Intel oneAPI Data Analytics Library (oneDAL)

2023 Q1

Intel® oneAPI Data Analytics Library (oneDAL) is a library that helps speed up big data analysis by providing highly optimized algorithmic building blocks for all stages of data analytics (preprocessing, transformation, analysis, modeling, validation, and decision making) in batch, online, and distributed processing modes of computation.

The library optimizes data ingestion along with algorithmic computation to increase throughput and scalability. It includes C++ and Java* APIs and connectors to popular data sources such as Spark* and Hadoop*. Python* wrappers for oneDAL are part of Intel Distribution for Python.

To setup and install oneDAL:

```
!pip install scikit-learn-intelex
from sklearnex import patch_sklearn
patch_sklearn()
```

Data Preprocessing: Feature Scaling

```
from sklearn.preprocessing import StandardScaler
from sklearnex import patch_sklearn
patch_sklearn()
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Data Preprocessing: Imputing Missing Data

```
from sklearn.impute import SimpleImputer
from sklearnex import patch_sklearn
patch_sklearn()
imputer = SimpleImputer(strategy='mean')
X_train_imputed =
imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)
```

K-means clustering

```
from sklearnex.cluster import KMeans
kmeans_algo = KMeans(n_clusters=2, max_iter=10)
kmeans_algo.fit(data)
centroids = kmeans_algo.cluster_centers_
```

PCA

```
from sklearnex.decomposition import PCA
pca_algo = PCA(n_components=2)
pca_result = pca_algo.fit_transform(data)
transformed_data = pca_result.transformed_data_
```

DBSCAN clustering

```
from sklearnex.cluster import DBSCAN
dbscan_algo = DBSCAN(eps=1.0, min_samples=10)
dbscan_result = dbscan_algo.fit(data)
labels = dbscan_result.labels_
```

SVC

```
from sklearnex.svm import SVC
svm_algo = SVC(C=1.0, cache_size=200.0)
svm_algo.fit(data, labels)
predicted_labels = svm_algo.predict(data)
```

Linear Regression

```
from sklearnex.linear_model import LinearRegression
lr_algo = LinearRegression()
lr_algo.fit(data, labels)
coef = lr_algo.coef_
intercept = lr_algo.intercept_
```

Logistic Regression

```
from sklearnex.linear_model import
LogisticRegression
logr_algo = LogisticRegression() logr_algo.fit(data,
labels)
predicted_labels = logr_algo.predict(data)
```

Naive Bayes Classification

```
from sklearnex.naive_bayes import GaussianNB
nb_algo = GaussianNB()
nb_algo.fit(data, labels)
predicted_labels = nb_algo.predict(data)
```

Random Forest Classification

```
from sklearnex.ensemble import
RandomForestClassifier
rfc_algo = RandomForestClassifier(n_estimators=100,
max_depth=5)
rfc_algo.fit(data, labels)
predicted_labels = rfc_algo.predict(data)
```

Support Forums:

GitHub Repo | Intel DevMesh Discord

We encourage you to check out Intel's full suite of AI tools and framework optimizations.

Intel oneAPI Deep Neural Network Library (oneDNN)

2023 Q1

Intel® oneAPI Deep Neural Network Library (oneDNN) is an open-source performance library for deep learning applications. The library includes basic building blocks for neural networks optimized for Intel Architecture Processors and Intel Processor Graphics. oneDNN is intended for deep learning applications and framework developers interested in improving application performance on Intel Architecture Processors and Intel Processor Graphics. Deep learning practitioners should use one of the applications enabled with oneDNN.

oneDNN is distributed as part of Intel® oneAPI DL Framework Developer Toolkit, the Intel oneAPI Base Toolkit, and is available via apt and yum channels.

Setup oneDNN in Devcloud:

```
mkdir MLoneAPI
cd MLoneAPI
source
/glob/development-tools/versions/oneapi/2022.3.1/inteloneapi/setvars.sh
conda activate base
pip install ipykernel
python -m ipykernel install --user --name 2022.3.1 --display-name "oneAPI
2022.3.1"
!pip install onednn-cpu-gomp
import oneDNN as dnn
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '1'
os.environ['TF_ENABLE_AUTO_MIXED_PRECISION'] = '1'
```

To configure oneDNN to use GPU acceleration:

```
import os
os.environ['DNNL_ENGINE_LIMI
T_CPU_CAPABILITIES'] = '0'
os.environ['DNNL_VERBOSE'] =
'1'
os.environ['SYCL_DEVICE_FILT
ER'] = 'opencl:gpu'
```

Creating a fully connected neural network layer with oneDNN:

```
import tensorflow as tf
import onednn as dnn

input_shape = [batch_size, input_size]
output_shape = [batch_size, output_size]
input_tensor = tf.keras.layers.Input(shape=input_shape)
dense_layer = dnn.Dense(units=output_size, input_shape=input_shape)(input_tensor)
model = tf.keras.Model(inputs=input_tensor, outputs=dense_layer)
```

Creating a convolutional neural network layer with oneDNN:

```
input_shape = [batch_size, in_channels, height, width]
output_shape = [batch_size, out_channels, out_height, out_width]
kernel_shape = [kernel_h, kernel_w]
input_tensor = tf.keras.layers.Input(shape=input_shape)
conv_layer = dnn.Conv2D(
    filters=out_channels,
    kernel_size=kernel_shape,
    strides=strides,
    padding=padding,
    input_shape=input_shape)(input_tensor)
model = tf.keras.Model(inputs=input_tensor, outputs=conv_layer)
```

Creating a max pooling layer with oneDNN:

```
import tensorflow as tf
import onednn as dnn

input_shape = [batch_size, in_channels, height, width]
output_shape = [batch_size, out_channels, out_height, out_width]
pool_size = [pool_h, pool_w]
input_tensor = tf.keras.layers.Input(shape=input_shape)
max_pool_layer = dnn.MaxPooling2D(pool_size=pool_size)(input_tensor)
model = tf.keras.Model(inputs=input_tensor, outputs=max_pool_layer)
```

Support Forums:

GitHub Repo | Intel DevMesh Discord

We encourage you to check out Intel's full suite of AI tools and framework optimizations.