 # Calculating silhouette score
 score = silhouette_score(data_scaled, preds)
```

In the above code, we are calculating the silhouette_score on the data and its predictions. For more understanding refer to this source.

4) **sklearn.decomposition.PCA**

High-dimensional data can be made simpler through the use of principal component analysis (PCA), while still preserving trends and patterns. It accomplishes this by compressing the data into fewer dimensions that act as feature summaries.

Importing PCA from sklearn by using the below code line:

```
from sklearn.decomposition import PCA
# Initializing PCA function with n components
pca = PCA(n_components=n, random_state=1)

# fit_transform PCA on scaled data
data_pca = pd.DataFrame(pca.fit_transform(df_scaled))
```

Here, we are creating the object for the PCA and then fitting it on the data data_scaled, storing the results into another variable. You can refer to the PCA sklearn documentation for a better understanding of the parameters and attributes here.

5) **sklearn.preprocessing.LabelEncoder**

Label encoding is the process of transforming labels into a numeric form so that they may be read by machines. The operation of those labels can then be better determined by machine learning techniques. It is a significant supervised learning pre-processing step for the structured dataset.

Encoding the variable from sklearn by using the below code line:

```
from sklearn.preprocessing import LabelEncoder
ln = LabelEncoder()
```

You can refer to the LabelEncoder sklearn documentation for a better understanding of the parameters and attributes here.

6) **sklearn.manifold.TSNE**

T-distributed Stochastic Neighbourhood Embedding (tSNE) is an unsupervised Machine Learning algorithm. It has become widely used in data science to visualize the structure of high-dimensional data in 2 or 3 dimensions.

Importing TSNE from sklearn by using the below code line:

```
from sklearn.manifold import TSNE
# Initializing T-SNE function with 2 components
tsne = TSNE(n_components=2, random_state=1, perplexity=35)

tsne_data = tsne.fit_transform(data)
```

Here, we are creating the object for the TNSE and then fitting it on the dataframe data, storing the results into another variable. You can refer to the TSNE sklearn documentation for a better understanding of the parameters and attributes here.

7) **sklearn.cluster.AgglomerativeClustering**

Agglomerative clustering uses a bottom-up approach, wherein each data point starts in its own cluster. These clusters are then joined greedily, by taking the two most similar clusters together and merging them.

Performing hierarchical clustering, computing cophenetic correlation, and creating dendrograms from sklearn by using the below code line:

```
from sklearn.cluster import AgglomerativeClustering
# Initializing Agglomerative Clustering
model = AgglomerativeClustering(n_clusters=3, affinity="euclidean", linkage="ward",)

# Fitting on PCA data
model.fit(data_pca)
```

Here, Initializing Agglomerative Clustering with distance as Euclidean, linkage as a ward with clusters=3. We are fitting the Agglomerative Clustering model on the pca data. You can refer to the Agglomerative clustering sklearn documentation for a better understanding of the parameters and attributes here.

8) **sklearn.mixture.GaussianMixture**

Subpopulations within a larger population that are normally distributed are represented using Gaussian Mixture models. Mixture models have the benefit of not requiring knowledge of the subpopulation to which a data point belongs. It enables the model to automatically learn the subpopulations.

Importing Gaussian Mixture from sklearn by using the below code line:

```
from sklearn.mixture import GaussianMixture
# Initializing Gaussian Mixture Model
gmm = GaussianMixture(n_components=5, random_state=1)

# Fitting and predicting Gaussian Mixture Model on data_pca
preds = gmm.fit_predict((data_pca))
```

In the above code, we are initializing Gaussian Mixture Model with a number of clusters as 5 and random_state=1. You can refer to the Gaussian Mixture sklearn documentation for a better understanding of the parameters and attributes here.