C++:Self-Organizing-Map:CUDA

0.0.1

Generated by Doxygen 1.8.17

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Namespace Documentation

3.1 confusion_matrix Namespace Reference

This namespace contains the methods that derive certain metrics from a provided confusion matrix.

Typedefs

- typedef std::vector< std::vector< int > > matrix

  This type hides the confusion matrix container implementation type.

Functions

- void print_matrix (std::ostream &out, matrix m, int dim)

  Prints the confusion matrix to an output stream.
- double accuracy (matrix m, int dim)

  calculates the overall accuracy of the classifier.
- std::vector< double > positive_precisions (matrix m, int dim)
- std::vector< double > sensitivity (matrix m, int dim)
- std::vector< double > specificity (matrix m, int dim)
- std::vector< double > fscore (std::vector< double > precisions, std::vector< double > sensitivities)
- double sampleVariance (std::vector< double > samples, double mean)
- double sampleVariance (std::vector< double > samples)
- std::pair< double, double > confidence95 (std::vector< double > samples)
- std::vector< double > AUC (matrix m, int dim)

3.1.1 Detailed Description

This namespace contains the methods that derive certain metrics from a provided confusion matrix.

3.1.2 Function Documentation

3.1.2.1 AUC()

```
std::vector<double> confusion_matrix::AUC (
            matrix m,
            int dim )
```

Calculates the Area Under Curve of each class for a particular classifier.

3.1.2.2 confidence95()

```
std::pair<double, double> confusion_matrix::confidence95 (
            std::vector< double > samples )
```

Returns a 95% confidence interval to the mean of a particular set of samples. A tuple (mu, error) is returned with mu being the mean and error being the +-error.

3.1.2.3 fscore()

```
std::vector<double> confusion_matrix::fscore (
            std::vector< double > precisions,
            std::vector< double > sensitivities )
```

Calculates the fscore (2 / ((1/precision) + (1/sensitivity)))) of the classifier. Employs a one versus all strategy, resulting in a value for every class.

Parameters

precisions	The precision of every class.
sensitivities	The sensitivity of every class.

3.1.2.4 positive_precisions()

```
std::vector<double> confusion_matrix::positive_precisions (
            matrix m,
            int dim )
```

Calculates the positive precision (TP / (TP + FN)) of the classifier. Employs a one versus all strategy, resulting in a value for every class.

3.1.2.5 sampleVariance() [1/2]

```
double confusion_matrix::sampleVariance (
            std::vector< double > samples )
```

Clalculates the variance of some samples.

3.1.2.6 sampleVariance() [2/2]

```
double confusion_matrix::sampleVariance (
            std::vector< double > samples,
            double mean )
```

Clalculates the variance of some samples, with the mean already computed.

3.1.2.7 sensitivity()

```
std::vector<double> confusion_matrix::sensitivity (
            matrix m,
            int dim )
```

Calculates the positive sensitivity (TP / (TP + FP)) of the classifier. Employs a one versus all strategy, resulting in a value for every class.

3.1.2.8 specificity()

```
std::vector<double> confusion_matrix::specificity (
            matrix m,
            int dim )
```

Calculates the specificity (TN / (TN + FP)) of the classifier. Employs a one versus all strategy, resulting in a value for every class.

3.2 io Namespace Reference

Namespaces

- SOM

Typedefs

- typedef std::map< std::string, std::string > argumentOptions

  This type hides the container type of the argument options.

Functions

- argumentOptions parse_options (int argc, const char ∗argv[ ])

3.2.1 Detailed Description

This namespace holds methods needed for the parsing of argument options and reading of training and testing datasets.

3.2.2 Function Documentation

3.2.2.1 parse_options()

```
argumentOptions io::parse_options (
             int argc,
             const char * argv[] )
```

Returns an argument options container that contains the configuration options passed to the executable via CLI.

3.3 io::SOM Namespace Reference

Functions

- ::SOM::configuration parse_SOM_configuration (argumentOptions options)
- digitSet parseInputSet (argumentOptions options, ::SOM::configuration conf)
- digitSet parseTestingSet (argumentOptions options, ::SOM::configuration conf)

3.3.1 Detailed Description

This sub-namespace contains the parsing of options needed for the configuring of the SOM.

3.3.2 Function Documentation

3.3.2.1 parse_SOM_configuration()

```
::SOM::configuration io::SOM::parse_SOM_configuration (
             argumentOptions options )
```

Returns a SOM::configuration read from CLI.

3.3.2.2 parseInputSet()

```
digitSet io::SOM::parseInputSet (
             argumentOptions options,
             ::SOM::configuration conf )
```

Returns a training dataset read from files provided by the CLI arguments and SOM configuration.

3.3.2.3 parseTestingSet()

```
digitSet io::SOM::parseTestingSet (
             argumentOptions options,
             ::SOM::configuration conf )
```

Returns a testing dataset read from files provided by the CLI arguments and SOM configuration.

3.4 modelValidation Namespace Reference

Contains methods for the validation of models.

Functions

- std::pair< digitSet, digitSet > random_split (const digitSet &train, const digitSet &test)
- std::pair< digitSet, digitSet > crossvalidate_split (const digitSet &train, const digitSet &test, int k, int test↩ FractionIndex)

3.4.1 Detailed Description

Contains methods for the validation of models.

3.4.2 Function Documentation

3.4.2.1 crossvalidate_split()

```
std::pair<digitSet, digitSet> modelValidation::crossvalidate_split (
             const digitSet & train,
             const digitSet & test,
             int k,
             int testFractionIndex )
```

Splits the dataset consisting of a training and testing part in a crossvalidating manner using a provided k fold parameter, and the index of the fold to be designated as the testing set.

Parameters

train	The training part of the dataset.
test	The testing part of the dataset.
k	The number of folds of the crossvalidation.
testFractionIndex	The index of the fold to be used as testing.

3.4.2.2 random_split()

```
std::pair<digitSet, digitSet> modelValidation::random_split (
             const digitSet & train,
             const digitSet & test )
```

Splits the whole of a dataset consisting of a training and testing dataset in a random manner into new training and testing parts, preserving the cardinalities.

Parameters

train	The training part of the dataset.
test	The testing part of the dataset.

3.5 SOM Namespace Reference

The namespace of SOM related classes, methods and structures.

Classes

- struct configuration

  Structure that provides the configuration of the SOM.
- class SelfOrganizingMap

  The class that implements the Self Organizing Map model.

Typedefs

- typedef std::vector< std::vector< digit > > **SOMContainer**

Functions

- int classify (std::vector< SelfOrganizingMap > &maps, const digit &sample)

  Classifies a sample image into one of the categories represented by the maps.

3.5.1 Detailed Description

The namespace of SOM related classes, methods and structures.

This namespace contains the SelfOrganizingMap class, a structure that configures the training of it, and a method used to classify query points.

3.5.2 Function Documentation

3.5.2.1 classify()

```
int SOM::classify (
            std::vector< SelfOrganizingMap > & maps,
            const digit & sample )
```

Classifies a sample image into one of the categories represented by the maps.

This method claculates the distance from the sample point to every map. The sample is then classified according to the class of the map to which it is closest.

Parameters

maps	The SOMs that were trained on distinct classes of the dataset.
sample	The query point that needs to be classified

Chapter 4

Class Documentation

4.1 SOM::configuration Struct Reference

Structure that provides the configuration of the SOM.

Public Member Functions

- void printConfiguration (std::ostream &out)

  Prints all of the configuration parameters into an output stream.

Public Attributes

- int digitW
- int digitH
- int mapW
- int mapH
- int maxT
- bool animation
- std::string animationPath
- int frameCount
- bool classification
- int classCount

4.1.1 Detailed Description

Structure that provides the configuration of the SOM.

4.1.2 Member Data Documentation

4.1.2.1 animation

`bool SOM::configuration::animation`

This flags that the user wants to save intermediate states during training.

4.1.2.2 animationPath

`std::string SOM::configuration::animationPath`

The path to save resulting animation frames to.

4.1.2.3 classCount

`int SOM::configuration::classCount`

The number of classes into which the images have to be categorized in.

4.1.2.4 classification

`bool SOM::configuration::classification`

This flags that classification procedure is enabled.

4.1.2.5 digitH

`int SOM::configuration::digitH`

Height of the input image in pixels

4.1.2.6 digitW

`int SOM::configuration::digitW`

Width of the input image in pixels

4.1.2.7 frameCount

`int SOM::configuration::frameCount`

The number of frames that need to be saved.

4.1.2.8 mapH

`int SOM::configuration::mapH`

Height of the SOM structure.

4.1.2.9 mapW

```
int SOM::configuration::mapW
```

Width of the SOM structure.

4.1.2.10 maxT

```
int SOM::configuration::maxT
```

Number of maximum iterations during training.

The documentation for this struct was generated from the following file:

- src/SOM.cpp

4.2 digit Class Reference

A feature vector container of continuous features.

Public Member Functions

- int dimension () const

  Return the number of features in the vector.
- digit (int width=8, int height=8)

  The constructor takes as argument a width and height of the vector.
- ∼digit ()

  Deallocates the feature vector.
- digit (const digit &other)

  Copies the feature values from the other container into a newly allocated memory region.
- void operator= (const digit &other)
- int getWidth () const

  Returns the width of the container.
- int getHeight () const

  Returns the height of the container.
- double ∗ getShades () const

  Returns the pointer to the allocated memory region.
- void initrandom (double min, double max)

  Randomly initializes the values in the container in a certain range.
- std::ostream & appendToFile (std::ostream &out, std::function< double(double)> &&mappedShade=[ ](double shade) { return shade;})
- digit operator+ (const digit &other) const

  Return a new digit that is the vectorial sum of the instance and the other vector.
- digit minus (const digit &other) const

  Return a new digit that is the vectorial subtraction of the instance and the other vector.
- digit operator∗ (double scalar) const

  Return a new digit that is the scalar multiplication of the instance with the provided scalar.
- double getMaxAbsShade ()

Returns the maximum absolut value of the features of the container.
- double getMaxShade ()

  Returns the maximum value of the features of the container.
- double getMinShade ()

  Returns the minimum absolut value between the features of the container.
- void minMaxNormalize (double min, double max)

  Normalizes the features using min,max normalization.
- double operator[ ] (int index) const

  Returns the feature in the container at a particular index.
- int getValue () const

  Returns the index of the label of the current vector.
- double operator- (const digit &other) const

  Return the distance between two vectors in euclidean distance metric.

Friends

- std::istream & operator>> (std::istream &in, digit &other)

  Friend function, reads a new feature vector from an input stream.
- std::ostream & operator<< (std::ostream &out, const digit &other)

  Prints the features to an output stream and the associated label index.

4.2.1 Detailed Description

A feature vector container of continuous features.

This class is a container of continuous features. It supports +,- vector operations, random initialization within a range, and min/max normalization. It also contains a classification label.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 digit()

```
digit::digit (
              int width = 8,
              int height = 8 )   [inline]
```

The constructor takes as argument a width and height of the vector.

Parameters

width	The width of the image.
height	The height of the image.

4.2.3 Member Function Documentation

4.2.3.1 appendToFile()

```
std::ostream& digit::appendToFile (
            std::ostream & out,
            std::function< double(double)> && mappedShade = [](double shade) { return shade; }
) [inline]
```

Prints the features to an output stream in lexicographical order, followed by an endline. It supports a mapping functionality as well.

Parameters

out	The output stream to print the features to.
mappedShade	The mapping function of the features. It defaults to the identity function.

4.2.3.2 initrandom()

```
void digit::initrandom (
            double min,
            double max ) [inline]
```

Randomly initializes the values in the container in a certain range.

Parameters

min	The lowest possible random value.
max	The highest possible random value.

4.2.3.3 minMaxNormalize()

```
void digit::minMaxNormalize (
            double min,
            double max ) [inline]
```

Normalizes the features using min,max normalization.

Parameters

min	The bottom of the normalization range.
max	The top of the normalization range.

4.2.3.4 operator*()

```
digit digit::operator* (
             double scalar ) const  [inline]
```

Return a new digit that is the scalar multiplication of the instance with the provided scalar.

Parameters

scalar	The scalar with which the instance is multiplied.

4.2.3.5 operator=()

```
void digit::operator= (
             const digit & other )  [inline]
```

Copies the feature values from the other container into a newly allocated memory region. Deallocates the old memory to enable resizing.

4.2.3.6 operator[]()

```
double digit::operator[] (
             int index ) const  [inline]
```

Returns the feature in the container at a particular index.

Parameters

index	The index of the feature to be returned.

4.2.4 Friends And Related Function Documentation

4.2.4.1 operator<<

```
std::ostream& operator<< (
             std::ostream & out,
             const digit & other )  [friend]
```

Prints the features to an output stream and the associated label index.

Parameters

out	The output stream to which the features will be written. After the features the associated label is also printed.

4.2.4.2 operator>>

```
std::istream& operator>> (
            std::istream & in,
            digit & other )  [friend]
```

Friend function, reads a new feature vector from an input stream.

Parameters

in	The input stream from which the features will be read. After the features the associated label is read.
other	The container reference that will contain the read features.

The documentation for this class was generated from the following file:

- src/digit.cpp

4.3 digitSet Class Reference

Public Member Functions

- digitSet (int width, int height=1)

    The constructor takes as argument the dimensionality of the vectors.
- void add (const digit &d)

    Adds a new sample to the set of samples.
- std::pair< double, double > minMaxFeatureScale ()

    This method normalizes the samples over the maximum and minimum value of the overall features.
- **digitSet** (const digitSet &other)
- int getWidth () const

    Returns the width of the feature vectors.
- int getHeight () const

    Returns the height of the feature vectors.
- const digit & getDigit (int index) const

    Returns the digit at a given index.
- const std::vector< digit > & getDigits () const

    Returns container that holds the samples.
- int size () const

    Returns the number of samples held in the container.
- int dimension () const

    Returns the overall dimensionality of every sample. Every sample is of the same dimensionality.

Static Public Member Functions

- static digitSet filterByValue (const digitSet &tofilter, double value)

  Filters all the samples of a particular class from a sample set into a new set.

Protected Attributes

- std::vector< digit > **_digits**
- int **_width**
- int **_height**

Friends

- std::istream & operator>> (std::istream &in, digitSet &other)
- std::ostream & operator<< (std::ostream &out, digitSet &other)

4.3.1 Detailed Description

This class is a container for a set of vector samples. It supports the addition of elements, sample normalization and readind writing to streams.

4.3.2 Friends And Related Function Documentation

4.3.2.1 operator<<

```
std::ostream& operator<< (
             std::ostream & out,
             digitSet & other )   [friend]
```

Prints a sample set to an output stream line by line.

4.3.2.2 operator>>

```
std::istream& operator>> (
             std::istream & in,
             digitSet & other )   [friend]
```

Reads a sample set from an input stream. It parses the vectors line by line until end of stream.

The documentation for this class was generated from the following file:

- src/digitSet.cpp

4.4 SOM::SelfOrganizingMap Class Reference

The class that implements the Self Organizing Map model.

Public Member Functions

- SelfOrganizingMap (const digitSet &points, int mapWidth, int mapHeight)
- SelfOrganizingMap (const SelfOrganizingMap &other)
- void setup_CUDA (int sampleDim, int sampleCount)
- void copy_samples_to_device (int sampleDim)

    Copies the samples from the host memory to CUDA device memory.
- void copy_map_to_device (int dim, int mapWidth, int mapHeight)

    Copies the map from the host memory to CUDA device memory.
- void copy_map_from_device (int sampleDim, int mapWidth, int mapHeight)

    Copies the resulting trained map from the CUDA device back to the host memory.
- ∼SelfOrganizingMap ()

    Deallocates all of the allocated memory by the class on the CUDA device.
- void initializeRandomSOM (SOM::SOMContainer &som, int dimx, int dimy)

    Initializes the map in a random manner.
- void initializeSampledSOM (SOM::SOMContainer &som, int dimx, int dimy)

    Initializes the map in a random manner.
- bool **safebound** (int i, int j)
- double normal_pdf (double x, double m, double s)
- double euclideanDistance (int i1, int j1, int i2, int j2)
- double normalNeighbourCoefficient (int i1, int j1, int i2, int j2, double radius)
- SOM::SOMContainer & getMap ()

    Returns the internal map.
- void train (int maxT, std::function< void(int, int, SelfOrganizingMap ∗)> &&everyFewIterations=[ ](int, int, SelfOrganizingMap ∗) {})

    The training method of the map. Results in a trained model on the host memory side.
- digit getClosestSample (int i, int j)

    Searches for the sample that is closest to a given neuron within the training data (host side).
- void printMap ()

    Prints the labeling of the neurons according to the category of their closest sample.
- void printMapToStream (std::ostream &out)

    Prints the map to an output stream.
- double getClosestPrototypeDistance (const digit &sample)

    Calculates the distance from a sample to its best matching unit.

4.4.1 Detailed Description

The class that implements the Self Organizing Map model.

This class encapsulates a dataset of training points on which the model is trained and the actual map. The CUDA implementation copies the trainig points and the map to the device memory and operates on them inplace. After the training finished, the trained map is copied back to the host user memory.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 SelfOrganizingMap() [1/2]

```
SOM::SelfOrganizingMap::SelfOrganizingMap (
            const digitSet & points,
            int mapWidth,
            int mapHeight )  [inline]
```

The constructor takes a set of points and the dimensions of the map as parameter. It normalizes the inputs in order to prevent the explosion of gradients. Afterwards it initializes the map randomly, allocates memory on the CUDA device for the samples and the map and copies them there.

Parameters

points	The training points on which the map will be trained.
mapWidth	The width of the map.
mapHeight	The height of the map.

4.4.2.2 SelfOrganizingMap() [2/2]

```
SOM::SelfOrganizingMap::SelfOrganizingMap (
            const SelfOrganizingMap & other )  [inline]
```

The copy constructor of the class copies the samples and map both from the host memory and the CUDA device memory.

4.4.3 Member Function Documentation

4.4.3.1 euclideanDistance()

```
double SOM::SelfOrganizingMap::euclideanDistance (
            int i1,
            int j1,
            int i2,
            int j2 )  [inline]
```

The euclidean distance between two 2D points.

4.4.3.2 initializeRandomSOM()

```
void SOM::SelfOrganizingMap::initializeRandomSOM (
            SOM::SOMContainer & som,
            int dimx,
            int dimy )  [inline]
```

Initializes the map in a random manner.

Initializes each representative node within the map using random values within the minimum and maximum values seen in the sample dataset.

Parameters

som	The map that needs to be initialized.
dimx	The width of the images in pixels.
dimy	The height of the images in pixels.

4.4.3.3 initializeSampledSOM()

```
void SOM::SelfOrganizingMap::initializeSampledSOM (
             SOM::SOMContainer & som,
             int dimx,
             int dimy )  [inline]
```

Initializes the map in a random manner.

Initializes each representative node within the map using random samples from the training dataset.

Parameters

som	The map that needs to be initialized.
dimx	The width of the images in pixels.
dimy	The height of the images in pixels.

4.4.3.4 normal_pdf()

```
double SOM::SelfOrganizingMap::normal_pdf (
             double x,
             double m,
             double s )  [inline]
```

Return the value of x in the kernel of the Gaussian Probability Density Function.

Parameters

x	The value at which the kernel needs to be evaluated.
m	The mean of the kernel.
s	The sigma of the kernel.

4.4.3.5 normalNeighbourCoefficient()

```
double SOM::SelfOrganizingMap::normalNeighbourCoefficient (
             int i1,
             int j1,
             int i2,
             int j2,
             double radius )  [inline]
```

Calculates the neighbouring coefficients with a given radius from point (i1,j1) to point (i2,j2).

Parameters

i1	The y index of the first neuron
j1	The x index of the first neuron
i2	The y index of the second neuron
j2	The x index of the second neuron
radius	The radius of the neighbouring function.

4.4.3.6 printMapToStream()

```
void SOM::SelfOrganizingMap::printMapToStream (
             std::ostream & out )  [inline]
```

Prints the map to an output stream.

Prints the dimensions of the map, afterwards it prints the actual neuron weights to an output stream.

4.4.3.7 setup_CUDA()

```
void SOM::SelfOrganizingMap::setup_CUDA (
             int sampleDim,
             int sampleCount )  [inline]
```

Allocates memory on the CUDA device for the provided samples, the map and the intermediary distances

Parameters

sampleDim	The dimensionality of an input sample
sampleCount	The number of samples provided to the map

4.4.3.8 train()

```
void SOM::SelfOrganizingMap::train (
             int maxT,
             std::function< void(int, int, SelfOrganizingMap *)> && everyFewIterations = [](int, int, SelfOrg
)  [inline]
```

The training method of the map. Results in a trained model on the host memory side.

This method trains the model for a given number of iterations that it receives as parameter. It also receives a callback function in order to facilitate the capturing the frames for the animation. Each iteration it samples a random sample from the dataset and updates the model accordingly. The best matching unit search is parallelized and so is the neighbourhood update phase. After the maximum number of iterations it copies the map from CUDA device to host memory.

The documentation for this class was generated from the following file:

- src/SOM.cpp