spscicomp Documentation

Release beta

The Project Group

January 15, 2015

CONTENTS

Contents:

CHAPTER

ONE

COMMON MODULES

Currently there is one common module for all algorithms, namely the data importer module. It provides the following classes for importing numerical data:

1.1 common_data_importer

class `common_data_importer.`**`CommonBinaryFileDataImporter`**(*filename*)

Import data from a binary file. The file format should be as generated by `numpy.save()`.

`get_data`(*size*)

Return a numpy array of floats where each data point occupies one row of the array. The data is read from the current position of the pointer onwards. If the pointer reaches the end of the file, an array of all data points up to the end of the file is returned and the hasMoreData flag is set to False.

Parameters **size** (*int*) – Number of data points to be returned.

Returns A numpy array of data points.

Return type numpy.array

`has_more_data`()

Test if the pointer is at the end of the file or not.

Returns True if there is more data after the pointer, and False if not.

Return type bool

`init_file_input_stream`()

Create a numpy array object which reads from the binary file using a memmap.

`rewind`()

Reset the file pointer to the beginning and set the hasMoreData flag to True.

class `common_data_importer.`**`CommonDataImporter`**

This is an abstract data importer class. Implementations are expected to override the get_data and has_more_data methods.

class `common_data_importer.`**`CommonFileDataImporter`**(*filename*)

Import data from a text file. The data structure should be as follows: One point occupies one line. Each point consists of several floats with space as a separator.

`close_file`()

Close the file handle if it is open.

`get_data`(*size*)

Return a numpy array of floats where each data point occupies one row of the array. The data is read from the current position of the pointer onwards. If the pointer reaches the end of the file, an array of all data points up to the end of the file is returned, the file is closed and the hasMoreData flag is set to False.

Parameters **size** (*int*) – Number of data points to be returned.

Returns A numpy array of data points.

Return type numpy.array

`has_more_data``()`

Test if the pointer is at the end of the file or not.

Returns True if there is more data after the pointer, and False if not.

Return type bool

`init_file_input_stream``()`

Initialize the file input stream, that is, open the file and create the iterator on the file's lines.

`rewind``()`

Reset the file pointer to the beginning, that is, initialize the file and set the hasMoreData flag to True.

class `common_data_importer.`**`CommonSimpleDataImporter`**`(`*data*`)`

"Import" data from a given data array.

`get_data``(`*size*`)`

Return all available data regardless of the requested size.

Parameters **size** (*int*) – Size of data which is to be returned. This parameter is disregarded as all data is returned.

Returns All data.

Return type numpy.array

`has_more_data``()`

Return if there is any more data. As all data is returned when using get_data, this function always returns False.

Returns False since there never is any more data.

Return type bool

CHAPTER

TWO

THE K-MEANS ALGORITHM

The implementation of the k-means algorithm consists of the following modules:

2.1 kmeans_main

`kmeans_main.`**`kmeans`**(*k*, *importer=None*)

Initialize and run the k-means algorithm. If any of the optimized implementations (CUDA, OpenCL, C extension) are available, they are selected and initialized automatically in the above order. Then the respective `kmeans.Kmeans.calculate_centers()` method is called and the output is returned.

Parameters

- **k** (*int*) – Number of cluster centers to compute.
- **importer** (`CommonDataImporter`) – A `CommonDataImporter` object to be used for importing the numerical data.

Returns An array of integers $[c(x_i)]$ where x_i is the i-th data point and $c(x_i)$ is the index of the cluster center to which x_i belongs.

Return type int[]

2.2 kmeans

class `kmeans.`**`DefaultKmeans`**(*metric=<kmeans_metric.EuclideanMetric object at 0x7f1901016750>*, *importer=None*, *chunk_size=1000*, *max_steps=100*)

Default implementation of the k-means algorithm. Once supplied with an `CommonDataImporter` object, use the calculate_centers method to compute k cluster centers.

Parameters

- **metric** (`KmeansMetric`) – A `KmeansMetric` object to be used for calculating distances between points. The default is the `EuclideanMetric`.
- **importer** (`CommonDataImporter`) – A `CommonDataImporter` object to be used for importing the numerical data.
- **chunk_size** (*int*) – The number of data points to be imported and processed at a time.
- **max_steps** (*int*) – The maximum number of steps to run the algorithm for. If the iteration did not converge after this number of steps, the algorithm is terminated and the last result returned.

calculate_centers(*k*, *initial_centers=None*, *return_centers=False*, *save_history=False*)

Main method of the k-means algorithm. Computes k cluster centers from the data supplied by a `CommonDataImporter` object.

Parameters

- **k** (*int*) – Number of cluster centers to compute.
- **initial_centers** (*numpy.array*) – Array of cluster centers to start the iteration with. If omitted, random data points from the first chunk of data are used.
- **return_centers** (*bool*) – If set to True then the cluster centers are returned.
- **save_history** (*bool*) – If this and return_centers is set to True then the cluster centers in each iteration step are returned.

Returns An array of integers $[c(x_i)]$ where x_i is the i-th data point and $c(x_i)$ is the index of the cluster center to which x_i belongs.

Return type int[]

Returns An array of the computed cluster centers.

Return type np.array

Returns A list of arrays of the cluster centers in each iteration step.

Return type np.array[]

class kmeans.**Kmeans**(*metric=<kmeans_metric.EuclideanMetric object at 0x7f1901016290>*, *importer=None*)

Abstract k-means algorithm. Implementations are expected to override the calculate_centers method.

2.3 c_kmeans

class extension.c_kmeans.**CKmeans**(*metric=<kmeans_metric.EuclideanMetric object at 0x7f1900f60290>*, *importer=None*, *chunk_size=1000*, *max_steps=100*)

An implementation of the k-means algorithm in C. Refer to the `DefaultKmeans` class for parameters and public methods.

2.4 cuda_kmeans

class cuda.cuda_kmeans.**CUDAKmeans**(*metric=EuclideanMetric()*, *importer=None*, *chunk_size=1000*, *max_steps=100*)

An implementation of the k-means algorithm in CUDA. Refer to the `DefaultKmeans` class for parameters and public methods.

2.5 opencl_kmeans

class opencl.opencl_kmeans.**OpenCLKmeans**(*metric=EuclideanMetric()*, *importer=None*, *chunk_size=1000*, *max_steps=100*)

An implementation of the k-means algorithm in OpenCL. Refer to the `DefaultKmeans` class for parameters and public methods.

2.6 kmeans_data_generator

class `kmeans_data_generator.`**`KmeansDataGenerator`**

Abstract data generator. Implementations are expected to override the generate_data method.

class `kmeans_data_generator.`**`KmeansRandomDataGenerator`**(*size*, *dimension*, *centers_count*)

Generate a test dataset for the k-means algorithm. The centers are generated uniformly. The other points are produced randomly near one of the centers with normal distribution.

Parameters

- **size** (*int*) – Number of data points to generate.
- **dimension** (*int*) – Dimension of the euclidean space the data points will belong to.
- **centers_count** (*int*) – Number of cluster centers around which the data points are to be generated.

`get_centers`()

Return the generated cluster centers.

Returns A list of numpy arrays representing the cluster centers.

Return type np.array[]

`get_data`()

Return the generated data points.

Returns A numpy array of size *size*x*dimension*.

Return type np.array

`to_binary_file`(*filename*)

Save the generated data to a binary file using `numpy.save()` which can be read later using the respective `CommonDataImporter` object.

Parameters **filename** (*str*) – The file name.

`to_file`(*filename*)

Save the generated data to a text file using `numpy.savetxt()` which can be read later using the respective `CommonDataImporter` object.

Parameters **filename** (*str*) – The file name.

CHAPTER THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX