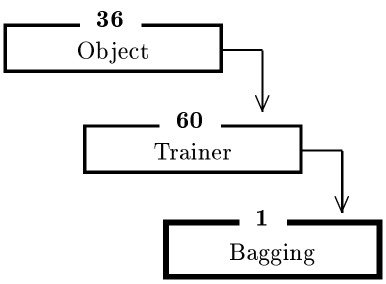
Contents

1 class **Bagging** : public Trainer

This class represents a `Trainer` that implements the well-known Bagging algorithm (Breiman, 1996).

Inheritance



Public Members

1.1	WeightedSumMachine*	**w_machine**	*This machine performs the combination.*	7
1.2	int	**n_trainers**	*The number of trainers in the bagging*	7
1.3	int**	**unselected_examples**	*for each trainer, keep the indices of examples not used during training*	8
1.4	int**	**selected_examples**	*for each trainer, keep the indices of examples used during training*	8
1.5	int*	**n_unselected_examples**	*for each trainer, keep the number of examples not used during training*	8
1.6	int*	**is_selected_examples**	*for each trainer, keep the number of examples used during training*	8
1.7		**Bagging** (WeightedSumMachine* w_machine_, DataSet* data_)		

This class represents a `Trainer` that implements the well-known Bagging algorithm (Breiman, 1996). A "bagger" contains a series of trainers, each trained on a bootstrap of the original dataset. The output of the bagging is then the average of the output of each trainer.

It is implemented using a `WeightedSumMachine` that performs the combination.

See Also: WeightedSumMachine (→63, *page 147*)
Author: Samy Bengio (bengio@idiap.ch)

1.1

WeightedSumMachine* **w_machine**

This machine performs the combination.

This machine performs the combination. It contains many trainers.

1.2

int **n_trainers**

The number of trainers in the bagging

The number of trainers in the bagging

1.3

int** **unselected_examples**

for each trainer, keep the indices of examples not used during training

for each trainer, keep the indices of examples not used during training

1.4

int** **selected_examples**

for each trainer, keep the indices of examples used during training

for each trainer, keep the indices of examples used during training

1.5

int* **n_unselected_examples**

for each trainer, keep the number of examples not used during training

for each trainer, keep the number of examples not used during training

1.6

int* **is_selected_examples**

for each trainer, keep the number of examples used during training

for each trainer, keep the number of examples used during training

1.7

Bagging (WeightedSumMachine* w_machine_, DataSet* data_)

1.8

virtual void **bootstrapData** (int* selected, int* is_selected)

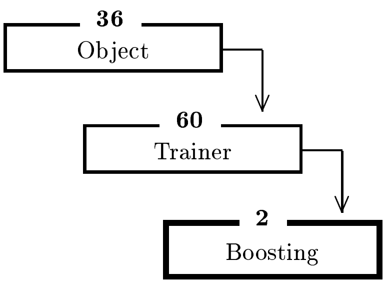
create a boostrap of the data and put in in selected

create a boostrap of the data and put in in selected

2 class **Boosting** : public Trainer

Boosting implementation.

Inheritance



Public Members

2.1	WeightedSumMachine*	**w_machine**	*This machine performs the combination.*	11
2.2	ClassFormat*	**class_format**	`ClassFormat` *to know how the classes are encoded*	11
2.3	int	**n_trainers**	*The number of trainers in the boosting*	11
2.4	real*	**weights**	*The weights of each machine in the boosting*	11
2.5		**Boosting** (WeightedSumMachine* w_machine_, DataSet* data_, ClassFormat* class_format_)		12

Boosting implementation. As the idea of boosting in regression hasn't been really well tested, this is boosting for *classification* only.

This trainer will "boost" the machine given by the `WeightedSumMachine`, on the given `DataSet`.

You have to provide a `ClassFormat` to know how the classes are encoded.

See Also: WeightedSumMachine (→63, *page 147*)
Author: Ronan Collobert (collober@iro.umontreal.ca)

2.1

WeightedSumMachine* **w_machine**

This machine performs the combination.

This machine performs the combination. It contains many trainers.

2.2

ClassFormat* **class_format**

`ClassFormat` *to know how the classes are encoded*

`ClassFormat` to know how the classes are encoded

2.3

int **n_trainers**

The number of trainers in the boosting

The number of trainers in the boosting

2.4

real* **weights**

The weights of each machine in the boosting

The weights of each machine in the boosting

2.5

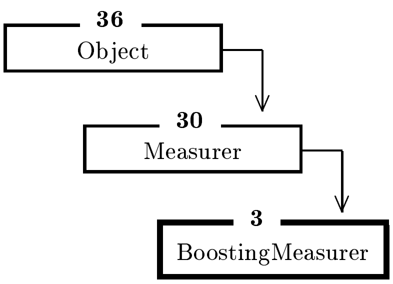
Boosting (WeightedSumMachine* w_machine_, DataSet* data_, ClassFormat* class_format_)

3

class **BoostingMeasurer** : public Measurer

Compute the classification error (in %) for `BoostingMachine` *of the* `inputs` *with respect to the* `targets` *of* `data`*.*

Inheritance



Compute the classification error (in %) for `BoostingMachine` of the `inputs` with respect to the `targets` of `data`.

Needed by `Boosting`. (Designed just for it).

The format of the class is given with a `ClassFormat`.

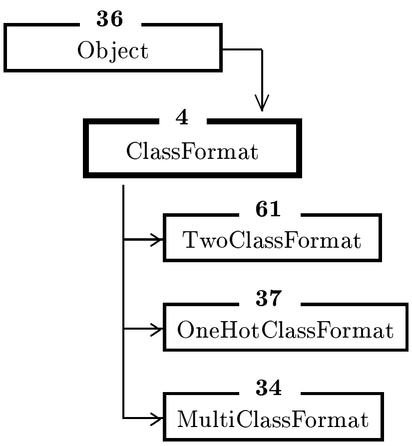
`measure` = <[real] error proportion>, <[real] current number of errors>

See Also: Boosting (→2, *page 10*)
Author: Ronan Collobert (collober@iro.umontreal.ca) Samy Bengio (bengio@idiap.ch)

4 class **ClassFormat** : public Object

Used to define a class code.

Inheritance



Public Members

Used to define a class code.

Author: Ronan Collobert (collober@iro.umontreal.ca)

4.1

int **n_classes**

Number of classes that the object currently handles

Number of classes that the object currently handles

4.2

real** **class_labels**

The label of each class

The label of each class

4.3

ClassFormat ()

4.4

int **getNumberOfClasses** ()

Returns `n_classes`

Returns `n_classes`

4.5

virtual int **getOutputSize** ()

Returns the output size;

Returns the output size;

4.6

virtual void **fromOneHot** (List* outputs, List* one_hot_outputs)

Transforms the output from a OneHot representation

Transforms the output from a OneHot representation

4.7

virtual void **toOneHot** (List* outputs, List* one_hot_outputs)

Transforms the output to a OneHot representation

Transforms the output to a OneHot representation

4.8

virtual int **getTargetClass** (void* target)

Returns the class of `target`

Returns the class of `target`

4.9

virtual int **getOutputClass** (List* outputs)

Returns the class of `outputs`

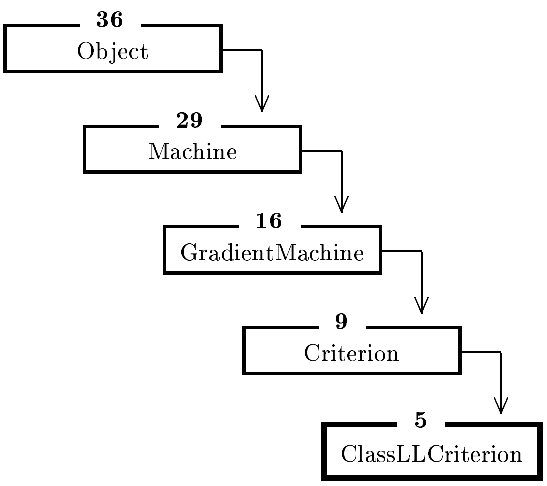
Returns the class of `outputs`

5

class **ClassLLCriterion** : public Criterion

*This criterion can be used to train *in classification* a* `GradientMachine` *object using the* `GMTrainer` *trainer.*

Inheritance



Public Members

5.1 **ClassLLCriterion** (ClassFormat* class_format)
The ClassFormat is needed just to know how the targets are encoded in the data 19

This criterion can be used to train *in classification* a `GradientMachine` object using the `GMTrainer` trainer. It then maximizes the log likelihood of the data.

If we write o_i for the output i of the `GradientMachine`, it supposes that

- the outputs o_i are log-probabilities.
- $exp(o_i)$ is the probability for the class i
- the predicted class follows a multinomial distribution with parameters $(exp(o_1), exp(o_2), exp(o_3)...)$

Author: Ronan Collobert (collober@iro.umontreal.ca)

5.1

ClassLLCriterion (ClassFormat* class_format)

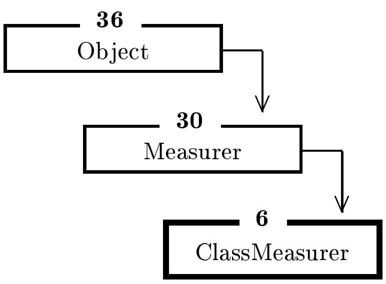
The ClassFormat is needed just to know how the targets are encoded in the data

The ClassFormat is needed just to know how the targets are encoded in the data

6 class **ClassMeasurer** : public Measurer

Compute the classification error (in %) of the `inputs` *with respect to the* `targets` *of* `data`.

Inheritance



Public Members

Compute the classification error (in %) of the `inputs` with respect to the `targets` of `data`.

The format of the class is given with a `ClassFormat`.

`measure` = <[real] error proportion>, <[real] current number of errors>

If the option "confusion matrix" is set, there is <[n_classes**2 int] confusion matrix>, <[n_classes**2 int] current confusion matrix> after that.

Options: "confusion matrix" bool compute the confusion matrix [false]

Author: Ronan Collobert (collober@iro.umontreal.ca)

6.1

ClassMeasurer (List* inputs_, DataSet* data_, ClassFormat* class_format_, FILE* file_)

6.2

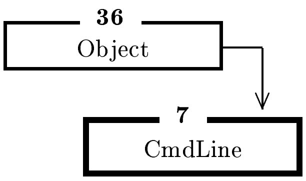
ClassMeasurer (List* inputs_, DataSet* data_, ClassFormat* class_format_, const char* filename)

7

class **CmdLine** : public Object

This class provides a useful interface for the user, to easily read some arguments/options from the command-line.

Inheritance



Public Members

This class provides a useful interface for the user, to easily read some arguments/options from the command-line.

Note that here, we make a difference between:

- **options** which aren't not required.
- **arguments** which are required.

Author: Ronan Collobert (collober@iro.umontreal.ca)

7.1

CmdLine ()

7.2

void **read** (int argc, char** argv)

Read the command-line.

Read the command-line. Call this function **after** adding options/arguments that you need, with the help of the following functions.

7.3

void **help** (char* name)

Print the help.

Print the help. Call this function **after** adding options/arguments that you need, with the help of the following functions.

7.4

Functions for adding options.

Functions for adding options. The calling order of the following functions will define the text order associated when you will call `help()`.

Add an option (Int, Bool, Real, String).

- `name` the name of the option (must be unique).
- `ptr` is the pointer on the optional variable.
- `initvalue` is the initialization value.
- `help` is the help text for this option.

The option will be setted to `value` in the command-line by printing "`name value`"

7.5

void **addBCmdOption** (const char* name, bool* ptr, bool initvalue, const char* help="")

7.6

void **addRCmdOption** (const char* name, real* ptr, real initvalue, const char* help="")

7.7

void **addSCmdOption** (const char* name, char** ptr, const char* initvalue, const char* help="")

7.8

Functions for adding an argument.

Functions for adding an argument. The argument will be setted to `value` in the command-line by writting "`value`" **after** all the options. If there are N arguments, you have to write "`value1` `value2` `value3` ... `valueN`" to set them in the command-line.

7.9

void **addBCmdArg** (const char* name, bool* ptr, const char* help="")

7.10

void **addRCmdArg** (const char* name, real* ptr, const char* help="")

7.11

void **addSCmdArg** (const char* name, char** ptr, const char* help="")

7.12

void **addText** (const char* text)

Add a text line in the help message

Add a text line in the help message

7.13

void **info** (const char* text)

Add a text at the beginnig of the help

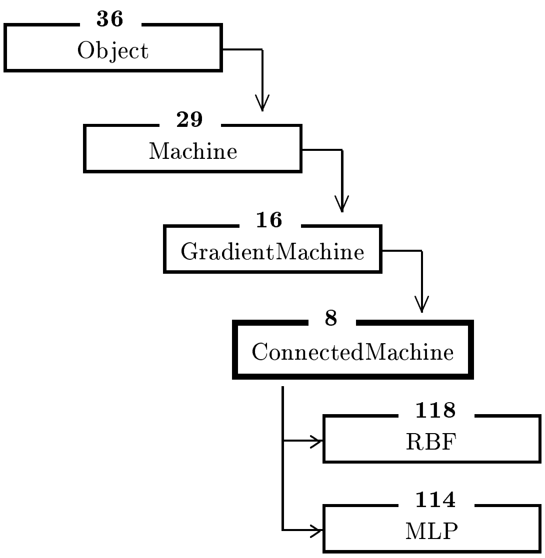
Add a text at the beginnig of the help

8

class **ConnectedMachine** : public GradientMachine

Easy connections between several `GradientMachine`.

Inheritance



Public Members

8.1		**ConnectedMachine** ()		28
8.2	void	**addFCL** (GradientMachine* machine)	*add a Full Connected Layer.*	28
8.3	void	**addMachine** (GradientMachine* machine)	*add a* `machine` *on the current layer*	28
8.4	void	**connectOn** (GradientMachine* machine)	*connect the last added machine on* `machine`*.*	29
8.5	void	**addLayer** ()	*add a layer (you don't have to call that for the first layer)*	29

Easy connections between several `GradientMachine`. GradientMachine has "layers" on which you can add some `GradientMachine`.

The inputs of the machine on the first layer will be the inputs of the `ConnectedMachine`.

The outputs of the `ConnectedMachine` are the union (in the order of adding) of all machines on the last layer.

Author: Ronan Collobert (collober@iro.umontreal.ca)

8.1

ConnectedMachine ()

8.2

void **addFCL** (GradientMachine* machine)

add a Full Connected Layer.

add a Full Connected Layer. The `machine` is fully connected to the previous layer. If necessary, a layer is added before adding the machine.

8.3

void **addMachine** (GradientMachine* machine)

add a `machine` *on the current layer*

add a `machine` on the current layer

8.4

void **connectOn** (GradientMachine* machine)

connect the last added machine on `machine`.

connect the last added machine on `machine`. Note that `machine` *must* be in a previous layer.

8.5

void **addLayer** ()

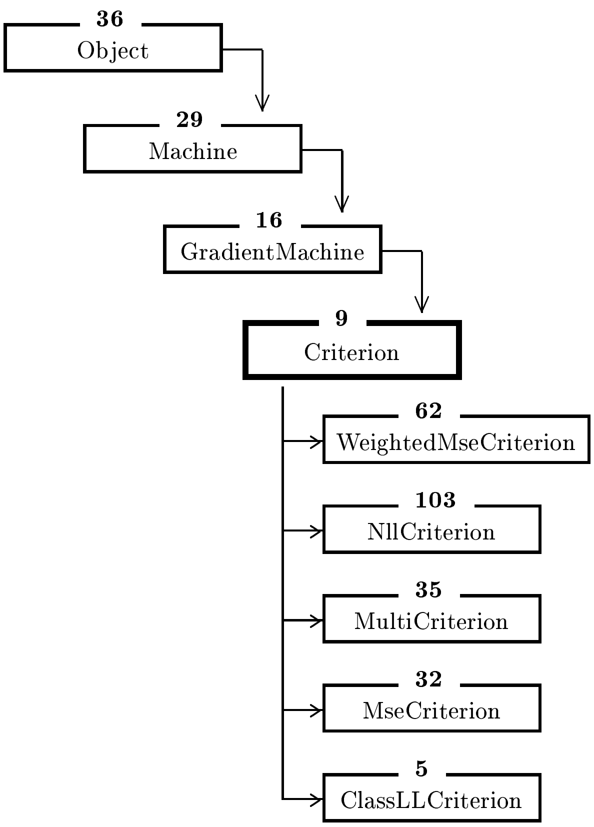
add a layer (you don't have to call that for the first layer)

add a layer (you don't have to call that for the first layer)

9 class **Criterion** : public GradientMachine

`Criterion` class for `GMTrainer`.

Inheritance



Public Members

`Criterion` class for `GMTrainer`. A `Criterion` computes an error with its inputs and a dataset `data`.

It knows how to backpropagate this error.

A `Criterion` shouldn't be connected to the inputs of another `GradientMachine`. ne doit pas se connecter sur l'entree d'une

By default, the number of outputs for a `Criterion` is fixed to one. It *must* contain the error: This one is used by the `GMTrainer` for the stopping criterion.

Author: Ronan Collobert (collober@iro.umontreal.ca)

9.1

DataSet* **data**

`DataSet` *used to compute the error*

`DataSet` used to compute the error

9.2

Criterion ()

9.3

virtual void **setDataSet** (DataSet* data_)

Set `data` *to* `data_`.

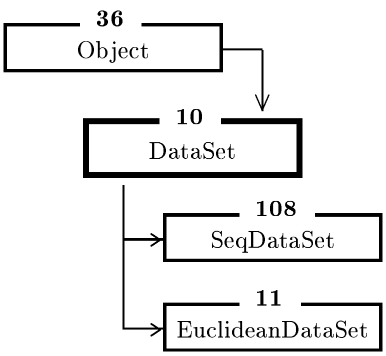
Set `data` to `data_`. The criterion should be able to react to this function.

10

class **DataSet** : public Object

Provides an interface to manipulate all kind of data.

Inheritance



Public Members

10.1	int*	**selected_examples**	*The indices of all selected examples.*	33
10.2	int	**current_example**	*Index of the current example.*	33
10.3	int	**n_targets**	*Targets size, for all examples*	34
10.4	void*	**targets**	*Pointer to the targets of the current example*	34
10.5	int	**n_inputs**	*Input size, for all examples*	34
10.6	List*	**inputs**	*Pointer on the inputs of the current example*	34
10.7	int	**n_examples**	*Number of examples in the dataset.*	35
10.8	int	**n_real_examples**	*Real number of examples in the dataset.*	35
10.9		**DataSet** ()		35
10.10	void	**setExample** (int t)		

Provides an interface to manipulate all kind of data.

Author: Ronan Collobert (collober@iro.umontreal.ca)

10.1

int* **selected_examples**

The indices of all selected examples.

The indices of all selected examples. When `select_examples` is false, it contains the indices of all examples.

10.2

int **current_example**

Index of the current example.

Index of the current example. Warning:

- don't forget to set it in `setRealExample()` !
- it's the *real* index and not the index in the `selected_examples` table.

10.3

int **n_targets**

Targets size, for all examples

Targets size, for all examples

10.4

void* **targets**

Pointer to the targets of the current example

Pointer to the targets of the current example

10.5

int **n_inputs**

Input size, for all examples

Input size, for all examples

10.6

List* **inputs**

Pointer on the inputs of the current example

Pointer on the inputs of the current example

10.7

int **n_examples**

Number of examples in the dataset.

Number of examples in the dataset. If you're using `select_examples`, it's the number of selected examples.

10.8

int **n_real_examples**

Real number of examples in the dataset.

Real number of examples in the dataset. It's the number of examples in memory. (= `n_examples` if `select_examples` is false)

10.9

DataSet ()

10.10

void **setExample** (int t)

Set `targets` *and* `inputs` *to the targets and inputs of the example with the index* `selected_examples[t]`

Set `targets` and `inputs` to the targets and inputs of the example with the index `selected_examples[t]`

10.11

virtual void **setRealExample** (int t)

Set `targets` *and* `inputs` *to the targets and inputs of the example with the index* `t`

Set `targets` and `inputs` to the targets and inputs of the example with the index `t`

10.12

void **pushSubset** (int* subset_, int n_examples_)

Set a new subset.

Set a new subset.

- `subset_` (of size `n_examples_`) is a set of indices which define a subset of `data`.
- if a `pushSubset()` has been already called, the next `pushSubset()` defines a subset of the previous subset, and so on...
- this function set `select_examples` to `true` and set the read indices of the examples in `selected_examples`.

10.13

void **popSubset** ()

Remove the last subset.

Remove the last subset.

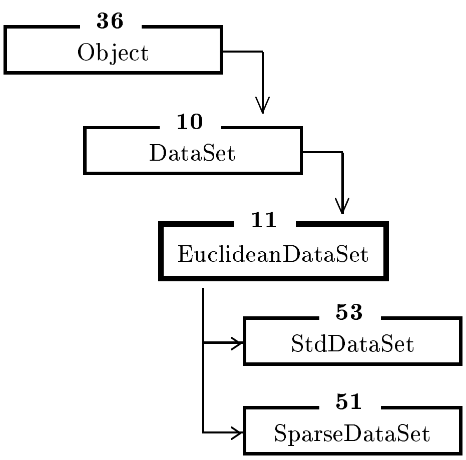
- recomputes "selected_examples".
- if it was the last subset, set `select_examples` to `false`.

11

class **EuclideanDataSet** : public DataSet

`DataSet` *with dot-products.*

Inheritance



Public Members

DataSet with dot-products.

Author: Ronan Collobert (collober@iro.umontreal.ca)

11.1

EuclideanDataSet ()

11.2

Dot product functions

Dot product functions

11.3

Dot products between two examples.

Dot products between two examples. Not that if a parameter is a List, this one *must* have the same format as the format generated by the DataSet class for inputs.

11.4

real **dotProduct** (int i, List* y)

Between selected_examples[i] *and* y

Between selected_examples[i] and y

11.5

real **realDotProduct** (int i, int j)

Between `i` *and* `selected_examples[j]`

Between `i` and `selected_examples[j]`

11.6

virtual real **realRealDotProduct** (int i, int j)

Between `i` *and* `j`

Between `i` and `j`

11.7

virtual real **realDotProduct** (int i, List* y)

Between `i` *and* `y`

Between `i` and `y`

11.8

virtual real **dotProduct** (List* x, List* y)

Between `x` *and* `y`

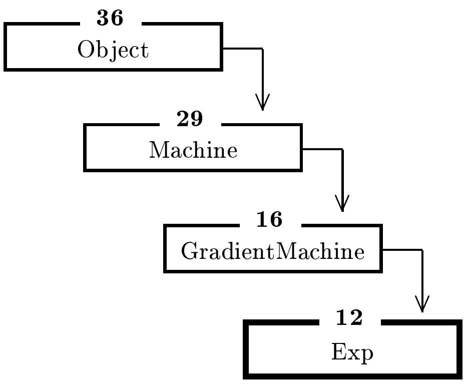
Between `x` and `y`

12

class **Exp** : public GradientMachine

Exponentiel layer for `GradientMachine`.

Inheritance



Public Members

Exponentiel layer for `GradientMachine`.

The number of inputs/outputs is the number of units for this machine. Formally speaking, $ouputs[i] = exp(inputs[i])$.

Author: Ronan Collobert (collober@iro.umontreal.ca)

12.1

Exp (int n_units)

Create a layer of size `n_units`

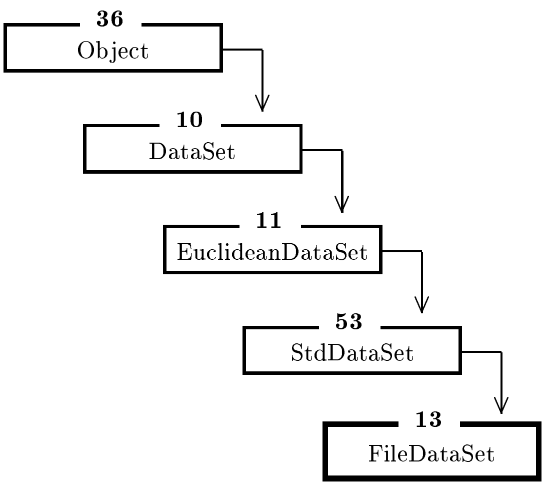
Create a layer of size `n_units`

13

class **FileDataSet** : public StdDataSet

Create a `DataSet` *from a disk file.*

Inheritance



Public Members

13.1 **FileDataSet** (const char* file, int n_inputs_, int n_targets, bool bin=false, int max_load=-1)
Load `file` *in memory.* 42

Create a `DataSet` from a disk file.

Author: Ronan Collobert (collober@iro.umontreal.ca)

13.1

FileDataSet (const char* file, int n_inputs_, int n_targets, bool bin=false, int max_load=-1)

Load `file` *in memory.*

Load `file` in memory.

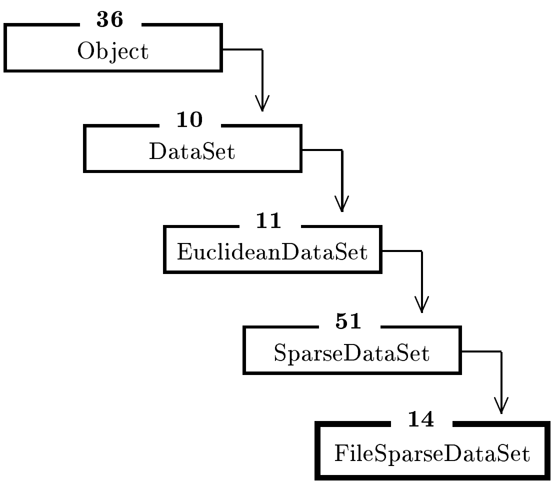
- `n_inputs_` is the input dimension of each example.
- `n_targets` is the target dimension of each example.
- `bin` is `true` if the file is in binary format.
- if `max_load` > 0, it loads only the first `max_load` examples.

14

class **FileSparseDataSet** : public SparseDataSet

Create a `DataSet` *from a* sparse *disk file.*

Inheritance



Public Members

Create a `DataSet` from a *sparse* disk file.

Author: Ronan Collobert (collober@iro.umontreal.ca)

14.1

FileSparseDataSet (const char* file, int n_inputs_, int n_targets, bool bin=false, int max_load=-1)

Load `file` *in memory.*

Load `file` in memory.

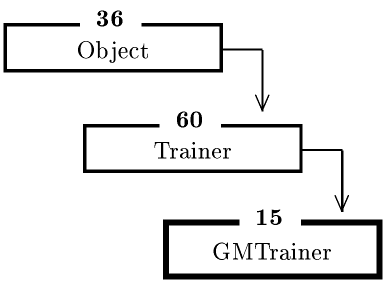
- `n_inputs_` is the input dimension of each example.
- `n_targets` is the target dimension of each example.
- `bin` is `true` if the file is in binary format.
- if `max_load` > 0, it loads only the first `max_load` examples.

15

class **GMTrainer** : public Trainer

Trainer for GradientMachine.

Inheritance



Public Members

Trainer for GradientMachine.

Author: Ronan Collobert (collober@iro.umontreal.ca)

15.1

GMTrainer (GradientMachine* machine_, DataSet* data_, Criterion* criterion_, Optimizer* optimizer_)

Take the `machine_` *to train, the train* `data_`*, the* `criterion_` *to use, and the* `optimizer_` *to use*

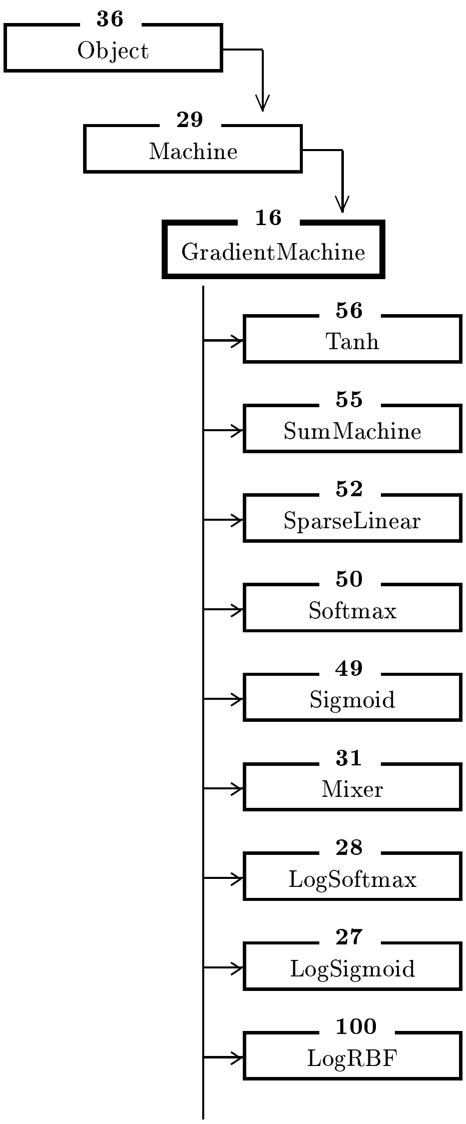
Take the `machine_` to train, the train `data_`, the `criterion_` to use, and the `optimizer_` to use

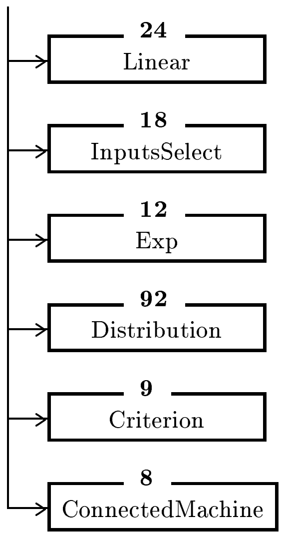
16

class **GradientMachine** : public Machine

Gradient machine: machine which can be trained with a gradient descent.

Inheritance





Public Members

16.1	bool	**is_free**	`true` *if* `freeMemory()` *has been already called*	49
16.2	List*	**params**	*Contains all parameters which will be updated with the gradient descent.*	50
16.3	List*	**der_params**	*Contains the derivatives for all parameters.*	50
16.4	int	**n_params**	*Contains the number of parameters.*	50
16.5	real*	**beta**	*Contains the derivative with respect to the inputs*	50
16.6		**GradientMachine** ()		51
16.7	virtual void	**init** ()	*It only calls* `allocateMemory()`	51
16.8	virtual int	**numberOfParams** ()	*Return the size of* `params`*.*	51
16.9	virtual void	**iterInitialize** ()	*This function is called before each training iteration.*	51
16.10	virtual void			

Gradient machine: machine which can be trained with a gradient descent.

Author: Ronan Collobert (collober@iro.umontreal.ca)

16.1

bool **is_free**

`true` *if* `freeMemory()` *has been already called*

`true` if `freeMemory()` has been already called

16.2

List* **params**

Contains all parameters which will be updated with the gradient descent.

Contains all parameters which will be updated with the gradient descent. Almost all machines will have only one node for this list. Note that the integer for each node is the number of parameters contained in the pointer of this node.

16.3

List* **der_params**

Contains the derivatives for all parameters.

Contains the derivatives for all parameters. Warning: `params` and `der_params` must have the same structure. (same number of nodes, same lenghts)

16.4

int **n_params**

Contains the number of parameters.

Contains the number of parameters. It's in fact the sum of all node size contained in `params` or `der_params`.

16.5

real* **beta**

Contains the derivative with respect to the inputs

Contains the derivative with respect to the inputs

16.6

GradientMachine ()

16.7

virtual void **init** ()

It only calls `allocateMemory()`

It only calls `allocateMemory()`

16.8

virtual int **numberOfParams** ()

Return the size of `params`.

Return the size of `params`. Note: it's the sum of the size in each node of `params`.

16.9

virtual void **iterInitialize** ()

This function is called before each training iteration.

This function is called before each training iteration. By default, do nothing.

16.10

virtual void **backward** (List* inputs, real* alpha)

Given the `inputs` *and the derivatives* `alpha` *with respect to the outputs, update* `beta` *and* `der_params`

Given the `inputs` and the derivatives `alpha` with respect to the outputs, update `beta` and `der_params`

16.11

virtual void **allocateMemory** ()

Allocate memory.

Allocate memory. By default, given `n_inputs`, `n_outputs` and `n_params`, allocate `beta`, `outputs`, `params` and `der_params`.

16.12

virtual void **freeMemory** ()

Free the memory allocated with `allocateMemory()` Do nothing *if* `is_free` *is* `true`.

Free the memory allocated with `allocateMemory()` *Do nothing* if `is_free` is `true`. Set `is_free` to `true`. This function *have to set* `n_params`. All classes which redefine this function should call it in the destructor.

16.13

virtual void **loadFILE** (FILE* file)

By default, load the `params`

By default, load the `params`

16.14

virtual void **saveFILE** (FILE* file)

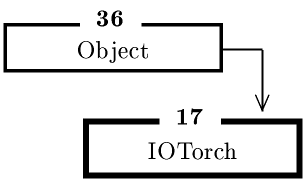
By default, save the `params`

By default, save the `params`

17 class **IOTorch** : public Object

Load and save file in the Torch format.

Inheritance



Public Members

Load and save file in the Torch format.

Author: Ronan Collobert (collober@iro.umontreal.ca)

17.1

IOTorch ()

17.2

int **loadData** (const char* file, real*** data_, real*** y_, int n_inputs, int n_targets, int* n_examples, bool bin=false, int max_load=-1)

Load the file `file`*.*

Load the file `file`. Each example must have `n_inputs` inputs and `n_targets` targets. `bin` is `true` if the file is in binary format. If `max_load` > 0, load only the first `max_load` examples. Returns:

- `data_` which contains all inputs (the memory allocation is done by this function)
- `y_` which contains all targets (the memory allocation is done by this function)
- `n_examples` which contains the real number of loaded examples.

17.3

int **loadData** (const char* file, sreal*** data_, real*** y_, int n_inputs, int n_targets, int* n_examples, bool bin=false, int max_load=-1)

Sparse version of the previous function

Sparse version of the previous function

17.4

void **saveData** (const char* file, real** data, real** y, int l, int n_inputs, int n_targets, bool bin=false, int max_save=-1)

Similar function, but to save data

Similar function, but to save data

17.5

void **saveData** (const char* file, sreal** data, real** y, int l, int n_inputs, int n_targets, bool bin=false, int max_save=-1)

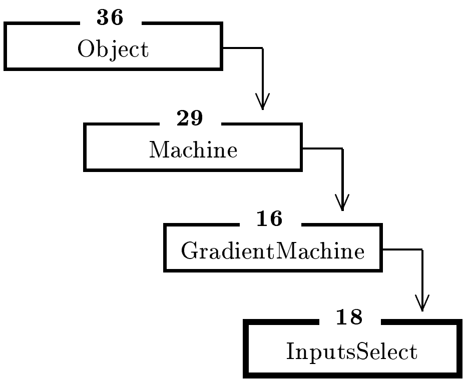
Similar function, but to save sparse data

Similar function, but to save sparse data

18 class **InputsSelect** : public GradientMachine

Machine which select a block of adjacent inputs, and put them in the outputs.

Inheritance



Public Members

18.1 **InputsSelect** (int n_inputs_, int offset_, int n_selected_)
Select `n_selected_` *inputs from* `offset_`. 57

Machine which select a block of adjacent inputs, and put them in the outputs.

Author: Ronan Collobert (collober@iro.umontreal.ca)

18.1 **InputsSelect** (int n_inputs_, int offset_, int n_selected_)

Select `n_selected_` *inputs from* `offset_`.

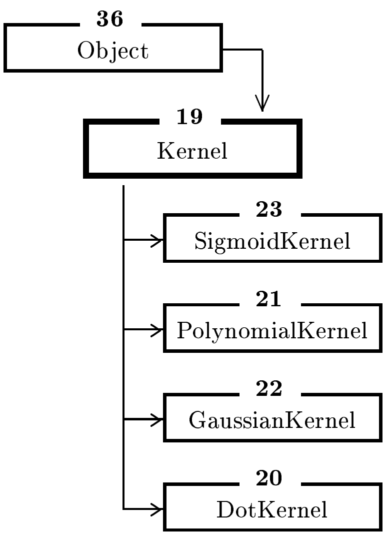
Select `n_selected_` inputs from `offset_`. (The input `offset_` will be included).

19

class **Kernel** : public Object

Kernel class.

Inheritance



Public Members

19.1 **Kernel** (DataSet* data_) 59

19.2 virtual real
eval (int i, int j) *Kernel between the example* `i` *and* `j` *of the dataset.* 59

19.3 virtual real
realEval (int i, List* y)
Kernel between the example `i` *of the dataset and one example* `y`*.* . 59

19.4 virtual void
reset () *Reset the kernel (Default, do nothing)* 59

Kernel class. Note that all kernels are based on the `dotProduct` of their associated `DataSet`.

Author: Ronan Collobert (collober@iro.umontreal.ca)

19.1

Kernel (DataSet* data_)

19.2

virtual real **eval** (int i, int j)

Kernel between the example `i` *and* `j` *of the dataset.*

Kernel between the example `i` and `j` of the dataset. This function takes in account the `selected_examples` of the dataset.

19.3

virtual real **realEval** (int i, List* y)

Kernel between the example `i` *of the dataset and one example* `y`*.*

Kernel between the example `i` of the dataset and one example `y`. The structure of `y` *must* be the same as those generated in `inputs` by the dataset `data`. Note that this function don't care about the `selected_examples` of the dataset: `i` is a *real* index.

19.4

virtual void **reset** ()

Reset the kernel (Default, do nothing)

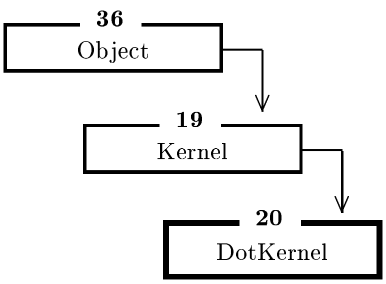
Reset the kernel (Default, do nothing)

20

class **DotKernel** : public Kernel

DotProduct

Inheritance



Public Members

DotProduct

Author: Ronan Collobert (collober@iro.umontreal.ca)

20.1

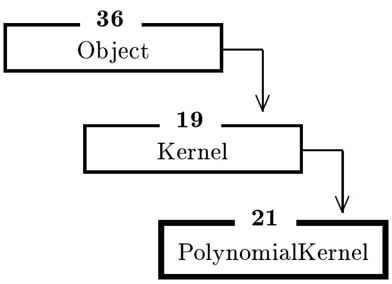
DotKernel (EuclideanDataSet* edata_)

21

class **PolynomialKernel** : public Kernel

Polynomial $k(x,y) = (s * xy + r)^d$.

Inheritance



Public Members

Polynomial $k(x,y) = (s * xy + r)^d$.

	"degree"	int	degree of the polynome	[2]
Options:	"mul cst"	real	s	[1]
	"add cst"	real	r	[1]

Author: Ronan Collobert (collober@iro.umontreal.ca)

21.1

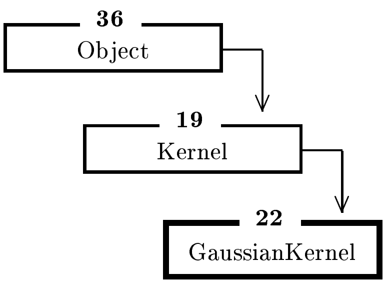
PolynomialKernel (EuclideanDataSet* edata_)

22

class **GaussianKernel** : public Kernel

$$Gaussian\ k(x,y) = exp(-g * ||x-y||^2)$$

Inheritance



Public Members

Gaussian $k(x,y) = exp(-g * ||x-y||^2)$

Options: "gamma" real g [0.01]

Author: Ronan Collobert (collober@iro.umontreal.ca)

22.1

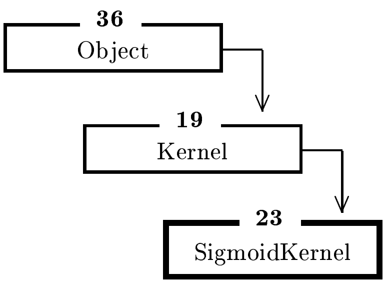
GaussianKernel (EuclideanDataSet* edata_)

23

class **SigmoidKernel** : public Kernel

Sigmoid $k(x,y) = tanh(s*xy+r)$

Inheritance



Public Members

Sigmoid $k(x,y) = tanh(s*xy+r)$

Options:			
"mul cst"	real	s	[1]
"add cst"	real	r	[1]

Author: Ronan Collobert (collober@iro.umontreal.ca)

23.1

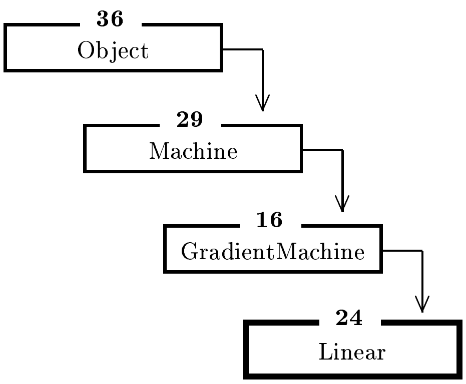
SigmoidKernel (EuclideanDataSet* edata_)

24

class **Linear** : public GradientMachine

Linear layer for `GradientMachine`.

Inheritance



Public Members

Linear layer for `GradientMachine`. Formally speaking, $ouputs[i] = \sum_j w_{ij} inputs[i] + b_i$.
w_{ij} and b_j are in `params`, with the following structure:
$w_0 0 ... w_0 n, b_0, w_1 0 ... w_1 n, b_1, ...$

If you want, you can add a weight decay which looks like $\sum_i, j w_{ij}^2 + sum_i b_i^2$.

Options: "weight decay" real the weight decay [0]

Author: Ronan Collobert (collober@iro.umontreal.ca)

24.1

Linear (int n_inputs_, int n_outputs_)

25 struct **List**

List structure used in all the library.

Members

List structure used in all the library.

Author: Ronan Collobert (collober@iro.umontreal.ca)

25.1 int **n**

Integer, which is not always used.

Integer, which is not always used. Usually, if `ptr` is a table, it contains the number of elements in this table

25.2 void* **ptr**

Pointer which can be used as you want

Pointer which can be used as you want

25.3

List* **next**

Next node, NULL if nothing

Next node, NULL if nothing

26 List functions

Names

List functions

26.1 void **addToList** (List** x, List* y)

Add `y` *to at the end of* `*x`.

Add `y` to at the end of `*x`. Note: if `*x` is empty, `*x` *must* be NULL.

Author: Ronan Collobert (collober@iro.umontreal.ca)

26.2 void **addToList** (List** x, int n, void* ptr)

Add the node `n`, `ptr` *at the end of* `*x`.

Add the node `n`, `ptr` at the end of `*x`. Note: if `*x` is empty, `*x` *must* be NULL.

Author: Ronan Collobert (collober@iro.umontreal.ca)

26.3

void **copyList** (List* to, List* from)

Copy the content of list `from` *into the list* `to`.

Copy the content of list `from` into the list `to`. This supposes that each list contains a vector of real in each node. The number of real which will be copied is given by `to`.

Author: Ronan Collobert (collober@iro.umontreal.ca)

26.4

void **copyList** (real* to, List* from)

Copy the content of list `from` *into the vector* `to`.

Copy the content of list `from` into the vector `to`. This supposes the list contains a vector of real in each node. The number of real which will be copied is given by `from`.

Author: Ronan Collobert (collober@iro.umontreal.ca)

26.5

void **freeList** (List** x, bool free_ptrs=false)

Destroy the list `*x`.

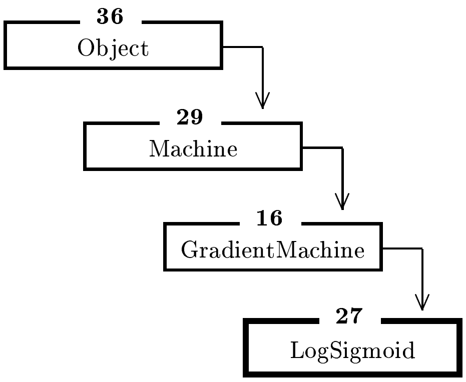
Destroy the list `*x`. It can free the memory inside each node (in `ptr`) if `free_ptrs` == `true`.

Author: Ronan Collobert (collober@iro.umontreal.ca)

27 class **LogSigmoid** : public GradientMachine

Log-sigmoid layer for `GradientMachine`.

Inheritance



Public Members

Log-sigmoid layer for `GradientMachine`. The number of inputs/outputs is the number of units of this machine. Formally speaking, $ouputs[i] = log(1/(1 + exp(-inputs[i])))$

Author: Ronan Collobert (collober@iro.umontreal.ca)

27.1 **LogSigmoid** (int n_units)

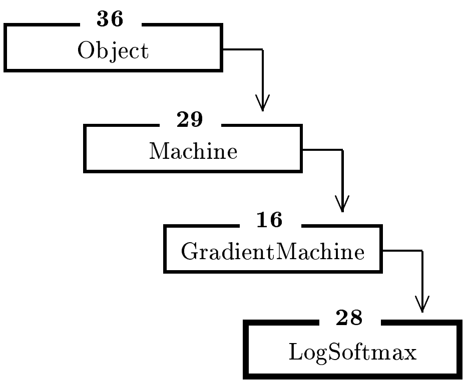
Create a layer with <n_units> units

Create a layer with <n_units> units

28 class **LogSoftmax** : public GradientMachine

LogSoftmax layer for `GradientMachine`.

Inheritance



Public Members

LogSoftmax layer for `GradientMachine`.

The number of inputs/outputs is the number of units in the machine.

Formally speaking, $ouputs[i] = log(1/a * exp(inputs[i]))$ where $a = \sum_j exp(inputs[j])$.

Author: Ronan Collobert (collober@iro.umontreal.ca)

28.1 **LogSoftmax** (int n_units)

Create a Sigmoid layer with `n_units` *units*

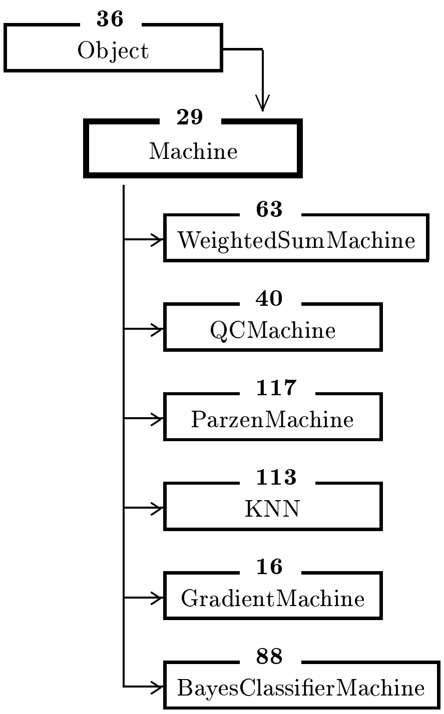
Create a Sigmoid layer with `n_units` units

29

class **Machine** : public Object

`Object` *which can compute some outputs, given some inputs.*

Inheritance



Public Members

29.1	int	**n_inputs**	*Number of inputs for the machine*	74
29.2	int	**n_outputs**	*Number of outputs*	74
29.3	List*	**outputs**	*The outputs of the machine.* ...	74
29.4		**Machine** ()	....	75
29.5	virtual void	**forward** (List* inputs)	*Given* `inputs`, *update* `outputs`.	75
29.6	virtual void			

`Object` which can compute some outputs, given some inputs.

Author: Ronan Collobert (collober@iro.umontreal.ca)

29.1

int **n_inputs**

Number of inputs for the machine

Number of inputs for the machine

29.2

int **n_outputs**

Number of outputs

Number of outputs

29.3

List* **outputs**

The outputs of the machine.

The outputs of the machine. Usually, the list contains only one node.

The integer in each node is the number of components in the pointer `ptr` of this node.

The sum of these integers is `n_outputs`.

29.4

Machine ()

29.5

virtual void **forward** (List* inputs)

Given `inputs`*, update* `outputs`*.*

Given `inputs`, update `outputs`. The integer in each node of `inputs` is the number of components in the pointer `ptr` of this node. The sum of these integers is `n_inputs`.

29.6

virtual void **reset** ()

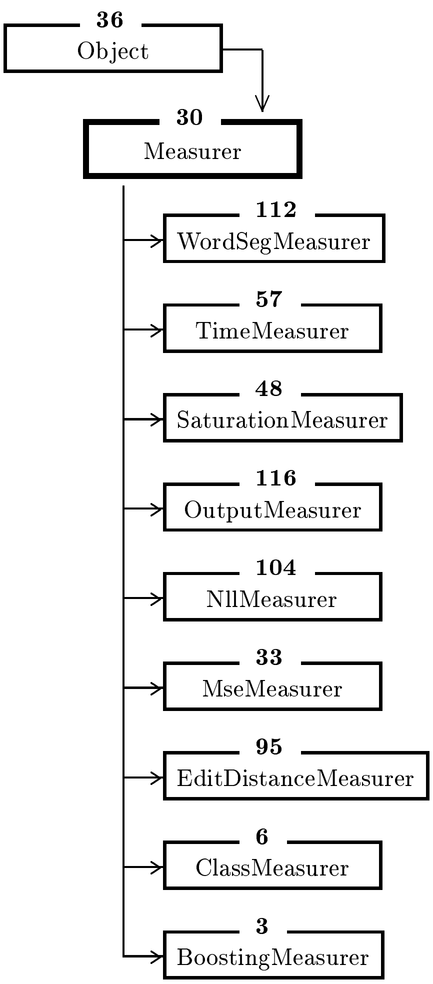
Reset the machine

Reset the machine

30 class **Measurer** : public Object

Used to measure what you want during training/testing.

Inheritance



Public Members

30.1	void*	**measure**	*The measurer put here what it measures.*	77

Used to measure what you want during training/testing. Usually, trainers call measurers. The `DataSet` associated to the measurer allow us to know when the measurer should be called. (if the `DataSet` is the train dataset, the measurer will be called during the train phase...)

Author: Ronan Collobert (collober@iro.umontreal.ca)

30.1

void* **measure**

The measurer put here what it measures.

The measurer put here what it measures. As it could be everything, you should know what's inside before using it.

30.2

FILE* **file**

The measurer save measures in this file

The measurer save measures in this file

30.3

DataSet* **data**

The associated `DataSet`

The associated `DataSet`

30.4

Measurer (DataSet* data_, FILE* file_)

Measurer with the associated `DataSet` `data_`*, and put results in the file* `file_`

Measurer with the associated `DataSet` `data_`, and put results in the file `file_`

30.5

Measurer (DataSet* data_, const char* filename)

Open the file `filename` *for writing measures.*

Open the file `filename` for writing measures. `data_` is the associated `DataSet`.

30.6

virtual void **measureEx** ()

Measure something for the current example.

Measure something for the current example. (This example has been selected in `data` by the trainer)

30.7

virtual void **measureIter** ()

Measure something after the current iteration.

Measure something after the current iteration. (After the call of `measureEx()` for each example of `data`)

30.8

virtual void **measureEnd** ()

Measure something at the end of the training/testing phase

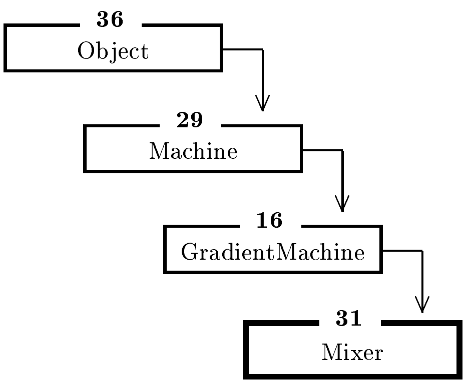
Measure something at the end of the training/testing phase

31

class **Mixer** : public GradientMachine

Mixer useful for experts mixtures.

Inheritance



Public Members

Mixer useful for experts mixtures. Formally speaking, it computes: $outputs[i] = \sum_j a_j * inputs_j[i]$

where

- $a_1, ..., a_n$ are in the table of the first node of the `inputs` list, when you call `forward()`.
- $inputs_j$ are the inputs of the j-th expert. Therefore, the `inputs` list has the structure $a, inputs_1, inputs_2, ...$ Only a must be alone in one node.

Author: Ronan Collobert (collober@iro.umontreal.ca)

31.1

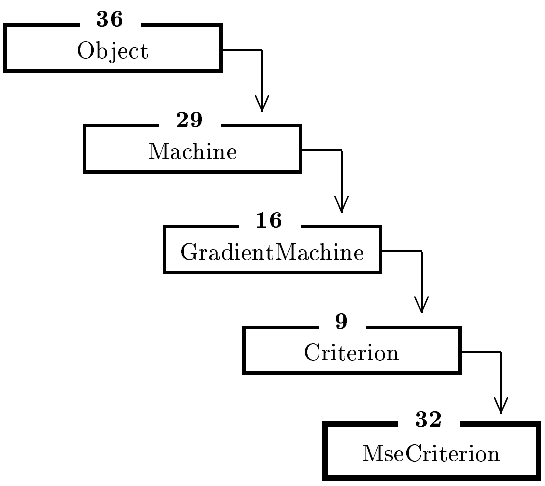
Mixer (int n_inputs_, int n_outputs_per_expert)

32

class **MseCriterion** : public Criterion

Mean Squared Error criterion.

Inheritance



Public Members

Mean Squared Error criterion. Compute the MSE between its inputs and, the targets of its associated `DataSet`.

Author: Ronan Collobert (collober@iro.umontreal.ca)

32.1

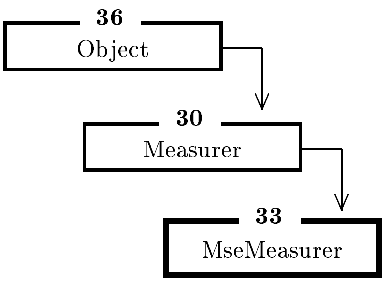
MseCriterion (int n_inputs_)

33

class **MseMeasurer** : public Measurer

Mean Squared Error measurer.

Inheritance



Public Members

Mean Squared Error measurer. Compute the MSE between its inputs, and the targets of its associated `DataSet`.

`measure` = <[real] previous MSE> <[real] current sum for computing MSE>

Author: Ronan Collobert (collober@iro.umontreal.ca)

33.1

MseMeasurer (List* inputs_, DataSet* data_, FILE* file_)

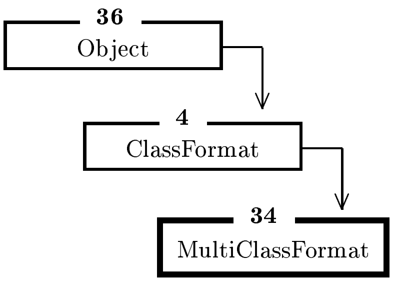
33.2

MseMeasurer (List* inputs_, DataSet* data_, const char* filename)

34 class **MultiClassFormat** : public ClassFormat

Define the multi class code.

Inheritance



Public Members

Define the multi class code. Each class is coded by a single number.

Author: Ronan Collobert (collober@iro.umontreal.ca)

34.1 real* **tabclasses**

Table of class labels

Table of class labels

34.2

MultiClassFormat (DataSet* data)

Autodetect classes

Autodetect classes

34.3

MultiClassFormat (int n_classes_, real* class_labels_=NULL)

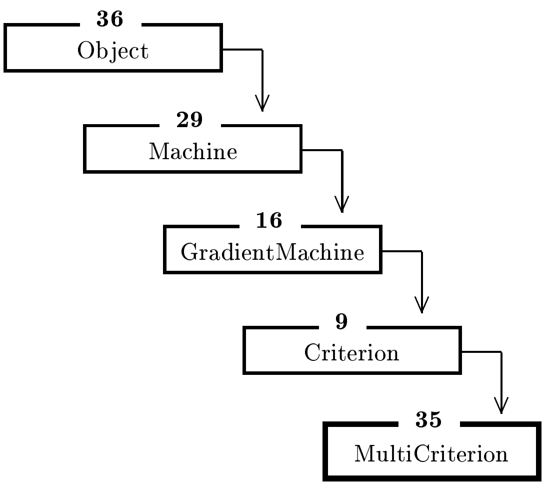
By default, class labels are 0, 1,

By default, class labels are 0, 1,

35 class **MultiCriterion** : public Criterion

MultiCriterion can be used to handle multiple criterions.

Inheritance



Public Members

MultiCriterion can be used to handle multiple criterions.

The `MultiCriterion` is a `Criterion` which is the weighted sum of given criterions. (By default, the weights are equal to `1`).

Author: Samy Bengio (bengio@idiap.ch)

35.1

int **n_criterions**

the number of criterions

the number of criterions

35.2

Criterion** **criterions**

the criterions

the criterions

35.3

real* **weights**

the relative weight of each criterion

the relative weight of each criterion

35.4

bool **allocate_weights**

do we need to allocate the weights

do we need to allocate the weights

35.5

MultiCriterion (int n_criterions_, Criterion** criterions_, real* weights_ = NULL)

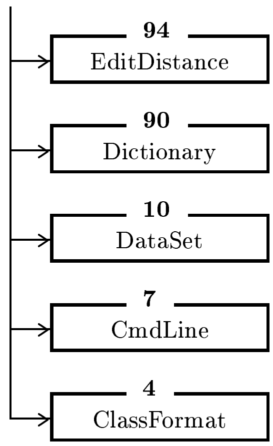
36

class **Object**

Provides a useful interface for managing options.

Inheritance

- **36** Object
  - **81** Vec
  - **60** Trainer
  - **39** QCCache
  - **79** Perm
  - **38** Optimizer
  - **30** Measurer
  - **77** Mat
  - **29** Machine
  - **19** Kernel
  - **17** IOTorch
  - **122** IOHtk
  - **97** Grammar



Public Members

36.1 **Object** () 91

36.2 virtual void **init** () *Initialize the object.* 91

36.3 void **addOption** (const char* name, int size, void* ptr, const char* help="", bool is_allowed_after_init=false)
Add the option `name`. 92

36.4 void **addIOption** (const char* name, int* ptr, int init_value, const char* help="", bool is_allowed_after_init=false)
Several wrappers of `addOption()` *for Int, Real and Bool options.* 92

36.5 void **addROption** (const char* name, real* ptr, real init_value, const char* help="", bool is_allowed_after_init=false)
.... 92

36.6 void **addBOption** (const char* name, bool* ptr, bool init_value, const char* help="", bool is_allowed_after_init=false)
.... 93

36.7 void **setOption** (const char* name, void* ptr)
Set the option `name` *to the value contained at* `ptr` 93

36.8 void **setIOption** (const char* name, int option)

Provides a useful interface for managing options. Class inherited by almost all classes in Torch.

Author: Ronan Collobert (collober@iro.umontreal.ca)

36.1

Object ()

36.2

virtual void **init** ()

Initialize the object.

Initialize the object. All objects *MUST* call the `init()` function of their parents.

36.3

void **addOption** (const char* name, int size, void* ptr, const char* help="", bool is_allowed_after_init=false)

Add the option `name`*.*

Add the option `name`. This one has a pointer on `ptr` and has the size `size`. You can provide an help in `help`. (Note that if `help` is empty, its an empty string and not NULL). If the option can be changed at any time, you can set `is_allowed_after_init` to `true`.

36.4

void **addIOption** (const char* name, int* ptr, int init_value, const char* help="", bool is_allowed_after_init=false)

Several wrappers of `addOption()` *for Int, Real and Bool options.*

Several wrappers of `addOption()` for Int, Real and Bool options. Be carefull: there is no type checking.

36.5

void **addROption** (const char* name, real* ptr, real init_value, const char* help="", bool is_allowed_after_init=false)

36.6

void **addBOption** (const char* name, bool* ptr, bool init_value, const char* help="", bool is_allowed_after_init=false)

36.7

void **setOption** (const char* name, void* ptr)

Set the option `name` *to the value contained at* `ptr`

Set the option `name` to the value contained at `ptr`

36.8

void **setIOption** (const char* name, int option)

Several wrappers of `setOption()` *for Int, Real and Bool options.*

Several wrappers of `setOption()` for Int, Real and Bool options. Be carefull: there is no type checking.

36.9

void **setROption** (const char* name, real option)

36.10

void **setBOption** (const char* name, bool option)

36.11

virtual void **loadFILE** (FILE* file)

*Load the object from a file pointer (*not the options*)*

Load the object from a file pointer (*not the options*)

36.12

virtual void **saveFILE** (FILE* file)

*Save the object to a file pointer (*not the options*)*

Save the object to a file pointer (*not the options*)

36.13

void **load** (const char* filename)

*Load the machine from a file name (*not the options*)*

Load the machine from a file name (*not the options*)

36.14

void **save** (const char* filename)

*Save the machine to a file name (*not the options*)*

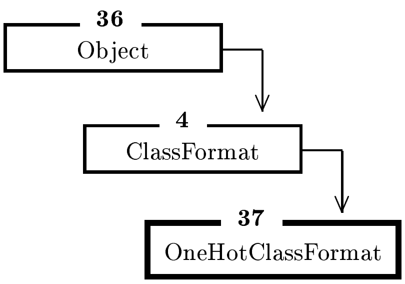
Save the machine to a file name (*not the options*)

37

class **OneHotClassFormat** : public ClassFormat

Define the one hot class code.

Inheritance



Public Members

Define the one hot class code.

Author: Ronan Collobert (collober@iro.umontreal.ca)

37.1

OneHotClassFormat (DataSet* data)

Autodetect

Autodetect

37.2

OneHotClassFormat (int n_targets)

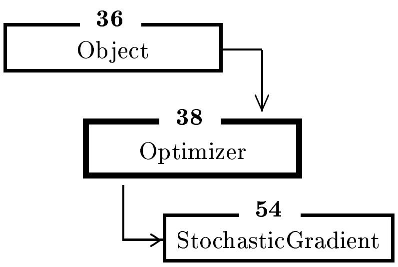
What's the number of classes ?

What's the number of classes ?

38 class **Optimizer** : public Object

Optimizer for the `GMTrainer` *class.*

Inheritance



Public Members

38.1		**Optimizer** ()	98
38.2	virtual void	**train** (List* measurers) *Optimize.*	98
38.3	void	**setMachine** (GradientMachine* machine_) *Set the* `machine`.	98
38.4	void	**setDataSet** (DataSet* data_) *Set the* `data`.	98
38.5	void	**setCriterion** (Criterion* criterion_) *Set the* `criterion`.	99

Optimizer for the `GMTrainer` class.

Given a `criterion`, a `machine` (`GradientMachine`) and a `DataSet` `data`, it optimizes the `machine` with respect to the `criterion` on the `data`.

See Also: GMTrainer (→15, *page 45*)
Author: Ronan Collobert (collober@iro.umontreal.ca)

38.1

Optimizer ()

38.2

virtual void **train** (List* measurers)

Optimize.

Optimize. The measurers `measurers` will be called during the optimization.

38.3

void **setMachine** (GradientMachine* machine_)

Set the `machine`.

Set the `machine`. This method could be called at every time before a call to `train()`.

38.4

void **setDataSet** (DataSet* data_)

Set the `data`.

Set the `data`. This method could be called at every time before a call to `train()`.

38.5

void **setCriterion** (Criterion* criterion_)

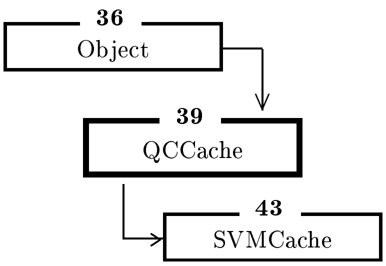
Set the `criterion`.

Set the `criterion`. This method could be called at every time before a call to `train()`.

39 class **QCCache** : public Object

"Cache" used by the Quadratic Constrained Trainer (`QCTrainer`).

Inheritance



Public Members

"Cache" used by the Quadratic Constrained Trainer (`QCTrainer`). This class provides the Q (symetric) matrix of the quadratic problem.

See Also: QCTrainer (→41, *page 106*)
Author: Ronan Collobert (collober@iro.umontreal.ca)

39.1

QCCache ()

39.2

virtual real* **adressCache** (int index)

Returns the adress of the row/column `index` *for the Q matrix*

Returns the adress of the row/column `index` for the Q matrix

39.3

virtual void **allocate** ()

Allocate the cache

Allocate the cache

39.4

virtual void **clear** ()

Erase the cache (but don't destroy it)

Erase the cache (but don't destroy it)

39.5

virtual void **destroy** ()

Destroy the cache

Destroy the cache

39.6

virtual void **setBoosterMode** (int* n_active_var_, int* active_var_)

The index of the active variables will be those contained in `active_var`*, and the number of active variable will be* `*n_active_var_`.

The index of the active variables will be those contained in `active_var`, and the number of active variable will be `*n_active_var_`.

(It's usefull for some problems)

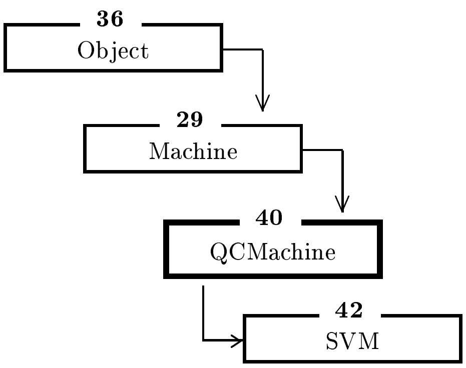
See Also: QCMachine
QCTrainer (→41, *page 106*)

40

class **QCMachine** : public Machine

"Quadratic Constrained Machine".

Inheritance



Public Members

"Quadratic Constrained Machine". It's a machine with `alpha` parameters which are the minimum of a constrained quadratic problem.

The problem:

minimize $QP(alpha) = alpha' * Q * alpha + beta' * alpha$

with the following constraints:

- $\sum_j alpha_j y_j = 1$
- $Cdown_j <= alpha_j <= Cup_j$

where y, $Cdown$, Cup are given by the user and $y_j = +-1$.

The number of `alpha` variable is determined here by `l`. (Therefore, it determines the size of de Cup, Cdown et y...)

Note: if $alpha_j$ is closed to Cup_j [ou $Cdown_j$] with the accuracy "eps bounds", $alpha_j$ is considered to be equal to Cup_j [ou $Cdown_j$].

Options: "eps bounds" real bound accuracy: [1E-4 in float, 1E-12 in double]

Author: Ronan Collobert (collober@iro.umontreal.ca)

40.1

real* **Cup**

40.2

real* **Cdown**

40.3

real **eps_bornes**

40.4

int **l**

40.5

real* **alpha**

40.6

real* **grad**

40.7

real* **y**

40.8

QCMachine ()

40.9

virtual void **checkSupportVectors** ()

Function called by `QCTrainer` *after the optimization.*

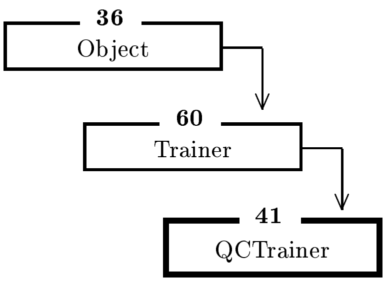
Function called by `QCTrainer` after the optimization.

See Also: QCTrainer (→41, *page 106*)

41 class **QCTrainer** : public Trainer

Train a `QCMachine`.

Inheritance



Public Members

Train a `QCMachine`. With the conventions of QCMachine.h, Q is given by the class QCCache (in `cache`)

Options:	"unshrink"	bool	unshrink or not unshrink	[false]
	"max unshrink"	int	maximal number of unshrinking	[1]
	"iter shrink"	int	minimal number of iterations to shrink	[100]
	"eps shrink"	real	shrinking accuracy	[1E-4 (f) 1E-9 (d)]
	"end accuracy"	real	end accuracy	[0.01]
	"iter message"	int	number of iterations between messages	[1000]

Note: "iter shrink" must be carefully chosen. Read http://www.ai.mit.edu/projects/jmlr/papers/volume1/collobert01a/collobert01a.ps.gz for more details.

See Also: QCCache
QCMachine (→40, *page 103*)

Author: Ronan Collobert (collober@iro.umontreal.ca)

41.1

QCTrainer (QCMachine* qcmachine_, DataSet* data_, QCCache* cache_)

41.2

void **train** (List* measurers)

Train it.

Train it. Before calling this function, `grad` in `qcmachine` must contain the gradient of QP(alpha) with respect to alpha = 0.

(= *beta*, with the conventions of QCMachine.h)

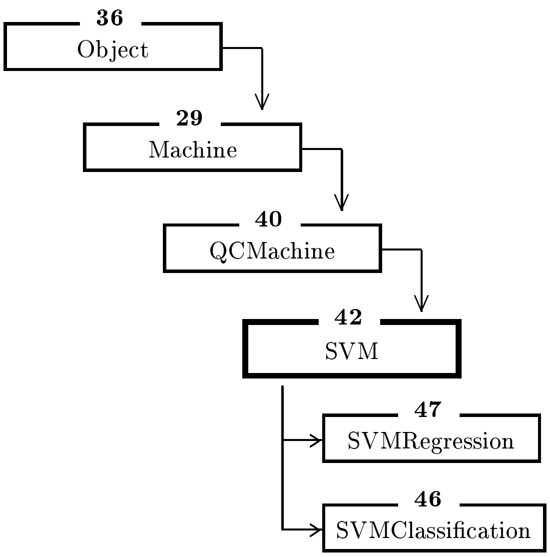
Moreover `alpha` in `qcmachine` has to be zero.

42

class **SVM** : public QCMachine

Support Vector Machine.

Inheritance



Public Members

42.1	DataSet*	**data**	*The dataset associated to the SVM.*	109
42.2	Kernel*	**kernel**	*The kernel associated to the SVM*	109
42.3	real	**C_cst**	*The "C" constant*	110
42.4	real	**b**		110
42.5	int*	**support_vectors**	*The support vectors*	110
42.6	int*	**real_index**	*The real index of the support vectors in the dataset*	110
42.7	int	**n_support_vectors**	*The number of support vectors*	111
42.8	int	**n_support_vectors_bound**		

Support Vector Machine.

The Q matrix of `QCMachine` is in this case $q_{ij} = k(x_i, x_j)$, where k is a kernel and x_i is the i-th example of `data`/

The goal is to looking for the `alpha` and `b` which are the best in a SVM-sense.

The learning function is $f(x) = \sum_j y_j alpha_j k(x_i, x) + b$

Options: "C" real trade off between the weight decay and the error [100]

Author: Ronan Collobert (collober@iro.umontreal.ca)

42.1

DataSet* **data**

The dataset associated to the SVM.

The dataset associated to the SVM. (Given by the kernel)

42.2

Kernel* **kernel**

The kernel associated to the SVM

The kernel associated to the SVM

42.3

real **C_cst**

The "C" constant

The "C" constant

42.4

real **b**

42.5

int* **support_vectors**

The support vectors

The support vectors

42.6

int* **real_index**

The real index of the support vectors in the dataset

The real index of the support vectors in the dataset

42.7

int **n_support_vectors**

The number of support vectors

The number of support vectors

42.8

int **n_support_vectors_bound**

The number of support vectors which are at the bound "C"

The number of support vectors which are at the bound "C"

42.9

SVM (Kernel* kernel_)

42.10

bool **bCompute** ()

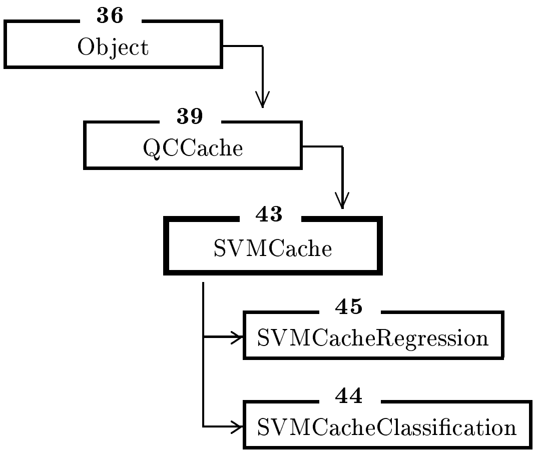
Computes the `b`

Computes the `b`

43 class **SVMCache** : public QCCache

`QCCache` *implementation for SVMs.*

Inheritance



Public Members

`QCCache` implementation for SVMs. Compute the Q matrix.

See Also: SVM
QCMachine
QCTrainer

Author: Ronan Collobert (collober@iro.umontreal.ca)

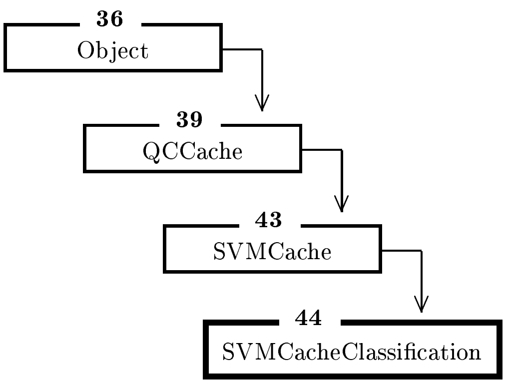
43.1 **SVMCache** (Kernel* kernel_, real taille_en_megs)

44

class **SVMCacheClassification** : public SVMCache

Cache for SVM classification.

Inheritance



Public Members

44.1 **SVMCacheClassification** (SVMClassification* svm, real taille_en_megs) 113

Cache for SVM classification.

Author: Ronan Collobert (collober@iro.umontreal.ca)

44.1

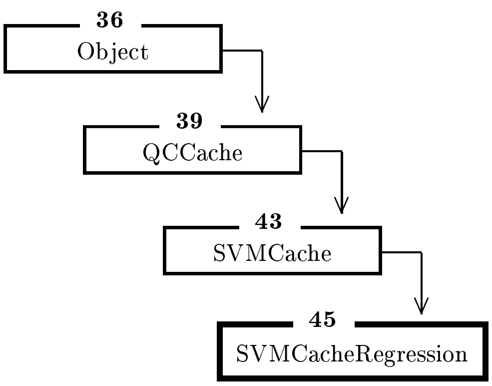
SVMCacheClassification (SVMClassification* svm, real taille_en_megs)

45

class **SVMCacheRegression** : public SVMCache

Cache for SVM regression.

Inheritance



Public Members

Cache for SVM regression.

Author: Ronan Collobert (collober@iro.umontreal.ca)

45.1

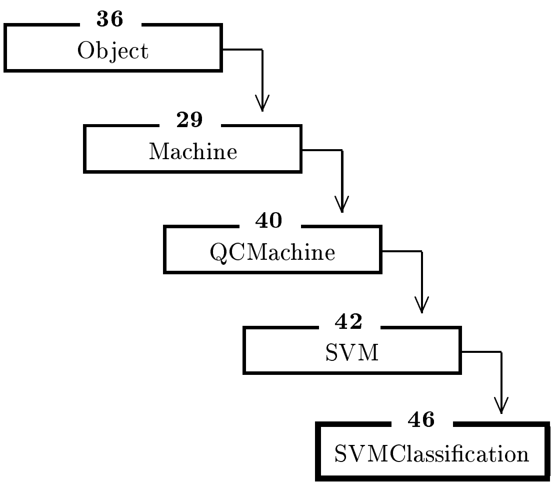
SVMCacheRegression (SVMRegression* svm, real taille_en_megs)

46

class **SVMClassification** : public SVM

SVM in classification.

Inheritance



Public Members

SVM in classification.

Try to find the hyperplane $w.x + b = 0$ as (w, b) minimize $0.5 * ||w||^2 + \sum_j C_j |1 - y_j * (w.x_j + b)|_+$

(where $|x|_+ = x$ if $x > 0$, else 0)

(in fact, we use a `kernel` instead of a dot product)

The C_j coefficients are given in `C_` when you call the constructor. If this one is NULL, then the value given by the "C" option is used for all C_j. (The size of `C_` *must be* `data->n_real_examples`)

Author: Ronan Collobert (collober@iro.umontreal.ca)

46.1

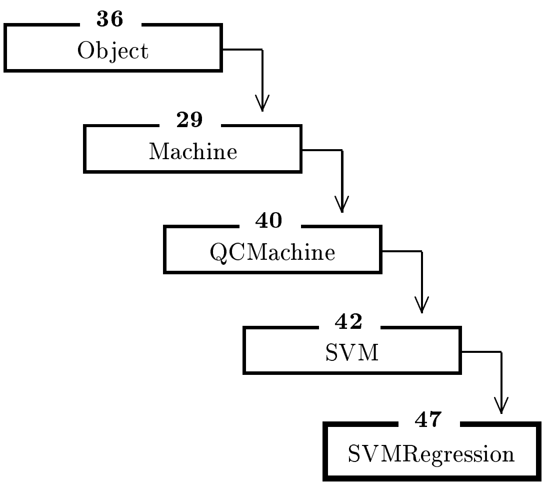
SVMClassification (Kernel* kernel_, real* C_=NULL)

47

class **SVMRegression** : public SVM

SVM in regression.

Inheritance



Public Members

SVM in regression.

Try to find the hyperplane f(x) = w.x+b as (w, b) minimize $0.5 * ||w||^2 + \sum_j C_j |w.x_j + b - y_j - eps|_+ + \sum_j C_(j+n) |y_j - w.x_j - b - eps|_+$

(where $|x|_+ = x$ if $x > 0$, else 0) (and n is the number of training examples) (the size of C is here 2*n)

("eps" is `eps_regression` in the code)

(in fact, we use a kernel `kernel` instead of a dot product)

The C_j coefficients are given by `C_` when you call the constructor. If this one is NULL, all `C_j` will have the value given by the "C" option. (The size of `C_` *must be* `2*data->n_real_examples`)

Options: "eps regression" real size of the error tube [0.7]

Author: Ronan Collobert (collober@iro.umontreal.ca)

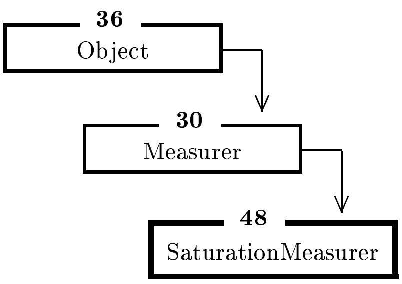
47.1

SVMRegression (Kernel* kernel_, real* C_=NULL)

48 class **SaturationMeasurer** : public Measurer

Measure the saturation of a `GradientMachine`.

Inheritance



Public Members

Measure the saturation of a `GradientMachine`.

Save in `file` (on two columns) - the absolute mean of the outputs of the machine. - the absolute mean of the derivative of the parameters of the machine.

`measure` = <[real] mean of outputs> <[real] mean of gradients> <[real] current sum of ouputs> <[real] current sum of gradients>

Author: Ronan Collobert (collober@iro.umontreal.ca)

48.1

SaturationMeasurer (GradientMachine* machine_, DataSet* data_, FILE* file_)

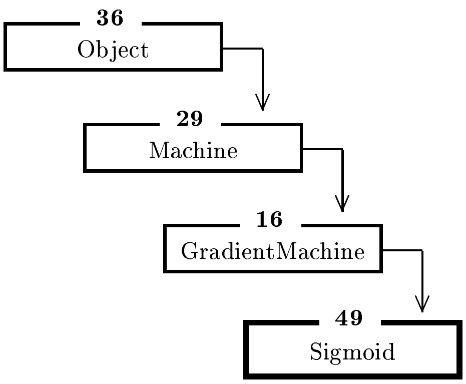
48.2

SaturationMeasurer (GradientMachine* machine_, DataSet* data_, const char* filename)

49 class **Sigmoid** : public GradientMachine

Sigmoid layer for `GradientMachine`.

Inheritance



Public Members

Sigmoid layer for `GradientMachine`.

The number of inputs/outputs is the number of units in the machine.

Formally speaking, $ouputs[i] = 1/(1 + exp(-inputs[i]))$

Author: Ronan Collobert (collober@iro.umontreal.ca)

49.1 **Sigmoid** (int n_units)

Create a Sigmoid layer with `n_units` *units*

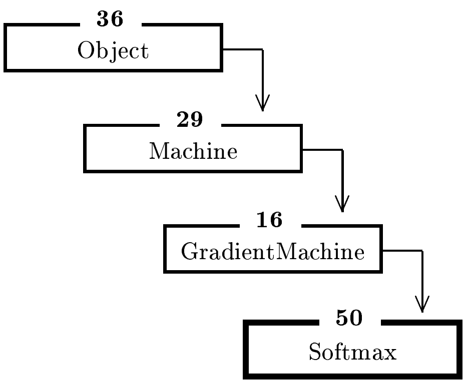
Create a Sigmoid layer with `n_units` units

50

class **Softmax** : public GradientMachine

Softmax layer for `GradientMachine`.

Inheritance



Public Members

50.1 **Softmax** (int n_units)
Create a Sigmoid layer with `n_units` *units* 123

Softmax layer for `GradientMachine`.

The number of inputs/outputs is the number of units in the machine.

Formally speaking, $ouputs[i] = 1/a * exp(inputs[i] - shift)$ where $a = \sum_j exp(inputs[j] - shift)$.

Options:				
	"shift"	real	shift to avoid overflow	[0]
	"compute shift"	bool	compute shift to avoid overflow	[false]

(you can have the "shift" you want, if you want, or you can automatically compute the shift)

Author: Ronan Collobert (collober@iro.umontreal.ca)

50.1

Softmax (int n_units)

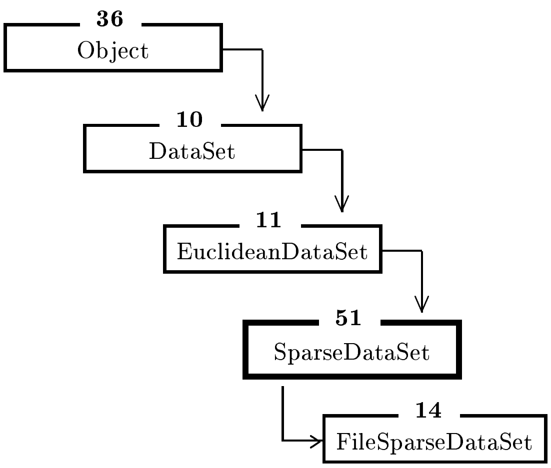
Create a Sigmoid layer with `n_units` *units*

Create a Sigmoid layer with `n_units` units

51 class **SparseDataSet** : public EuclideanDataSet

Sparse Data Set.

Inheritance



Public Members

Sparse Data Set.

DataSet with a vector of `sreals` for inputs, and a vector of real for targets.

Author: Ronan Collobert (collober@iro.umontreal.ca)

51.1

sreal** **all_inputs**

The inputs table

The inputs table

51.2

real** **all_targets**

The targets table

The targets table

51.3

SparseDataSet ()

51.4

void **setInputs** (sreal** data_, int n_dim, int n_examples_)

Set the inputs and the targets to a user tabular `n_dim` *is the dimension of inputs vectors* `n_targets_` *is the dimension of targets vectors.*

Set the inputs and the targets to a user tabular `n_dim` is the dimension of inputs vectors `n_targets_` is the dimension of targets vectors.

Call them BEFORE `init()`.

51.5

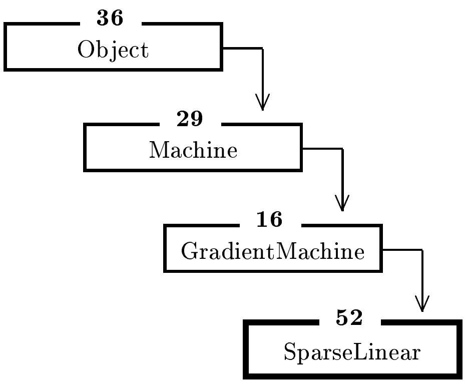
void **setTargets** (real** y_, int n_targets_)

52

class **SparseLinear** : public GradientMachine

Sparse Linear layer for `GradientMachine`.

Inheritance



Public Members

Sparse Linear layer for `GradientMachine`.

This is designed to use with a SparseDataset... The inputs of this layer must be sparse.

Formally speaking, $ouputs[i] = \sum_j w_{ij} inputs[i] + b_i$.
w_{ij} and b_j are in `params`, with the following structure:
$w_0 0 ... w_0 n, b_0, w_1 0 ... w_1 n, b_1, ...$

If you want, you can add a weight decay which looks like $\sum_i, jw_{ij}^2 + sum_i b_i^2$.

Options: "weight decay" real the weight decay [0]

Author: Ronan Collobert (collober@iro.umontreal.ca)

52.1

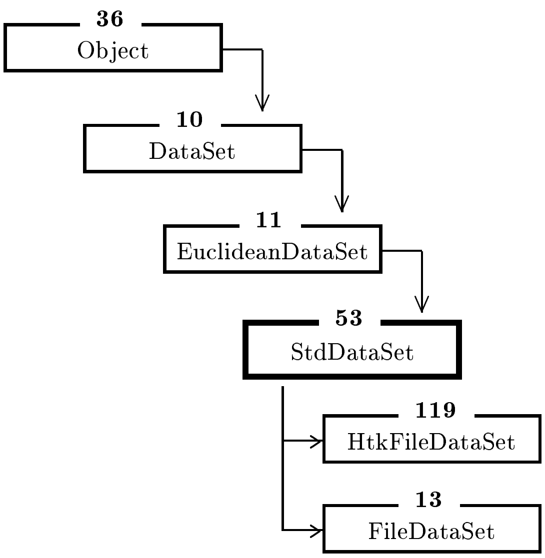
SparseLinear (int n_inputs_, int n_outputs_)

53

class **StdDataSet** : public EuclideanDataSet

Standard Data Set.

Inheritance



Public Members

53.1	real**	**all_inputs**	*The inputs table*	129
53.2	real**	**all_targets**	*The targets table*	130
53.3		**StdDataSet** ()		130
53.4	virtual void	**normalizeUsingDataSet** (StdDataSet* data_norm)	*Normalize the* `StdDataSet` *using the normalization values of another* `StdDataSet`.	130
53.5	void	**setInputs** (real** data_, int n_dim, int n_examples_)		

Standard Data Set.

DataSet with a vector of reals for inputs, and a vector of real for targets.

This DataSet support normalization with mean/variance.

Options:				
	"normalize inputs"	bool	normalize the inputs by mean/stdv	[false]
	"normalize targets"	bool	normalize the targets by mean/stdv	[false]

Author: Ronan Collobert (collober@iro.umontreal.ca)

53.1

real** **all_inputs**

The inputs table

The inputs table

53.2

real** **all_targets**

The targets table

The targets table

53.3

StdDataSet ()

53.4

virtual void **normalizeUsingDataSet** (StdDataSet* data_norm)

Normalize the `StdDataSet` *using the normalization values of another* `StdDataSet`.

Normalize the `StdDataSet` using the normalization values of another `StdDataSet`. If the other dataset is not normalize, nothing will be done.

Call it AFTER `init()`.

53.5

void **setInputs** (real** data_, int n_dim, int n_examples_)

Set the inputs and the targets to a user tabular `n_dim` *is the dimension of inputs vectors* `n_targets_` *is the dimension of targets vectors.*

Set the inputs and the targets to a user tabular `n_dim` is the dimension of inputs vectors `n_targets_` is the dimension of targets vectors.

Call them BEFORE `init()`.

53.6

void **setTargets** (real** y_, int n_targets_)

53.7

virtual void **loadFILE** (FILE* file)

Load the normalization parameters

Load the normalization parameters

53.8

virtual void **saveFILE** (FILE* file)

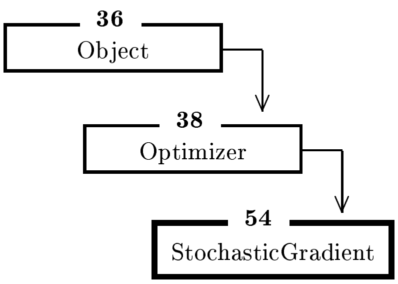
Save the normalization parameters

Save the normalization parameters

54 class **StochasticGradient** : public Optimizer

Stochastic Gradient Optimizer for GMTrainer

Inheritance



Public Members

Stochastic Gradient Optimizer for GMTrainer

Options:	"end accuracy"	real	end accuracy	[0.0001]
	"learning rate"	real	learning rate	[0.01]
	"learning rate decay"	real	learning rate decay	[0]
	"max iter"	int	maximum number of iterations	[-1]

See Also: GMTrainer
Author: Ronan Collobert (collober@iro.umontreal.ca)

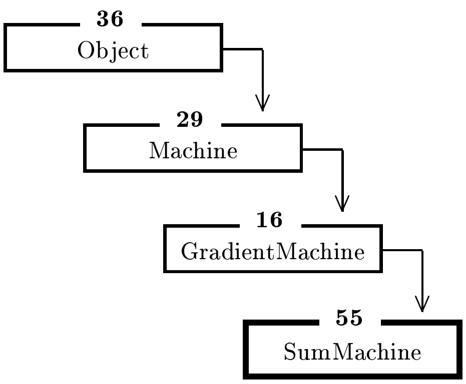
54.1 **StochasticGradient** ()

55

class **SumMachine** : public GradientMachine

This machine simply adds up its input vectors.

Inheritance



Public Members

This machine simply adds up its input vectors. Formally speaking, $outputs[i] = \sum_j inputs_j[i]$ where i spans the size of the vectors and j spans the number of input vectors.

Author: Samy Bengio (bengio@idiap.ch)

55.1

int **n_machines**

the number of input vectors

the number of input vectors

55.2

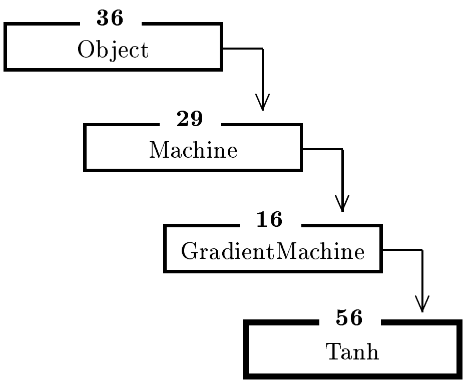
SumMachine (int n_units, int n_machines_)

56

class **Tanh** : public GradientMachine

Tanh layer for `GradientMachine`.

Inheritance



Public Members

Tanh layer for `GradientMachine`.

The number of inputs/outputs is the number of units for this machine. Formally speaking, $ouputs[i] = tanh(inputs[i])$.

Author: Ronan Collobert (collober@iro.umontreal.ca)

56.1

Tanh (int n_units)

Create a layer of size `n_units`

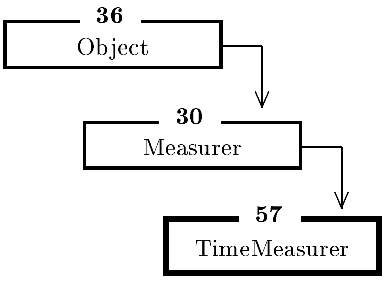
Create a layer of size `n_units`

57

class **TimeMeasurer** : public Measurer

Measure the time (in seconds) between two `measureIter()` *calls.*

Inheritance



Public Members

Measure the time (in seconds) between two `measureIter()` calls.

Save in `file` the sum of these time, at each iteration.

`measure` = [real] sum of time in sec

Author: Ronan Collobert (collober@iro.umontreal.ca)

57.1

TimeMeasurer (DataSet* data_, FILE* file_)

57.2

TimeMeasurer (DataSet* data_, const char* filename)

58

void **extractMeasurers** (List* measurers, DataSet* train, DataSet*** datas, Measurer**** mes, int** n_mes, int* n_datas)

Make a table of measurers from a `List`.

Make a table of measurers from a `List`.

Given a `List` of `measurers`, and, if you want, a `train` `DataSet` (else NULL)

- Returns all datasets associated to the measurers in `datas`. For i != j, (*datas)[i] != (*datas)[j]. Moreover, if `train` != NULL, (*datas)[0] = `train`.
- Returns the list of measurers associated to (*datas)[i] in (*mes)[i].
- Returns the number of measureurs associated to (*datas)[i] in (*n_mes)[i].
- Returns in *n_datas the number of datasets in *datas.

(Memory allocations are made by the fonction: use `deleteExtractedMeasurers()` to free memory)

See Also: deleteExtractedMeasurers
Author: Ronan Collobert (collober@iro.umontreal.ca)

59

void **deleteExtractedMeasurers** (DataSet** datas, Measurer*** mes, int* n_mes, int n_datas)

Free memory allocations did by `extractMeasurers()`.

Free memory allocations did by `extractMeasurers()`.

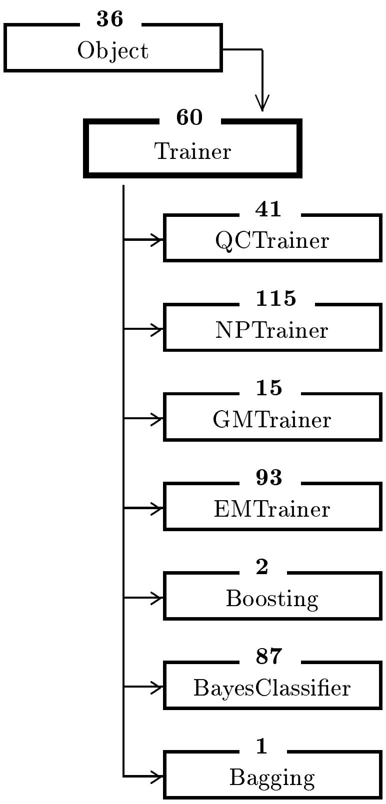
See Also: extractMeasurers
Author: Ronan Collobert (collober@iro.umontreal.ca)

60

class **Trainer** : public Object

Trainer.

Inheritance



Public Members

60.1 **Trainer** (Machine* machine_, DataSet* data_) 140

60.2 virtual void
train (List* measurers)
Train the machine. 140

60.3 virtual void

Trainer.

A trainer takes a `Machine` and a `DataSet`, and is able to train this machine on this dataset.

60.1

Trainer (Machine* machine_, DataSet* data_)

60.2

virtual void **train** (List* measurers)

Train the machine.

Train the machine. The Trainer has to call the measurers when it want.

60.3

virtual void **test** (List* measurers)

Test the machine.

Test the machine. This method call all the measurers, for all the examples of their associated dataset. It's already written...

60.4

virtual void **testExample** (List* measurers, int t)

Test on one example.

Test on one example. It supposes that all the measurers have the same dataset, and call the measurers for the example `t` of this dataset.

60.5

virtual void **crossValidate** (int k_fold, List* train_measurers, List* test_measurers, List* cross_valid_measurers=NULL)

K-fold cross-validation.

K-fold cross-validation. Do a K-fold cross-validation on `data`.

- `k_fold` is the number of subsets that it will make on `data`.
- `train_measurers` are the measurers called during the train phase. (NULL if nothing).
- `test_measurers` are the measurers called during the test phase.

- `cross_valid_measurers` are the measurers called after each iteration of the cross-validation. (these last measurers should'nt use `measureEx()`)

60.6

virtual void **loadFILE** (FILE* file)

Load the parameters of the machine and the dataset

Load the parameters of the machine and the dataset

60.7

virtual void **saveFILE** (FILE* file)

Save the parameters of the machine and the dataset

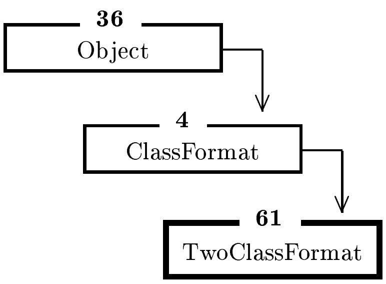
Save the parameters of the machine and the dataset

61

class **TwoClassFormat** : public ClassFormat

Define the two class code.

Inheritance



Public Members

Define the two class code.

Author: Ronan Collobert (collober@iro.umontreal.ca)

61.1

TwoClassFormat (DataSet* data)

Autodetect classes

Autodetect classes

61.2

TwoClassFormat (real class_1, real class_2)

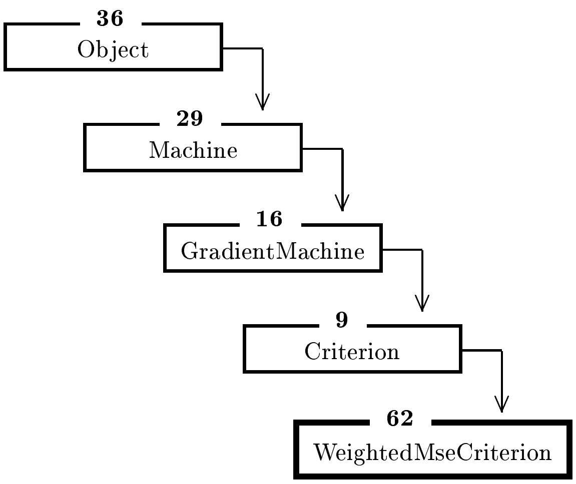
Give the label of the classes

Give the label of the classes

62 class **WeightedMseCriterion** : public Criterion

Similar to `MseCriterion`, but you can put a weight on each example.

Inheritance



Public Members

Similar to `MseCriterion`, but you can put a weight on each example.

Author: Ronan Collobert (collober@iro.umontreal.ca)

62.1

real* **weights**

Pointer on the weights.

Pointer on the weights. You can modify the weights when you want.

The length of `weights` *is* the `n_real_examples` of `data`.

62.2

WeightedMseCriterion (DataSet* data_)

With this constructor, the weights are allocated and initializated to 1

With this constructor, the weights are allocated and initializated to 1

62.3

WeightedMseCriterion (DataSet* data_, real* weights_)

With this constructor, you provide the `weights`.

With this constructor, you provide the `weights`.

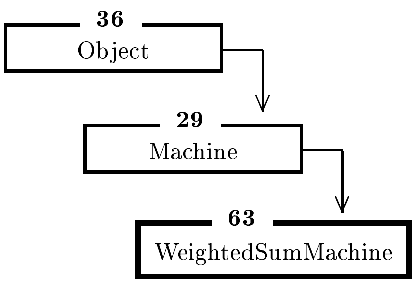
The length of `weights` *must be* the `n_real_examples` of `data`.

63

class **WeightedSumMachine** : public Machine

Weighted-sum machine.

Inheritance



Public Members

63.1	Trainer**	**trainers**	*The trainers used in the combination*	148
63.2	List**	**trainers_measurers**	*The corresponding measurers* ...	148
63.3	int	**n_trainers**	*The number of trainers in the combination*	148
63.4	int	**n_trainers_trained**	*The number of trainers that have been already trained.*	149
63.5	real*	**weights**	*The weights of the combination* .	149
63.6	bool	**weights_is_allocated**	*True if the weights aren't given by the user, false otherwise*	149
63.7		**WeightedSumMachine** (Trainer** trainer_, int n_trainers_, List** trainers_measurers, real* weights_=NULL)	....	149

Weighted-sum machine. This class contains a series of `Trainers`, and its forward method simply performs the average of the output of each trainer on the same input.

Author: Ronan Collobert (collober@iro.umontreal.ca)

63.1

Trainer** **trainers**

The trainers used in the combination

The trainers used in the combination

63.2

List** **trainers_measurers**

The corresponding measurers

The corresponding measurers

63.3

int **n_trainers**

The number of trainers in the combination

The number of trainers in the combination

63.4

int **n_trainers_trained**

The number of trainers that have been already trained.

The number of trainers that have been already trained. After the initialization, it's zero. Note that the forward method depends on this value. (only the first `n_trainers_trained` trainers are used)

63.5

real* **weights**

The weights of the combination

The weights of the combination

63.6

bool **weights_is_allocated**

True if the weights aren't given by the user, false otherwise

True if the weights aren't given by the user, false otherwise

63.7

WeightedSumMachine (Trainer** trainer_, int n_trainers_, List** trainers_measurers, real* weights_=NULL)

64 struct **sreal**

Sparse definition.

Members

Sparse definition.

Author: Ronan Collobert (collober@iro.umontreal.ca)

64.1 int **index**

Index which is not zero

Index which is not zero

64.2 real **value**

Value of this index

Value of this index

65

int **sparseVectorLength** (sreal* line)

Return the size of a sparse vector `line`.

Return the size of a sparse vector `line`.

Author: Ronan Collobert (collober@iro.umontreal.ca)

66

Several functions for Inputs/Ouputs on disk.

Names

Several functions for Inputs/Ouputs on disk.

Author: Ronan Collobert (collober@iro.umontreal.ca)

66.1

bool **isLittleEndian** ()

Returns `true` *if the computer use the Little Endian coding*

Returns `true` if the computer use the Little Endian coding

66.2

bool **isNativeMode** ()

Returns `true` *if we'll load/save using the native mode*

Returns `true` if we'll load/save using the native mode

66.3

void **setNativeMode** ()

We'll load/save in native mode

We'll load/save in native mode

66.4

void **setLittleEndianMode** ()

We'll load/save in Little Endian mode

We'll load/save in Little Endian mode

66.5

void **setBigEndianMode** ()

We'll load/save in Big Endian mode

We'll load/save in Big Endian mode

66.6

void **reverseMemory** (void* ptr_, int block_size, int n_blocks)

Reverse `n_blocks` *of memory starting from* `ptr_`*.*

Reverse `n_blocks` of memory starting from `ptr_`. In each block of size `block_size`, reverse the memory byte per byte.

66.7

int **xfread** (void* ptr, int block_size, int n_blocks, FILE* f)

As `fread()`*, but reverse the memory if it has been specified with* `setNativeMode()`*,* `setLittleEndianMode()` *or* `setBigEndianMode()`

As `fread()`, but reverse the memory if it has been specified with `setNativeMode()`, `setLittleEndianMode()` or `setBigEndianMode()`

66.8

int **xfwrite** (void* ptr, int block_size, int n_blocks, FILE* f)

As `fwrite()`*, but reverse the memory if it has been specified with* `setNativeMode()`*,* `setLittleEndianMode()` *or* `setBigEndianMode()`

As `fwrite()`, but reverse the memory if it has been specified with `setNativeMode()`, `setLittleEndianMode()` or `setBigEndianMode()`

67 Memory functions

Names

Memory functions

67.1 void* **xalloc** (int size)

As `malloc()`*, but send an error message if there is no more memory.*

As `malloc()`, but send an error message if there is no more memory.

Author: Ronan Collobert (collober@iro.umontreal.ca)

67.2 void* **xcalloc** (int n_blocks, int size)

As `calloc()`*, but send an error message if there is no more memory.*

As `calloc()`, but send an error message if there is no more memory.

Author: Ronan Collobert (collober@iro.umontreal.ca)

67.3

void* **xrealloc** (void* ptr, int size)

Almost as `realloc()`*, but send an error message if there is no more memory.*

Almost as `realloc()`, but send an error message if there is no more memory. Use `free()` is `size` is `0`.

Author: Ronan Collobert (collober@iro.umontreal.ca)

68 Text outputs functions.

Names

Text outputs functions.

These functions are like `printf()`, but you should use them instead. Note that you should never try to print a message larger than 10000 characters.

Author: Ronan Collobert (collober@iro.umontreal.ca)

68.1

void **error** (const char* msg, ...)

Print an error message.

Print an error message. The program will exit.

68.2

void **warning** (const char* msg, ...)

Print a warning message

Print a warning message

68.3

void **message** (const char* msg, ...)

Print a message

Print a message

68.4

void **print** (const char* msg, ...)

Like printf

Like printf

68.5

void **controlBar** (int level, int max_level)

Print a control bar [##########].

Print a control bar [##########].

First time, you can it with `level=-1`. It'll print the control bar at each time you will call that.

`max_level` is the value of the last `level` you'll call this function.

Author: Ronan Collobert (collober@iro.umontreal.ca)

69

long **getRuntime** (void)

Return the time in CLOCKS_PER_SEC

Return the time in CLOCKS_PER_SEC

Author: Ronan Collobert (collober@iro.umontreal.ca)

70

#define **min** (a,b)

The min function

The min function

71

#define **max** (a,b)

The max function

The max function

72 Some simple functions for log operations.

Names

Some simple functions for log operations.

Author: Samy Bengio (bengio@idiap.ch)

72.1 real **log_add** (real log_a, real log_b)

log_add(log_a,log_b) = log(a+b) = log(exp(log_a)+exp(log_b)) but done in a smart way so that if log_a or log_b are large but not their difference the computation works correctly

log_add(log_a,log_b) = log(a+b) = log(exp(log_a)+exp(log_b)) but done in a smart way so that if log_a or log_b are large but not their difference the computation works correctly

72.2 real **log_sub** (real log_a, real log_b)

log_sub(log_a,log_b) = log(a-b)

log_sub(log_a,log_b) = log(a-b)

73

void **MSTDVNormalize** (real** tab, real* mean_buff, real* std_buff, int n_ex, int n_dim)

Compute means and variances for normalizing a matrix.

Compute means and variances for normalizing a matrix. Returns in

- `mean_buff` means table of each columns of `tab`.
- `std_buff` the standard deviations table.

Note: the memory allocation is not made by this function.

`n_ex` is the number of examples in `tab`, and `n_dim` is the dimension of each example.

Author: Ronan Collobert (collober@iro.umontreal.ca)

74

void **MSTDVSparseNormalize** (sreal** tab, real* mean_buff, real* std_buff, int n_ex, int n_dim)

Compute means and variances for normalizing a sparse *matrix.*

Compute means and variances for normalizing a *sparse* matrix. Returns in

- `mean_buff` means table of each columns of `tab`.
- `std_buff` the standard deviations table.

Note: the memory allocation is not made by this function.

`n_ex` is the number of examples in `tab`, and `n_dim` is the dimension of each example.

Author: Ronan Collobert (collober@iro.umontreal.ca)

75

Random functions.

Names

Random functions.

Author: Samy Bengio (bengio@idiap.ch) Yoshua Bengio (bengioy@iro.umontreal.ca) Ronan Collobert (collober@iro.umontreal.ca)

75.1

real **gammln** (real xx)

Returns the natural logarithm of the gamma function

Returns the natural logarithm of the gamma function

75.2

void **getShuffledIndices** (int* indices, int n_indices)

Returns in `indices` `n_indices` *shuffled.*

Returns in `indices` `n_indices` shuffled. (between 0 and `n_indices`).

75.3

void **shuffle** (void* tabular, int size_elem, int n_elems)

Shuffles tabular, which contains `n_elems` *of size* `size_elem`

Shuffles tabular, which contains `n_elems` of size `size_elem`

75.4

void **seed** ()

Initializes the random number generator with the cpu time

Initializes the random number generator with the cpu time

75.5

void **manual_seed** (long x)

Initialzes the random number generator with the given long "x"

Initialzes the random number generator with the given long "x"

75.6

long **get_seed** ()

Returns the current seed used by the random number generator

Returns the current seed used by the random number generator

75.7

real **uniform** ()

Returns a random number uniformly distributed between 0 and 1

Returns a random number uniformly distributed between 0 and 1

75.8

real **bounded_uniform** (real a, real b)

Returns a random number uniformly distributed between a and b

Returns a random number uniformly distributed between a and b

75.9

real **expdev** ()

Returns an exponential distributed random number

Returns an exponential distributed random number

75.10

real **gaussian_01** ()

Returns a random number gaussian with mean 0 and standard deviation 1

Returns a random number gaussian with mean 0 and standard deviation 1

75.11

real **gaussian_mu_sigma** (real mu, real sigma)

Returns a random number gaussian with mean "mu" and standard dev "sigma"

Returns a random number gaussian with mean "mu" and standard dev "sigma"

75.12

real **gamdev** (int ia)

Returns a gamma distributed random number

Returns a gamma distributed random number

75.13

real **poidev** (real xm)

Returns a poisson random number with lambda = "xm"

Returns a poisson random number with lambda = "xm"

75.14

real **bnldev** (real pp, int n)

Returns a binomial random number with probability = 'pp' and trials number = 'n'

Returns a binomial random number with probability = 'pp' and trials number = 'n'

76 Some simple functions for string operations.

Names

Some simple functions for string operations.

Author: Samy Bengio (bengio@idiap.ch) Johnny Mariethoz (Johnny.Mariethoz@idiap.ch) Ronan Collobert (collober@iro.umontreal.ca)

76.1 char* **strBaseName** (char* filename)

Returns the name of a file without leading pathname.

Returns the name of a file without leading pathname. (It's not a new string, but a pointer in the given string)

76.2 char* **strRemoveSuffix** (char* filename, char c='.')

Returns a fresh copy of the name of a file without suffix.

Returns a fresh copy of the name of a file without suffix. (Trailing chars after c) You have to free the memory!

76.3

char* **strConcat** (int n, ...)

Returns the concatenation `n` *strings.*

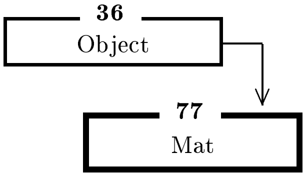
Returns the concatenation `n` strings. The strings are the parameters given after `n`; You have to free the memory!

77

class **Mat** : public Object

Matrix object.

Inheritance



Public Members

77.1	bool	**ptr_is_allocated**	`false` *if* `ptr` *is not allocated by Mat.*	174
77.2	int	**m**	*Size of the matrix*	174
77.3	real**	**ptr**	*Data of the matrix*	175
77.4	real*	**base**	*NULL if not allocated by Mat.*	175
77.5		**Mat** (real** ptr_, int n_rows, int n_cols)	*Create a matrix from values in* `ptr_`.	175
77.6		**Mat** (int n_rows, int n_cols)	*Create a new matrix*	175
77.7	void	**copy** (Mat* mat)	*Copy the matrix* `mat`	176
77.8	void	**zero** ()	*Zero the matrix*	176
77.9	real	**norm1** ()	*Compute the norm1*	176
77.10	real	**normFrobenius** ()	*Compute the Frobenius norm*	176
77.11	real	**normInf** ()	*Compute the norm inf*	177
77.12	Vec*	**getRow** (int row, Vec* vec = NULL)	*Return the row* `row` *of the matrix.*	177
77.13	Vec*	**getCol** (int col, Vec* vec = NULL)		

Matrix object.

Author: Ronan Collobert (collober@iro.umontreal.ca)

77.1

bool **ptr_is_allocated**

`false` *if* `ptr` *is not allocated by Mat.*

`false` if `ptr` is not allocated by Mat. (With the first constructor)

77.2

int **m**

Size of the matrix

Size of the matrix

77.3

real** **ptr**

Data of the matrix

Data of the matrix

77.4

real* **base**

NULL if not allocated by Mat.

NULL if not allocated by Mat. (when you're using the first constructor of Mat, or for the matrix returned by subMat)

77.5

Mat (real** ptr_, int n_rows, int n_cols)

Create a matrix from values in `ptr_`.

Create a matrix from values in `ptr_`. (No memory copy).

77.6

Mat (int n_rows, int n_cols)

Create a new matrix

Create a new matrix

77.7

void **copy** (Mat* mat)

Copy the matrix `mat`

Copy the matrix `mat`

77.8

void **zero** ()

Zero the matrix

Zero the matrix

77.9

real **norm1** ()

Compute the norm1

Compute the norm1

77.10

real **normFrobenius** ()

Compute the Frobenius norm

Compute the Frobenius norm

77.11

real **normInf** ()

Compute the norm inf

Compute the norm inf

77.12

Vec* **getRow** (int row, Vec* vec = NULL)

Return the row `row` *of the matrix.*

Return the row `row` of the matrix. If `vec` is NULL, return a new vector. Else copy the row in `vec`.

77.13

Vec* **getCol** (int col, Vec* vec = NULL)

Return the column `col` *of the matrix.*

Return the column `col` of the matrix. If `vec` is NULL, return a new vector. Else copy the column in `vec`.

77.14

void **setRow** (int row, Vec* vec)

Set the row `row` *to values in* `vec`

Set the row `row` to values in `vec`

77.15

void **setCol** (int row, Vec* vec)

Set the column `col` *to values in* `vec`

Set the column `col` to values in `vec`

77.16

Mat* **subMat** (int row1, int col1, int row2, int col2)

Return a sub-matrix.

Return a sub-matrix. Note that the memory is shared with the original matrix, so *be carefull*.

78

Collection of matrix operation functions.

Names

Collection of matrix operation functions. Based on the "Meschach Library", available at the anonymous ftp site thrain.anu.edu.au in the directory pub/meschach.

Author: David E. Stewart (david.stewart@anu.edu.au) Zbigniew Leyk (zbigniew.leyk@anu.edu.au) Ronan Collobert (collober@iro.umontreal.ca)

78.1

void **mxMatAddMat** (Mat* mat1, Mat* mat2, Mat* out)

Matrix addition – may be in-situ

Matrix addition – may be in-situ

78.2

void **mxMatSubMat** (Mat* mat1, Mat* mat2, Mat* out)

Matrix subtraction – may be in-situ

Matrix subtraction – may be in-situ

78.3

void **mxMatMulMat** (Mat* mat1, Mat* mat2, Mat* out)

Matrix-matrix multiplication

Matrix-matrix multiplication

78.4

void **mxMatMulTrMat** (Mat* mat1, Mat* mat2, Mat* out)

Matrix-matrix transposed multiplication.

Matrix-matrix transposed multiplication. – `A.B^T` is stored in out

78.5

void **mxTrMatMulMat** (Mat* mat1, Mat* mat2, Mat* out)

Matrix transposed-matrix multiplication.

Matrix transposed-matrix multiplication. – `A^T.B` is stored in out

78.6

void **mxMatMulVec** (Mat* mat, Vec* b, Vec* out)

Matrix-vector multiplication.

Matrix-vector multiplication. – Note: `b` is treated as a column vector

78.7

void **mxRealMulMat** (real scalar, Mat* matrix, Mat* out)

Scalar-matrix multiply – may be in-situ

Scalar-matrix multiply – may be in-situ

78.8

void **mxVecMulMat** (Vec* b, Mat* mat, Vec* out)

Vector-matrix multiplication.

Vector-matrix multiplication. – Note: `b` is treated as a row vector

78.9

void **mxTrMat** (Mat* in, Mat* out)

Transpose matrix

Transpose matrix

78.10

void **mxSwapRowsMat** (Mat* mat, int i, int j, int lo, int hi)

Swaps rows i and j of matrix A upto column lim.

Swaps rows i and j of matrix A upto column lim. `lo` and `hi` to -1 if you want to swap all

78.11

void **mxSwapColsMat** (Mat* mat, int i, int j, int lo, int hi)

Swap columns i and j of matrix A upto row lim.

Swap columns i and j of matrix A upto row lim. `lo` and `hi` to -1 if you want to swap all

78.12

void **mxMatAddRealMulMat** (Mat* mat1, Mat* mat2, real s, Mat* out)

Matrix-scalar multiply and add.

Matrix-scalar multiply and add. – may be in situ. – returns out == A1 + s*A2

78.13

void **mxVecAddRealMulMatMulVec** (Vec* v1, real alpha, Mat* mat, Vec* v2, Vec* out)

Matrix-vector multiply and add.

Matrix-vector multiply and add. – may not be in situ – returns out == v1 + alpha*A*v2

78.14

void **mxVecAddRealMulVecMulMat** (Vec* v1, real alpha, Vec* v2, Mat* mat, Vec* out)

*Vector-matrix multiply and add – may not be in situ – returns out' == v1' + alpha * v2'*A*

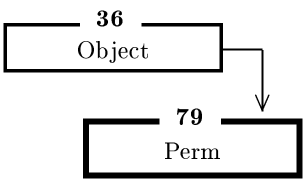
Vector-matrix multiply and add – may not be in situ – returns out' == v1' + alpha * v2'*A

79

class **Perm** : public Object

Permutation object.

Inheritance



Public Members

79.1	int	**n**	*Size of the permutation*	185
79.2	int*	**ptr**	*Data of the permutation*	186
79.3		**Perm** (int n_dim)	*Create a new permutation*	186
79.4	void	**copy** (Perm* perm)	*Copy the permutation* `perm`	186

Permutation object.

Author: Ronan Collobert (collober@iro.umontreal.ca)

79.1

int **n**

Size of the permutation

Size of the permutation

79.2

int* **ptr**

Data of the permutation

Data of the permutation

79.3

Perm (int n_dim)

Create a new permutation

Create a new permutation

79.4

void **copy** (Perm* perm)

Copy the permutation `perm`

Copy the permutation `perm`

80 Collection of permutations operation functions.

Names

Collection of permutations operation functions. Based on the "Meschach Library", available at the anonymous ftp site thrain.anu.edu.au in the directory pub/meschach.

Author: David E. Stewart (david.stewart@anu.edu.au) Zbigniew Leyk (zbigniew.leyk@anu.edu.au) Ronan Collobert (collober@iro.umontreal.ca)

80.1 void **mxPermInv** (Perm* px, Perm* out)

Invert permutation – in situ.

Invert permutation – in situ. Taken from ACM Collected Algorithms 250.

80.2

void **mxPermMulPerm** (Perm* px1, Perm* px2, Perm* out)

Permutation multiplication (composition) – not in-situ

Permutation multiplication (composition) – not in-situ

80.3

void **mxPermVec** (Perm* px, Vec* vec, Vec* out)

Permute vector – can be in-situ

Permute vector – can be in-situ

80.4

void **mxPermInvVec** (Perm* px, Vec* x, Vec* out)

Apply the inverse of px to x, returning the result in out.

Apply the inverse of px to x, returning the result in out. Can be in-situ, but "oh booooy!"...

80.5

void **mxTrPerm** (Perm* px, int i1, int i2)

Transpose elements of permutation.

Transpose elements of permutation. Really multiplying a permutation by a transposition. i1 and i2 are the elements to transpose.

80.6

void **mxPermColsMat** (Perm* px, Mat* mat, Mat* out)

Permute columns of matrix A; out = Apx'.

Permute columns of matrix A; out = Apx'. – May NOT be in situ

80.7

void **mxPermRowsMat** (Perm* px, Mat* mat, Mat* out)

Permute rows of matrix A; out = pxA.

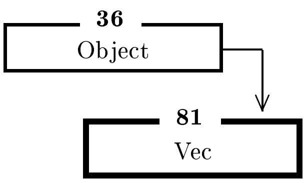
Permute rows of matrix A; out = pxA. – May NOT be in situ

81

class **Vec** : public Object

Vector object.

Inheritance



Public Members

81.1	bool	**ptr_is_allocated**	`false` *if* `ptr` *is not allocated by Vec.*	191
81.2	int	**n**	*Size of the vector*	191
81.3	real*	**ptr**	*Data of the vector*	191
81.4		**Vec** (real* ptr_, int n_dim)	*Create a vector from values in* `ptr_`.	192
81.5		**Vec** (int n_dim)	*Create a new vector*	192
81.6	void	**copy** (Vec* vec, int start_i = 0)	*Copy the vector* `vec` *starting from index* `start_i`	192
81.7	void	**zero** ()	*Zero the matrix*	192
81.8	real	**norm1** (Vec* weights = NULL)	*Compute the norm1*	193
81.9	real	**norm2** (Vec* weights = NULL)	*Compute the norm2*	193
81.10	real	**normInf** ()	*Compute the norm inf*	193
81.11	real	**iP** (Vec* vec, int start_i = 0)	*Inner product of two vectors from* `start_i` *downwards*	193
81.12	Vec*	**subVec** (int dim1, int dim2)		

Vector object.

Author: Ronan Collobert (collober@iro.umontreal.ca)

81.1

bool **ptr_is_allocated**

`false` *if* `ptr` *is not allocated by Vec.*

`false` if `ptr` is not allocated by Vec. (With the first constructor)

81.2

int **n**

Size of the vector

Size of the vector

81.3

real* **ptr**

Data of the vector

Data of the vector

81.4

Vec (real* ptr_, int n_dim)

Create a vector from values in `ptr_`.

Create a vector from values in `ptr_`. (No memory copy).

81.5

Vec (int n_dim)

Create a new vector

Create a new vector

81.6

void **copy** (Vec* vec, int start_i = 0)

Copy the vector `vec` *starting from index* `start_i`

Copy the vector `vec` starting from index `start_i`

81.7

void **zero** ()

Zero the matrix

Zero the matrix

81.8

real **norm1** (Vec* weights = NULL)

Compute the norm1

Compute the norm1

81.9

real **norm2** (Vec* weights = NULL)

Compute the norm2

Compute the norm2

81.10

real **normInf** ()

Compute the norm inf

Compute the norm inf

81.11

real **iP** (Vec* vec, int start_i = 0)

Inner product of two vectors from `start_i` *downwards*

Inner product of two vectors from `start_i` downwards

81.12

Vec* **subVec** (int dim1, int dim2)

Return a sub-vector.

Return a sub-vector. Note that the memory is shared with the original vector, so *be carefull*.

82

Givens matrix operations routines.

Names

Givens matrix operations routines. Routines for calculating and applying Givens rotations for/to vectors and also to matrices by row and by column.

Based on the "Meschach Library", available at the anonymous ftp site thrain.anu.edu.au in the directory pub/meschach.

Author: David E. Stewart (david.stewart@anu.edu.au) Zbigniew Leyk (zbigniew.leyk@anu.edu.au) Ronan Collobert (collober@iro.umontreal.ca)

82.1

void **mx_givens** (real x, real y, real* c, real* s)

Returns `c`, `s` *parameters for Givens rotation to eliminate* `y` *in the vector* `[ x y ]'`

Returns `c`, `s` parameters for Givens rotation to eliminate `y` in the vector `[ x y ]'`

82.2

void **mx_rot_vec** (Vec* x, int i, int k, real c, real s, Vec* out)

Apply Givens rotation to `x`*'s* `i` *and* `k` *components*

Apply Givens rotation to `x`'s `i` and `k` components

82.3

void **mx_rot_rows** (Mat* mat, int i, int k, real c, real s, Mat* out)

Premultiply `mat` *by givens rotation described by* `c`,`s`

Premultiply `mat` by givens rotation described by `c`,`s`

82.4

void **mx_rot_cols** (Mat* mat, int i, int k, real c, real s, Mat* out)

Postmultiply `mat` *by givens rotation described by* `c`,`s`

Postmultiply `mat` by givens rotation described by `c`,`s`

83

Routines for determining Hessenberg factorisations.

Names

Routines for determining Hessenberg factorisations.

Based on the "Meschach Library", available at the anonymous ftp site thrain.anu.edu.au in the directory pub/meschach.

Author: David E. Stewart (david.stewart@anu.edu.au) Zbigniew Leyk (zbigniew.leyk@anu.edu.au) Ronan Collobert (collober@iro.umontreal.ca)

83.1

void **mxHFactor** (Mat* mat, Vec* diag, Vec* beta)

Compute Hessenberg factorisation in compact form.

Compute Hessenberg factorisation in compact form. Factorisation performed in situ.

83.2

void **mxMakeHQ** (Mat* h_mat, Vec* diag, Vec* beta, Mat* q_out)

Construct the Hessenberg orthogonalising matrix Q.

Construct the Hessenberg orthogonalising matrix Q. i.e. Hess M = Q.M.Q'.

83.3

void **mxMakeH** (Mat* h_mat, Mat* h_out)

Construct actual Hessenberg matrix

Construct actual Hessenberg matrix

84

Householder transformation routines.

Names

Householder transformation routines. Contains routines for calculating householder transformations, applying them to vectors and matrices by both row and column.

Based on the "Meschach Library", available at the anonymous ftp site thrain.anu.edu.au in the directory pub/meschach.

Author: David E. Stewart (david.stewart@anu.edu.au) Zbigniew Leyk (zbigniew.leyk@anu.edu.au) Ronan Collobert (collober@iro.umontreal.ca)

84.1

void **mxHhVec** (Vec* vec, int i0, real* beta, Vec* out, real* newval)

Calulates Householder vector.

Calulates Householder vector. To eliminate all entries after the i0 entry of the vector vec. It is returned as out. May be in-situ.

84.2

void **mxHhTrVec** (Vec* hh, real beta, int i0, Vec* in, Vec* out)

Apply Householder transformation to vector.

Apply Householder transformation to vector. May be in-situ. (`hh` is the Householder vector).

84.3

void **mxHhTrRows** (Mat* mat, int i0, int j0, Vec* hh, real beta)

Transform a matrix by a Householder vector by rows.

Transform a matrix by a Householder vector by rows. Start at row i0 from column j0. In-situ.

85

Collection of matrix factorisation operation functions.

Names

Collection of matrix factorisation operation functions. Based on the "Meschach Library", available at the anonymous ftp site thrain.anu.edu.au in the directory pub/meschach.

Most matrix factorisation routines are in-situ unless otherwise specified.

Author: David E. Stewart (david.stewart@anu.edu.au) Zbigniew Leyk (zbigniew.leyk@anu.edu.au) Ronan Collobert (collober@iro.umontreal.ca)

85.1

void **mxLUFactor** (Mat* mat, Perm* pivot)

Gaussian elimination with scaled partial pivoting.

Gaussian elimination with scaled partial pivoting. – Note: returns LU matrix which is `A`.

85.2

void **mxLUSolve** (Mat* mat, Perm* pivot, Vec* b, Vec* x)

Given an LU factorisation in `A`*, solve* `Ax=b`

Given an LU factorisation in `A`, solve `Ax=b`

85.3

void **mxLUTSolve** (Mat* mat, Perm* pivot, Vec* b, Vec* x)

Given an LU factorisation in `A`*, solve* `A^Tx=b`

Given an LU factorisation in `A`, solve `A^Tx=b`

85.4

void **mxInverse** (Mat* mat, Mat* out)

Returns inverse of `A`*, provided* `A` *is not too rank deficient.*

Returns inverse of `A`, provided `A` is not too rank deficient. Uses LU factorisation.

86

Routines for symmetric eigenvalue problems.

Names

Routines for symmetric eigenvalue problems.

Based on the "Meschach Library", available at the anonymous ftp site thrain.anu.edu.au in the directory pub/meschach.

Author: David E. Stewart (david.stewart@anu.edu.au) Zbigniew Leyk (zbigniew.leyk@anu.edu.au) Ronan Collobert (collober@iro.umontreal.ca)

86.1

void **mxTriEig** (Vec* a, Vec* b, Mat* mat_q)

Finds eigenvalues of symmetric tridiagonal matrices.

Finds eigenvalues of symmetric tridiagonal matrices. The matrix is represented by a pair of vectors `a` (diag entries) and `b` (sub-diag and super-diag entries). Eigenvalues in `a` on return, and eigenvectors in `mat_q`, if this one is not `NULL`.

86.2

void **mxSymEig** (Mat* mat_a, Mat* mat_q, Vec* out)

Computes eigenvalues of a dense symmetric matrix.

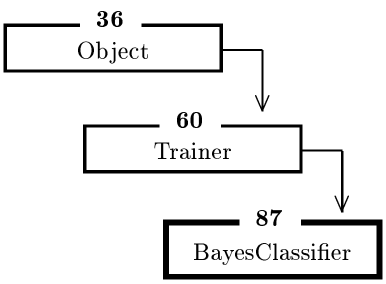
Computes eigenvalues of a dense symmetric matrix. `mat_a` *must* be symmetric on entry. Eigenvalues stored in `out`. `mat_q` contains orthogonal matrix of eigenvectors if not `NULL`.

87

class **BayesClassifier** : public Trainer

A multi class bayes classifier – maximizes the likelihood of each class separately using a trainer for distribution.

Inheritance



Public Members

87.1	BayesClassifierMachine*	**bayesmachine**	*the bayes machine*	206
87.2	int	**n_classes**	*the number of different classes* .	206
87.3	int**	**classes**	*all the example indices of each class*	206
87.4	int*	**classes_n**	*the number of examples per class*	206
87.5		**BayesClassifier** (BayesClassifierMachine*, DataSet*)	*you need to define a BayesClassifierMachine to use this class* ...	207

A multi class bayes classifier – maximizes the likelihood of each class separately using a trainer for distribution. When testing, the predicted class corresponds to the trainer giving the maximum output, weighted by its prior probability.

Author: Samy Bengio (bengio@idiap.ch) Bison Ravi (francois.belisle@idiap.ch)

87.1

BayesClassifierMachine* **bayesmachine**

the bayes machine

the bayes machine

87.2

int **n_classes**

the number of different classes

the number of different classes

87.3

int** **classes**

all the example indices of each class

all the example indices of each class

87.4

int* **classes_n**

the number of examples per class

the number of examples per class

87.5

BayesClassifier (BayesClassifierMachine*, DataSet*)

you need to define a BayesClassifierMachine to use this class

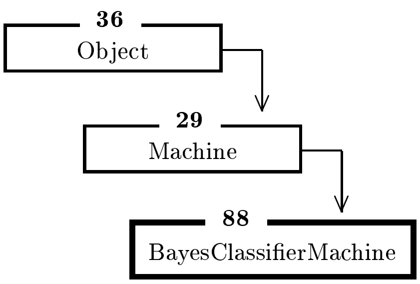
you need to define a BayesClassifierMachine to use this class

88

class **BayesClassifierMachine** : public Machine

BayesClassifierMachine is the machine used by the `BayesClassifier` *trainer to perform a Bayes Classification using different distributions.*

Inheritance



Public Members

88.1	int	**n_trainers**	*the number of classes corresponds to the number of* `Trainer`	209
88.2	Trainer**	**trainers**	*the actual trainers*	210
88.3	real*	**log_priors**	*the log_prior probabilities of each class.*	210
88.4	real*	**log_posteriors**	*contains the log posterior probability plus the log prior of the class*	210
88.5	bool	**allocated_log_priors**	*used to know if log_priors where given or allocated*	210
88.6	ClassFormat*	**class_format**	*the format of the data*	211
88.7	List**	**trainers_measurers**		

BayesClassifierMachine is the machine used by the `BayesClassifier` trainer to perform a Bayes Classification using different distributions. The output corresponds to the class that is the most probable (using prior AND posterior information).

Author: Samy Bengio (bengio@idiap.ch) Bison Ravi (francois.belisle@idiap.ch)

88.1

int **n_trainers**

the number of classes corresponds to the number of `Trainer`

the number of classes corresponds to the number of `Trainer`

88.2

Trainer** **trainers**

the actual trainers

the actual trainers

88.3

real* **log_priors**

the log_prior probabilities of each class.

the log_prior probabilities of each class. default: log_priors are taken as the log of the proportions in the training set.

88.4

real* **log_posteriors**

contains the log posterior probability plus the log prior of the class

contains the log posterior probability plus the log prior of the class

88.5

bool **allocated_log_priors**

used to know if log_priors where given or allocated

used to know if log_priors where given or allocated

88.6

ClassFormat* **class_format**

the format of the data

the format of the data

88.7

List** **trainers_measurers**

the measurers for each individual trainer

the measurers for each individual trainer

88.8

BayesClassifierMachine (Trainer**, int n_trainers_, List** trainers_measurers_, ClassFormat* class_format_, real* log_priors_=NULL)

creates a machine for BayesClassifier trainers, given a vector of trainers (one per class), an associate measurer for each trainer, a class_format that explains how the classes are coded, and an eventual vector (of size `n_trainers_`*) containing the log of the class priors*

creates a machine for BayesClassifier trainers, given a vector of trainers (one per class), an associate measurer for each trainer, a class_format that explains how the classes are coded, and an eventual vector (of size `n_trainers_`) containing the log of the class priors

88.9

virtual void **forward** (List* inputs)

definition of virtual functions of `Machine`

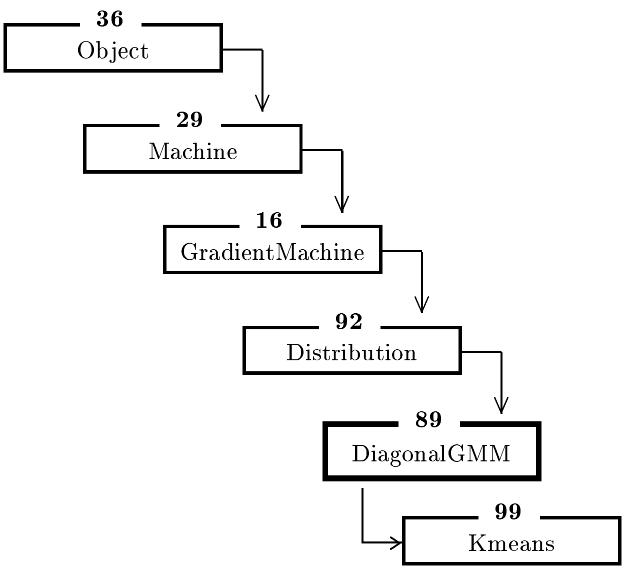
definition of virtual functions of `Machine`

89

class **DiagonalGMM** : public Distribution

This class can be used to model Diagonal Gaussian Mixture Models.

Inheritance



Public Members

89.1	int	**n_gaussians**	*number of Gaussians in the mixture*	215
89.2	real	**prior_weights**	*prior weights of the Gaussians, used in EM to give a small prior on each Gaussian*	215
89.3	EMTrainer*	**initial_kmeans_trainer**	*optional initializations if nothing is given, then random, at your own risks.*	215
89.4	List*	**initial_kmeans_trainer_measurers**		

This class can be used to model Diagonal Gaussian Mixture Models. They can be trained using either EM (with EMTrainer) or gradient descent (with GMTrainer).

Author: Samy Bengio (bengio@idiap.ch)

89.1

int **n_gaussians**

number of Gaussians in the mixture

number of Gaussians in the mixture

89.2

real **prior_weights**

prior weights of the Gaussians, used in EM to give a small prior on each Gaussian

prior weights of the Gaussians, used in EM to give a small prior on each Gaussian

89.3

EMTrainer* **initial_kmeans_trainer**

optional initializations if nothing is given, then random, at your own risks.

optional initializations if nothing is given, then random, at your own risks. one can give a initial trainer containing a kmeans

89.4

List* **initial_kmeans_trainer_measurers**

as well as a measurer of this trainer

as well as a measurer of this trainer

89.5

List* **initial_params**

or one can give an initial parameter List

or one can give an initial parameter List

89.6

char* **initial_file**

or one can give an initial file

or one can give an initial file

89.7

real* **log_weights**

the pointers to the parameters

the pointers to the parameters

89.8

real* **dlog_weights**

the pointers to the derivative of the parameters

the pointers to the derivative of the parameters

89.9

real* **var_threshold**

this contains the minimal value of each variance

this contains the minimal value of each variance

89.10

real** **log_probabilities_g**

for each frame, for each gaussian, keep its log probability

for each frame, for each gaussian, keep its log probability

89.11

real* **sum_log_var_plus_n_obs_log_2_pi**

*in order to faster the computation, we can do some "pre-computation" pre-computed sum_log_var + n_obs * log_2_pi*

in order to faster the computation, we can do some "pre-computation" pre-computed sum_log_var + n_obs * log_2_pi

89.12

real** **minus_half_over_var**

pre-computed -05 / var

pre-computed -05 / var

89.13

real** **means_acc**

accumulators for EM

accumulators for EM

89.14

DiagonalGMM (int n_observations_, int n_gaussians_, real* var_threshold_, real prior_weights_)

89.15

virtual real **frameLogProbabilityOneGaussian** (real* observations, real* inputs, int g)

this method returns the log probability of the "g" Gaussian

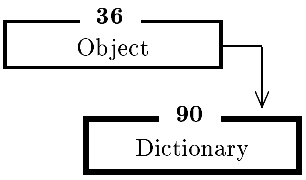
this method returns the log probability of the "g" Gaussian

90

class **Dictionary** : public Object

This class contains the dictionary of accepted words for a speech recognition experiment, such as the one used by SpeechHMM.

Inheritance



Public Members

This class contains the dictionary of accepted words for a speech recognition experiment, such as the one used by SpeechHMM. It contains a list of words, each word being a list of phonemes.

Author: Samy Bengio (bengio@idiap.ch)

90.1

int** **words**

the vector of words, which are represented by vectors of phonemes

the vector of words, which are represented by vectors of phonemes

90.2

int* **word_length**

the length of each word (number of phonemes)

the length of each word (number of phonemes)

90.3

int **n_words**

the number of words in the dictionary

the number of words in the dictionary

90.4

char** **words_string**

for each word, the literal transcription

for each word, the literal transcription

90.5

int **silence_word**

the index of the "silence" word

the index of the "silence" word

90.6

char** **phonemes**

the phonemes

the phonemes

90.7

int **n_phonemes**

the number of different phonemes

the number of different phonemes

90.8

Dictionary ()

90.9

Dictionary (char* filename, char** phonemes, int n_phonemes)

creates a Dictionary given a filename containing the words, as well as a vector of phonemes

creates a Dictionary given a filename containing the words, as well as a vector of phonemes

90.10

virtual int **findWord** (char* word)

this methods return the index of the given word, if found

this methods return the index of the given word, if found

90.11

virtual int **findPhoneme** (char* phoneme)

this methods return the index of the given phoneme, if found

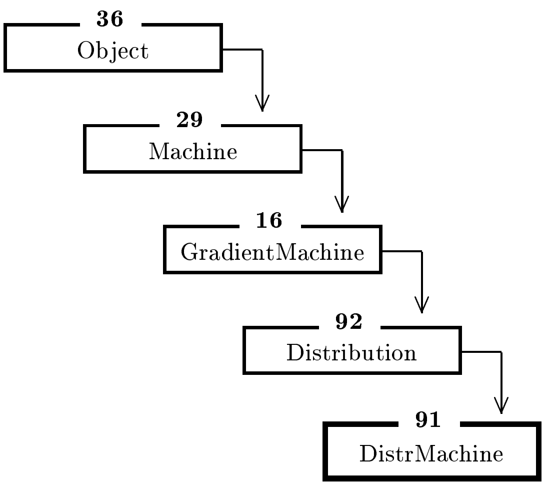
this methods return the index of the given phoneme, if found

91

class **DistrMachine** : public Distribution

This class can be used to implement a conditional distribution P(y—x;theta).

Inheritance



Public Members

91.1	Distribution*	**distribution**	*the output distribution*	224
91.2	GradientMachine*	**machine**	*the machine that produces the parameters of distribution*	224
91.3	List	**input_machine**	*to interface the input*	224
91.4	real*	**der_params_distribution**	*to keep the derivative of the parameters for the machine alpha* .	225
91.5		**DistrMachine** (Distribution* distribution_, GradientMachine* machine_)	*creates a DistrMachine given a machine and and output distribution*	225

This class can be used to implement a conditional distribution P(y—x;theta). It is represented as a machine followed by a distribution. The machine, given some inputs "x", computes the parameters "theta" of the distribution, then given these parameters, the distribution computes its log probability.

Author: Samy Bengio (bengio@idiap.ch)

91.1

Distribution* **distribution**

the output distribution

the output distribution

91.2

GradientMachine* **machine**

the machine that produces the parameters of distribution

the machine that produces the parameters of distribution

91.3

List **input_machine**

to interface the input

to interface the input

91.4

real* **der_params_distribution**

to keep the derivative of the parameters for the machine alpha

to keep the derivative of the parameters for the machine alpha

91.5

DistrMachine (Distribution* distribution_, GradientMachine* machine_)

creates a DistrMachine given a machine and and output distribution

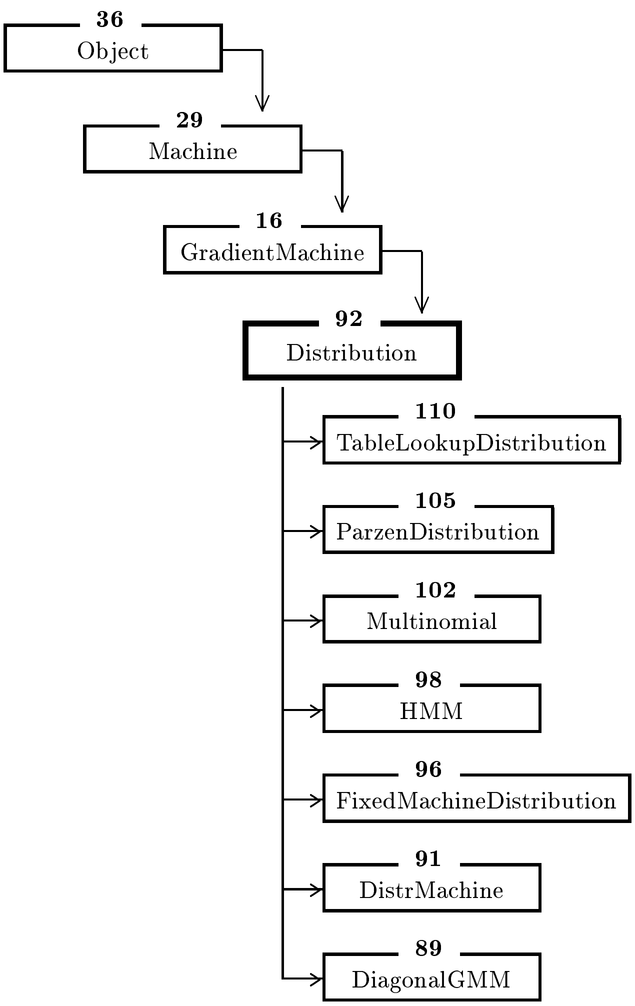
creates a DistrMachine given a machine and and output distribution

92

class **Distribution** : public GradientMachine

This class is designed to handle generative distribution models such as Gaussian Mixture Models and Hidden Markov Models.

Inheritance



Public Members

This class is designed to handle generative distribution models such as Gaussian Mixture Models and Hidden Markov Models. As distribution inherits from GradientMachine, they can be trained by gradient descent or by Expectation Maximization (EM) or even Viterbi.

Note that the output of a distribution is the negative log likelihood.

Author: Samy Bengio (bengio@idiap.ch)

92.1

int **n_observations**

size of the observation vectors

size of the observation vectors

92.2

int **tot_n_frames**

total number of frames

total number of frames

92.3

int **max_n_frames**

the longest sequence in the database (used to dimensionate the variables)

the longest sequence in the database (used to dimensionate the variables)

92.4

real **log_probability**

the log likelihood

the log likelihood

92.5

real* **log_probabilities**

the log likelihood for each frame

the log likelihood for each frame

92.6

Distribution ()

92.7

virtual real **logProbability** (List* inputs)

Returns the log probability of a sequence represented by `inputs`

Returns the log probability of a sequence represented by `inputs`

92.8

virtual real **viterbiLogProbability** (List* inputs)

Returns the viterbi score of a sequence represented by `inputs`

Returns the viterbi score of a sequence represented by `inputs`

92.9

virtual real **frameLogProbability** (real* observations, real* inputs, int t)

Returns the log probability of a frame of a sequence

Returns the log probability of a frame of a sequence

92.10

virtual void **frameExpectation** (real* observations, real* inputs, int t)

Returns the expected value of `observations` *given* `inputs`

Returns the expected value of `observations` given `inputs`

92.11

virtual void **eMIterInitialize** ()

Methods used to initialize the model at the beginning of each EM iteration

Methods used to initialize the model at the beginning of each EM iteration

92.12

virtual void **iterInitialize** ()

Methods used to initialize the model at the beginning of each gradient descent iteration

Methods used to initialize the model at the beginning of each gradient descent iteration

92.13

virtual void **eMSequenceInitialize** (List* inputs)

Methods used to initialize the model at the beginning of each example during EM training

Methods used to initialize the model at the beginning of each example during EM training

92.14

virtual void **sequenceInitialize** (List* inputs)

Methods used to initialize the model at the beginning of each example during gradient descent training

Methods used to initialize the model at the beginning of each example during gradient descent training

92.15

virtual void **eMAccPosteriors** (List* inputs, real log_posterior)

The backward step of EM for a sequence

The backward step of EM for a sequence

92.16

virtual void **frameEMAccPosteriors** (real* observations, real log_posterior, real* inputs, int t)

The backward step of EM for a frame

The backward step of EM for a frame

92.17

virtual void **viterbiAccPosteriors** (List* inputs, real log_posterior)

The backward step of Viterbi learning for a sequence

The backward step of Viterbi learning for a sequence

92.18

virtual void **frameViterbiAccPosteriors** (real* observations, real log_posterior, real* inputs, int t)

The backward step of Viterbi for a frame

The backward step of Viterbi for a frame

92.19

virtual void **eMUpdate** ()

The update after each iteration for EM

The update after each iteration for EM

92.20

virtual void **decode** (List* inputs)

For some distribution like SpeechHMM, decodes the most likely path

For some distribution like SpeechHMM, decodes the most likely path

92.21

virtual void **eMForward** (List* inputs)

Same as forward, but for EM

Same as forward, but for EM

92.22

virtual void **viterbiForward** (List* inputs)

Same as forward, but for Viterbi

Same as forward, but for Viterbi

92.23

virtual void **frameBackward** (real* observations, real* alpha, real* inputs, int t)

Same as backward, but for one frame only

Same as backward, but for one frame only

92.24

virtual void **viterbiBackward** (List* inputs, real* alpha)

Same as backward, but for Viterbi

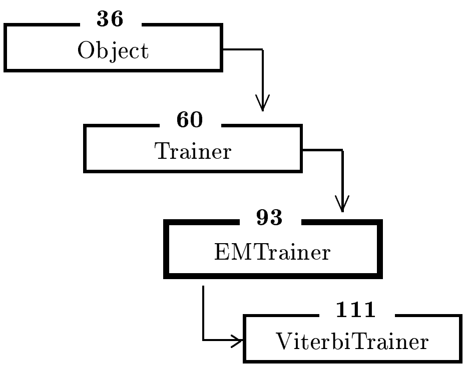
Same as backward, but for Viterbi

93

class **EMTrainer** : public Trainer

This class is used to train any distribution using the EM algorithm.

Inheritance



Public Members

93.1 Distribution* **distribution** *the distribution to train* 237

93.2 SeqDataSet* **sdata** *the training set is a SeqDataSet, since we are working with distributions* 237

93.3 real **end_accuracy** *the stopping criterion regarding the accuracy for EM* 237

93.4 int **max_iter** *the stopping criterion regarding the number of iterations for EM* 237

93.5 **EMTrainer** (Distribution* distribution_, SeqDataSet* data_) 238

93.6 virtual void **decode** (List* measurers) *this method computes the most likely path into the distribution.* 238

This class is used to train any distribution using the EM algorithm.

Author: Samy Bengio (bengio@idiap.ch)

93.1

Distribution* **distribution**

the distribution to train

the distribution to train

93.2

SeqDataSet* **sdata**

the training set is a SeqDataSet, since we are working with distributions

the training set is a SeqDataSet, since we are working with distributions

93.3

real **end_accuracy**

the stopping criterion regarding the accuracy for EM

the stopping criterion regarding the accuracy for EM

93.4

int **max_iter**

the stopping criterion regarding the number of iterations for EM

the stopping criterion regarding the number of iterations for EM

93.5

EMTrainer (Distribution* distribution_, SeqDataSet* data_)

93.6

virtual void **decode** (List* measurers)

this method computes the most likely path into the distribution.

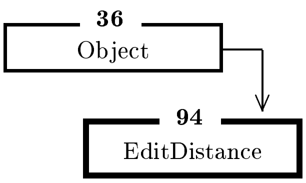
this method computes the most likely path into the distribution. mainly used for sequential distribution such as HMMs.

94

class **EditDistance** : public Object

This class can be used to compute the "edit distance" between two sequences.

Inheritance



Public Members

94.1	real	**accuracy**	*the total edit distance between two sequences* 240
94.2	int	**n_insert**	*the number of insertions (weighted by their cost)* 241
94.3	int	**n_delete**	*the number of deletions (weighted by their cost)* 241
94.4	int	**n_subst**	*the number of substitutions (weighted by their cost)* 241
94.5	int	**n_seq**	*the number of sequences measured (used to normalize* `dist` *by* `n_seq`*)* 241
94.6	int	**insert_cost**	*the cost of one insertion* 242
94.7	int	**delete_cost**	*the cost of one deletion* 242
94.8	int	**subst_cost**	*the cost of one substitution* 242
94.9	int*	**obtained**	*the obtained sequence* 242
94.10	int	**obt_size**	*the size of the obtained sequence* 243
94.11	int*	**desired**	*the desired sequence* 243
94.12	int	**des_size**	*the size of the desired sequence* . 243
94.13		**EditDistance** ()	 243
94.14	virtual	void	

This class can be used to compute the "edit distance" between two sequences. It computes the number of insertions, deletions and substitutions. The overall distance is the sum of these numbers weighted by their cost (which are intergers equal to 1 by default).

Author: Samy Bengio (bengio@idiap.ch)

94.1

real **accuracy**

the total edit distance between two sequences

the total edit distance between two sequences

94.2

int **n_insert**

the number of insertions (weighted by their cost)

the number of insertions (weighted by their cost)

94.3

int **n_delete**

the number of deletions (weighted by their cost)

the number of deletions (weighted by their cost)

94.4

int **n_subst**

the number of substitutions (weighted by their cost)

the number of substitutions (weighted by their cost)

94.5

int **n_seq**

the number of sequences measured (used to normalize `dist` *by* `n_seq`*)*

the number of sequences measured (used to normalize `dist` by `n_seq`)

94.6

int **insert_cost**

the cost of one insertion

the cost of one insertion

94.7

int **delete_cost**

the cost of one deletion

the cost of one deletion

94.8

int **subst_cost**

the cost of one substitution

the cost of one substitution

94.9

int* **obtained**

the obtained sequence

the obtained sequence

94.10

int **obt_size**

the size of the obtained sequence

the size of the obtained sequence

94.11

int* **desired**

the desired sequence

the desired sequence

94.12

int **des_size**

the size of the desired sequence

the size of the desired sequence

94.13

EditDistance ()

94.14

virtual void **setCosts** (int i_cost, int d_cost, int s_cost)

sets the different costs

sets the different costs

94.15

virtual void **distance** (int* obtained, int obt_size, int* desired, int des_size)

computes the edit distance between `obtained` *and* `desired`

computes the edit distance between `obtained` and `desired`

94.16

virtual void **add** (EditDistance* d)

accumulates the distances of the current object and the given object

accumulates the distances of the current object and the given object

94.17

virtual void **print** (FILE* f)

prints the edit distance and optionally the sequences

prints the edit distance and optionally the sequences

94.18

virtual void **printRatio** (FILE* f)

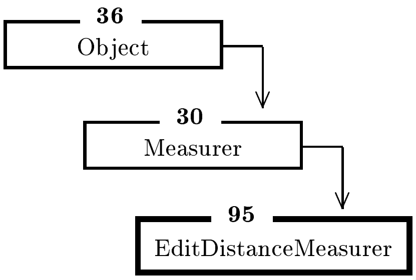
prints the edit distance ratio (divided by the number of sequnces) and optionally the obtained and desired sequences

prints the edit distance ratio (divided by the number of sequnces) and optionally the obtained and desired sequences

95 class **EditDistanceMeasurer** : public Measurer

This class can be used to measure and print an `EditDistance` *object.*

Inheritance



Public Members

95.1	EditDistance*	**edit**	*the* `EditDistance` *to measure*	247
95.2	EditDistance*	**sum**	*an accumulator of* `EditDistance` *objects, to compute statistics*	247
95.3	bool	**print_ex**	*do we print the edit distance for each example?*	247
95.4		**EditDistanceMeasurer**	(EditDistance* edit_, DataSet* data_, FILE* file_, bool print_ex_ = false)	247
95.5		**EditDistanceMeasurer**	(EditDistance* edit_, DataSet* data_, const char* filename, bool print_ex_ = false)	248

This class can be used to measure and print an `EditDistance` object. It is used for instance in speech recognition experiments.

Author: Samy Bengio (bengio@idiap.ch)

95.1

EditDistance* **edit**

the `EditDistance` *to measure*

the `EditDistance` to measure

95.2

EditDistance* **sum**

an accumulator of `EditDistance` *objects, to compute statistics*

an accumulator of `EditDistance` objects, to compute statistics

95.3

bool **print_ex**

do we print the edit distance for each example?

do we print the edit distance for each example?

95.4

EditDistanceMeasurer (EditDistance* edit_, DataSet* data_, FILE* file_, bool print_ex_ = false)

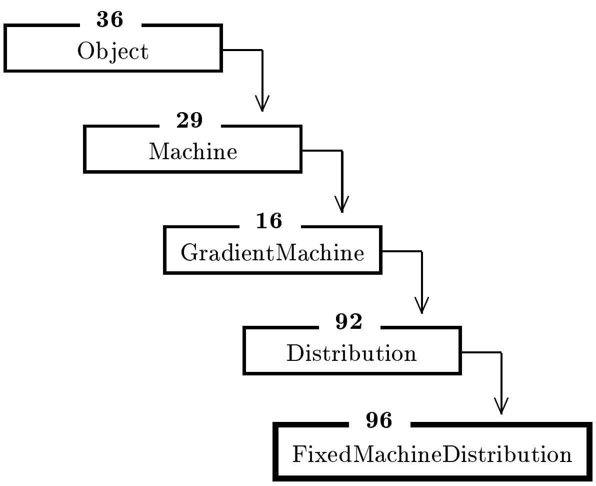
95.5

EditDistanceMeasurer (EditDistance* edit_, DataSet* data_, const char* filename, bool print_ex_ = false)

96 class **FixedMachineDistribution** : public Distribution

This class uses one of the outputs of a given pre-trained machine as an estimate of a probability (used in the method `frameLogProbability`.

Inheritance



Public Members

96.1	Machine*	**machine**	*The (already trained) Machine that computes the probability.* ..	250
96.2	int	**output_number**	*which output should we take*	250
96.3	real	**prior**	*prior probability used to normalize the machine output*	251
96.4	List	**machine_inputs**	*temporary List used to compute machine forward*	251
96.5		**FixedMachineDistribution** (Machine* machine_, int output_number_ = 0, real prior_ = 1.)		

This class uses one of the outputs of a given pre-trained machine as an estimate of a probability (used in the method `frameLogProbability`. It can normalize it using an optional prior value. This class can therefore be used in conjunction with HMMs to implement the HMM/ANN hybrid model...

Author: Samy Bengio (bengio@idiap.ch)

96.1

Machine* **machine**

The (already trained) Machine that computes the probability.

The (already trained) Machine that computes the probability. We suppose that the outputs of the machines are positive and sum to 1. One possibility is to use softmax outputs

96.2

int **output_number**

which output should we take

which output should we take

96.3

real **prior**

prior probability used to normalize the machine output

prior probability used to normalize the machine output

96.4

List **machine_inputs**

temporary List used to compute machine forward

temporary List used to compute machine forward

96.5

FixedMachineDistribution (Machine* machine_, int output_number_ = 0, real prior_ = 1.)

this distribution is created with an already trained machine, the index of the output corresponding to this distribution, and an eventual prior

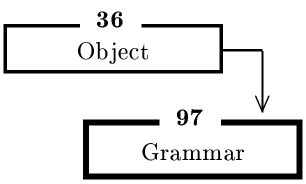
this distribution is created with an already trained machine, the index of the output corresponding to this distribution, and an eventual prior

97

class **Grammar** : public Object

This class contains the grammar of accepted sentences for a speech recognition experiment such as the one using SpeechHMM A grammar is a transition table where each node is a word.

Inheritance



Public Members

97.1	int	**n_words**	*the number of words in the grammar (different than the number of words in the dictionary, because the same word can appear twice in the grammar)*	253
97.2	int*	**words**	*the index of the words (in the dictionary object)*	253
97.3	int*	**start**	*this vector is used in SpeechHMM to keep the state index in the decoding state matrix, corresponding to the given word*	253
97.4	bool**	**transitions**	*the transition matrix.*	253
97.5		**Grammar** (int n_words_)		254

This class contains the grammar of accepted sentences for a speech recognition experiment such as the one using SpeechHMM A grammar is a transition table where each node is a word. The user is responsible to set the transition table as he wishes. (by default, there are no transition!)

Author: Samy Bengio (bengio@idiap.ch)

97.1

int **n_words**

the number of words in the grammar (different than the number of words in the dictionary, because the same word can appear twice in the grammar)

the number of words in the grammar (different than the number of words in the dictionary, because the same word can appear twice in the grammar)

97.2

int* **words**

the index of the words (in the dictionary object)

the index of the words (in the dictionary object)

97.3

int* **start**

this vector is used in SpeechHMM to keep the state index in the decoding state matrix, corresponding to the given word

this vector is used in SpeechHMM to keep the state index in the decoding state matrix, corresponding to the given word

97.4

bool** **transitions**

the transition matrix.

the transition matrix. each true transition (i,j) means the word whose index is i can be followed by the word whose index is j

97.5

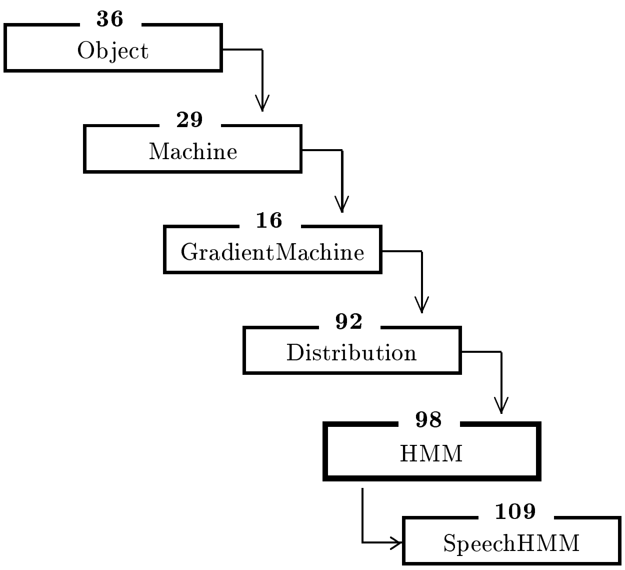
Grammar (int n_words_)

98

class **HMM** : public Distribution

This class implements a Hidden Markov Model distribution.

Inheritance



Public Members

98.1	SeqDataSet*	**data**	*keep the training data to compute the size of the longest sequence*	257
98.2	int	**n_states**	*The number of states of the HMM.*	258
98.3	real	**prior_transitions**	*a prior on the transition probabilities*	258
98.4	Distribution**	**states**	*keep the emission distributions*	258
98.5	Distribution**			

This class implements a Hidden Markov Model distribution. In fact, it also implements the Input Output version (IOHMM). It can be trained either by EM, Viterbi, or Gradient Descent.

Note that this kind of HMM always contain one initial state and one final state. Both are non-emitting.

Author: Samy Bengio (bengio@idiap.ch)

98.1

SeqDataSet* **data**

keep the training data to compute the size of the longest sequence

keep the training data to compute the size of the longest sequence

98.2

int **n_states**

The number of states of the HMM.

The number of states of the HMM. the first model is the initial state, the last model is the final (absorbing) state, (neither of them are emitting). hence, n_states > 2

98.3

real **prior_transitions**

a prior on the transition probabilities

a prior on the transition probabilities

98.4

Distribution** **states**

keep the emission distributions

keep the emission distributions

98.5

Distribution** **unique_states**

in the case where the emission distribution share their parameters, `unique_states` *contains the prototypes which will be used for updates (otherwise, it is a pointer to* `states`*).*

in the case where the emission distribution share their parameters, `unique_states` contains the prototypes which will be used for updates (otherwise, it is a pointer to `states`). Note that if given, and as for `states`, `unique_states` contains an empty state for positions 0 (initial state) and `n_states` - 1 (final state)

98.6

int **n_unique_states**

if `unique_states` *is given, n_unique_states is its size*

if `unique_states` is given, n_unique_states is its size

98.7

real** **transitions**

the initial transitions between states are kept as a matrix

the initial transitions between states are kept as a matrix

98.8

real** **log_transitions**

in fact, we keep the transitions in log

in fact, we keep the transitions in log

98.9

real** **dlog_transitions**

the derivative of the log transitions for gradient descent

the derivative of the log transitions for gradient descent

98.10

real** **transitions_acc**

the accumulators of the transitions for EM

the accumulators of the transitions for EM

98.11

real** **log_alpha**

accumulator used in the forward phase to compute log likelihood

accumulator used in the forward phase to compute log likelihood

98.12

real** **log_beta**

accumulator used in the backward phase to compute log likelihood

accumulator used in the backward phase to compute log likelihood

98.13

int** **arg_viterbi**

for each state, for each time step, keep the best predecessor

for each state, for each time step, keep the best predecessor

98.14

int* **viterbi_sequence**

for each time step, keep the best state

for each time step, keep the best state

98.15

real** **log_probabilities_s**

keep for each time step and each model its emission log probability

keep for each time step and each model its emission log probability

98.16

virtual void **printTransitions** (bool real_values=false, bool transitions_only=false)

this method can be used for debugging purpose to see the transitions

this method can be used for debugging purpose to see the transitions

98.17

virtual void **logAlpha** (SeqExample* ex)

computes the log_alpha during forward phase of EM

computes the log_alpha during forward phase of EM

98.18

virtual void **logBeta** (SeqExample* ex)

computes the log_beta during backward phase of EM

computes the log_beta during backward phase of EM

98.19

virtual void **logViterbi** (SeqExample* ex)

computes the log_viterbi during forward phase of Viterbi

computes the log_viterbi during forward phase of Viterbi

98.20

virtual void **decode** (List* input)

this method returns the state sequence associated to the input

this method returns the state sequence associated to the input

98.21

virtual void **logProbabilities** (List* inputs)

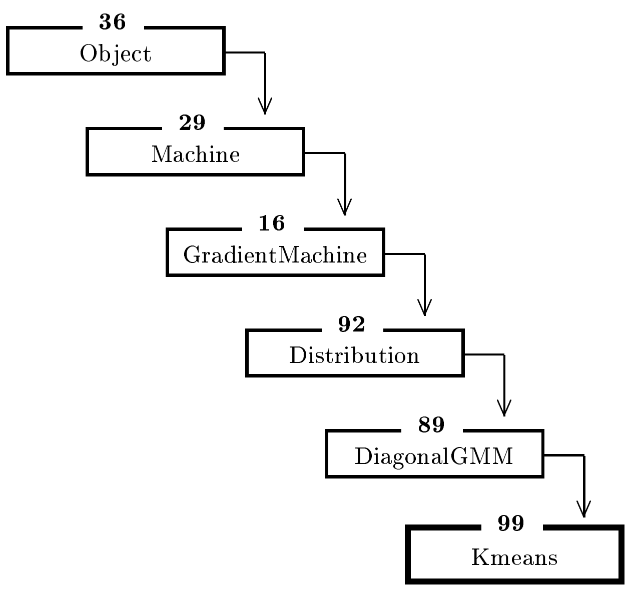
computes for each state and each time step of the sequence `inputs` *its associated emission probability*

computes for each state and each time step of the sequence `inputs` its associated emission probability

99 class **Kmeans** : public DiagonalGMM

This class can be used to do a "kmeans" on a given set of data.

Inheritance



Public Members

99.1 int* **min_cluster** *for each example, keep the index of the neirest cluster* 264

99.2 SeqDataSet* **data** *we need the dataset in order to initialize correctly the centers of the clusters* 264

99.3 **Kmeans** (int n_observations_, int n_gaussians_, real* stdev_threshold_, real prior_weights_, SeqDataSet* data_) 265

99.4 virtual real

This class can be used to do a "kmeans" on a given set of data. It has been implemented in the framework of a Distribution that can be trained with EM. This means that the kmeans distance is in fact returned by the method logProbability.

Note that as Kmeans is a subclass of DiagonalGMM, they share the same parameter structure. Hence, a DiagonalGMM can be easily initialized by a Kmeans.

Author: Samy Bengio (bengio@idiap.ch)

99.1

int* **min_cluster**

for each example, keep the index of the neirest cluster

for each example, keep the index of the neirest cluster

99.2

SeqDataSet* **data**

we need the dataset in order to initialize correctly the centers of the clusters

we need the dataset in order to initialize correctly the centers of the clusters

99.3

Kmeans (int n_observations_, int n_gaussians_, real* stdev_threshold_, real prior_weights_, SeqDataSet* data_)

99.4

virtual real **frameLogProbability** (real* observations, real* inputs, int t)

note that this method returns in fact the euclidean distance between the observation and the neirest cluster

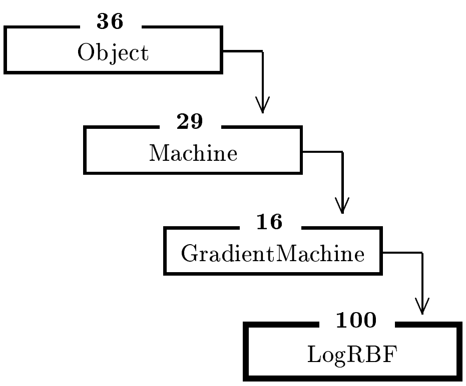
note that this method returns in fact the euclidean distance between the observation and the neirest cluster

100

class **LogRBF** : public GradientMachine

LogRBF layer for `GradientMachine`*.*

Inheritance



Public Members

100.1 EMTrainer*
initial_kmeans_trainer
optional initialization using a Kmeans 267

100.2 **LogRBF** (int n_inputs_, int n_outputs_, EMTrainer* kmeans_trainer=NULL)
.... 267

LogRBF layer for `GradientMachine`. Formally speaking, $ouputs[i] = -0.5 \sum_j gamma_i j^2 * (inputs[j] - mu_i j)^2$.
$mu_i j$ and $gamma_i j$ are in `params`, with the following structure:
$mu_0 0 ... mu_0 n, gamma_0 0 .. gamma_0 n, ...,$

For a better initialization, one can provide a `EMTrainer` using a `Kmeans` distribution that will be used to initialize the means and gamma.

Author: Samy Bengio (bengio@idiap.ch)

100.1

EMTrainer* **initial_kmeans_trainer**

optional initialization using a Kmeans

optional initialization using a Kmeans

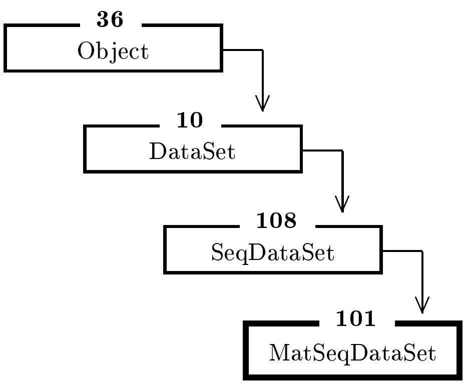
100.2

LogRBF (int n_inputs_, int n_outputs_, EMTrainer* kmeans_trainer=NULL)

101 class **MatSeqDataSet** : public SeqDataSet

This class enable to read sequences (hence inherits from `SeqDataSet`*) from the classical Torch data format.*

Inheritance



Public Members

101.1 bool **examples_from_stddataset**
this boolean is used to know if the data has already been read by a `StdDataSet` *and hence do not need to be freed at the end* 269

101.2 **MatSeqDataSet** (StdDataSet* data, int n_inputs_, int n_observations_)
puts a data set in the Sequence framework 270

101.3 **the following method is inherited from #SeqDataSet#**
.... 270

101.4 void **readMat** (char* file, bool bin=false, int max_load=-1)

This class enable to read sequences (hence inherits from `SeqDataSet`) from the classical Torch data format. Depending on the options, it can read one sequence per line (thus having sequences of size 1) or one sequence per file (then having sequences of size the size of the files). One can specify the number of inputs, observations and targets in the given file.

Author: Bison Ravi (francois.belisle@idiap.ch) Samy Bengio (samy.bengio@idiap.ch)

101.1

bool **examples_from_stddataset**

this boolean is used to know if the data has already been read by a `StdDataSet` *and hence do not need to be freed at the end*

this boolean is used to know if the data has already been read by a `StdDataSet` and hence do not need to be freed at the end

101.2

MatSeqDataSet (StdDataSet* data, int n_inputs_, int n_observations_)

puts a data set in the Sequence framework puts a data set in the Sequence framework

101.3

the following method is inherited from #SeqDataSet#

the following method is inherited from `SeqDataSet`

101.4

void **readMat** (char* file, bool bin=false, int max_load=-1)

reads the data into the `all_inputs` *structure which will be transfered in the* `examples` *structure in* `createData`

reads the data into the `all_inputs` structure which will be transfered in the `examples` structure in `createData`

101.5

int **lines** (char*, bool bin)

finds the number of lines in all of the given files

finds the number of lines in all of the given files

101.6

int **columns** (char*, bool bin)

finds the number of columns in a file

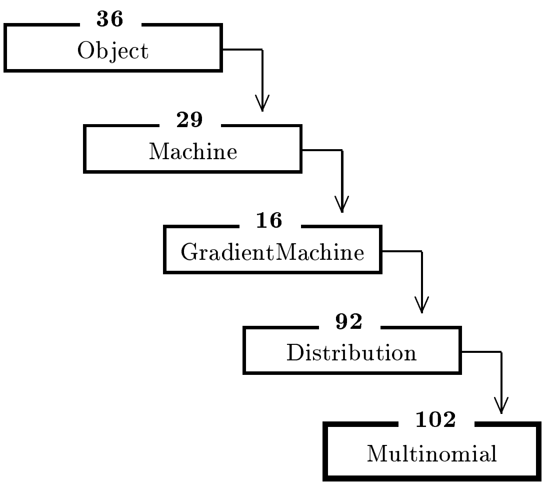
finds the number of columns in a file

102

class **Multinomial** : public Distribution

This class can be used to model Multinomial Distributions.

Inheritance



Public Members

102.1	int	**n_values**	*the number of different values that can take this discrete distribution*	272
102.2	real	**prior_weights**	*the prior weight given to each value.*	272
102.3	List*	**initial_params**	*optional initialization parameters*	272
102.4	char*	**initial_file**	*optional initialization file*	272
102.5	real*	**log_weights**	*the pointers to the parameters*	273
102.6	real*	**dlog_weights**	*the pointers to the d_parameters*	273
102.7	real*	**weights_acc**	*accumulators for EM*	273

This class can be used to model Multinomial Distributions. They can be trained using either EM (with EMTrainer) or gradient descent (with GMTrainer).

Author: Samy Bengio (bengio@idiap.ch)

102.1

int **n_values**

the number of different values that can take this discrete distribution

the number of different values that can take this discrete distribution

102.2

real **prior_weights**

the prior weight given to each value.

the prior weight given to each value. kind of smoother

102.3

List* **initial_params**

optional initialization parameters

optional initialization parameters

102.4

char* **initial_file**

optional initialization file

optional initialization file

102.5

real* **log_weights**

the pointers to the parameters

the pointers to the parameters

102.6

real* **dlog_weights**

the pointers to the d_parameters

the pointers to the d_parameters

102.7

real* **weights_acc**

accumulators for EM

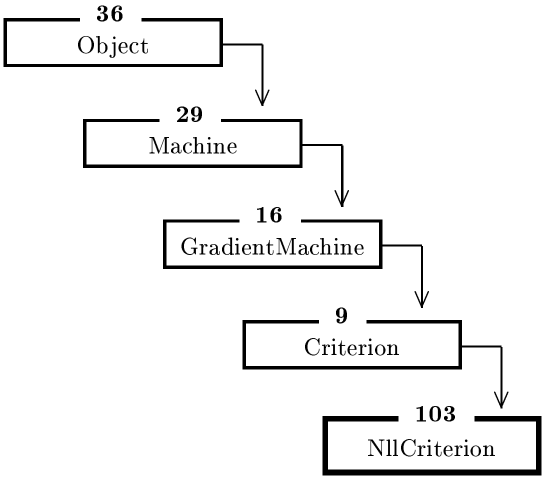
accumulators for EM

103

class **NllCriterion** : public Criterion

This criterion can be used to train `Distribution` *object using the* `GMTrainer` *trainer.*

Inheritance



Public Members

This criterion can be used to train `Distribution` object using the `GMTrainer` trainer. It then maximizes the log likelihood of the data.

The `forward` method always return its input, which is the negative log likelihood, while the `backward` method sets the gradient to -1.

Author: Samy Bengio (bengio@idiap.ch)

103.1

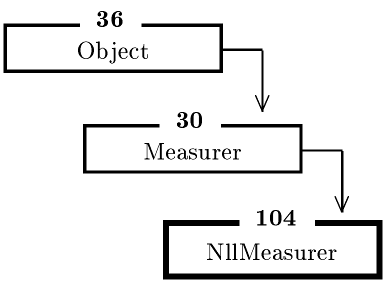
NllCriterion ()

104

class **NllMeasurer** : public Measurer

This class measures the negative log likelihood.

Inheritance



Public Members

104.1	real*	**erreur**	*contains the average negative log likelihood*	276
104.2	real*	**erreur_c**	*contains the total negative log likelihood*	276
104.3	List*	**inputs**	*contains a pointer to the negative log likelihood to measure*	276
104.4	int	**tot_n_frames**	*local counter of the total number of frames*	276
104.5	SeqDataSet*	**sdata**	*local pointer to the dataset, used to count the number of frames*	277
104.6		**NllMeasurer** (List* inputs_, SeqDataSet* data_, FILE* file_)		277
104.7		**NllMeasurer** (List* inputs_, SeqDataSet* data_, const char* filename)		277

This class measures the negative log likelihood. In fact, it supposes that the input given is already the negative log likelihood. It is then normalized by the total number of frames.

Author: Samy Bengio (bengio@idiap.ch)

104.1

real* **erreur**

contains the average negative log likelihood

contains the average negative log likelihood

104.2

real* **erreur_c**

contains the total negative log likelihood

contains the total negative log likelihood

104.3

List* **inputs**

contains a pointer to the negative log likelihood to measure

contains a pointer to the negative log likelihood to measure

104.4

int **tot_n_frames**

local counter of the total number of frames

local counter of the total number of frames

104.5

SeqDataSet* **sdata**

local pointer to the dataset, used to count the number of frames

local pointer to the dataset, used to count the number of frames

104.6

NllMeasurer (List* inputs_, SeqDataSet* data_, FILE* file_)

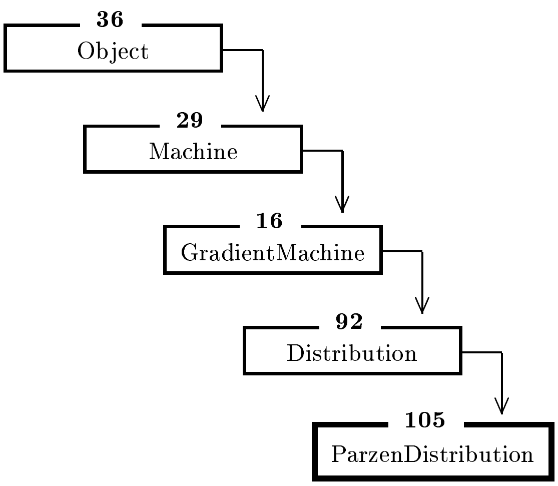
104.7

NllMeasurer (List* inputs_, SeqDataSet* data_, const char* filename)

105 class **ParzenDistribution** : public Distribution

This class can be used to model a Parzen density estimator with a Gaussian kernel:

Inheritance



Public Members

This class can be used to model a Parzen density estimator with a Gaussian kernel:

$p(x) = \frac{1}{N} \sum_i \frac{1}{(2\Pi var)^{d/2}} \exp(-\frac{||x-x_i||^2}{2var})$

where the sum is done on the whole training set.

Author: Samy Bengio (bengio@idiap.ch)

105.1

real **var**

the variance used

the variance used

105.2

SeqDataSet* **data**

the dataset

the dataset

105.3

int* **real_examples**

the indices of the training examples

the indices of the training examples

105.4

real **sum_log_var_plus_n_obs_log_2_pi**

*in order to faster the computation, we can do some "pre-computation" pre-computed sum_log_var + n_obs * log_2_pi*

in order to faster the computation, we can do some "pre-computation" pre-computed sum_log_var + n_obs * log_2_pi

105.5

real **minus_half_over_var**

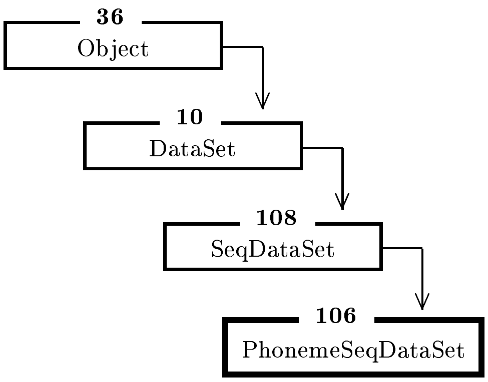
pre-computed -05 / var

pre-computed -05 / var

106 class **PhonemeSeqDataSet** : public SeqDataSet

This class is designed to create a dataset based on another dataset using parts only related to a given phoneme

Inheritance



Public Members

This class is designed to create a dataset based on another dataset using parts only related to a given phoneme

Author: Samy Bengio (bengio@idiap.ch)

106.1 create a new Phoneme dataset from generic SeqDataSet

create a new Phoneme dataset from generic SeqDataSet

107 struct **SeqExample**

This structure keeps a sequence example

Members

107.1	int	**n_frames**	*the number of frames of the sequence*	283
107.2	int	**n_seqtargets**	*if there are targets, the number of targets (the length of the matrix)*	283
107.3	real**	**inputs**	*if there are inputs, the matrix containing the inputs*	283
107.4	real**	**observations**	*if there are observations, the matrix containing the observations*	283
107.5	real**	**seqtargets**	*if there are targets, the matrix containing the targets*	284
107.6	int*	**alignment**	*alignment between observations and seqtargets, when available.*	284
107.7	int*	**alignment_phoneme**	*alignment phoneme index*	284
107.8	int	**n_alignments**	*the size of the alignment vector (needs not to coincide with seqtargets)*	284
107.9	char*	**name**	*some examples have names.*	285
107.10	int*	**selected_frames**	*if only some frames of the example is selected*	285
107.11	int	**n_real_frames**	*how many real frames are there in the example*	285
107.12	int	**current_frame**	*which frame are we handling now*	285

This structure keeps a sequence example

107.1

int **n_frames**

the number of frames of the sequence

the number of frames of the sequence

107.2

int **n_seqtargets**

if there are targets, the number of targets (the length of the matrix)

if there are targets, the number of targets (the length of the matrix)

107.3

real** **inputs**

if there are inputs, the matrix containing the inputs

if there are inputs, the matrix containing the inputs

107.4

real** **observations**

if there are observations, the matrix containing the observations

if there are observations, the matrix containing the observations

107.5

real** **seqtargets**

if there are targets, the matrix containing the targets

if there are targets, the matrix containing the targets

107.6

int* **alignment**

alignment between observations and seqtargets, when available.

alignment between observations and seqtargets, when available. alignment[i] gives the index of the inputs/observations corresponding to phoneme alignment_phoneme[i].

107.7

int* **alignment_phoneme**

alignment phoneme index

alignment phoneme index

107.8

int **n_alignments**

the size of the alignment vector (needs not to coincide with seqtargets)

the size of the alignment vector (needs not to coincide with seqtargets)

107.9

char* **name**

some examples have names.

some examples have names. this is used to make a relation between alignment datafiles and observation datafiles

107.10

int* **selected_frames**

if only some frames of the example is selected

if only some frames of the example is selected

107.11

int **n_real_frames**

how many real frames are there in the example

how many real frames are there in the example

107.12

int **current_frame**

which frame are we handling now

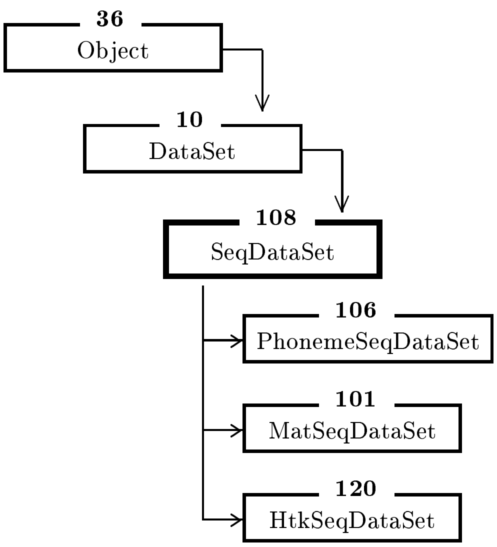
which frame are we handling now

108

class **SeqDataSet** : public DataSet

This class defines the standard framework of Sequence Data processing

Inheritance



Public Members

108.1	SeqExample*	**examples**	*the vector of examples of the dataset*	289
108.2	char**	**file_names**	*names of data files*	289
108.3	int	**n_file_names**	*the number of data files to read*	289
108.4	bool	**norm_inputs**	*do we normalize the inputs?*	290
108.5	bool	**norm_observations**	*do we normalize the observations?*	290
108.6	real*	**mean_in**	*if we normalize the inputs, this vector contains the mean of the inputs*	290

This class defines the standard framework of Sequence Data processing

DataSet with a vector of reals for inputs, vector of reals for observations and vector of reals for targets

Options:

"normalize inputs" bool normalize the inputs by mean/stdv [false] "normalize observations" bool normalize the observations by mean/stdv [false] "normalize targets" bool normalize the targets by mean/stdv [false]

p

Author: Bison Ravi (francois.belisle@idiap.ch) Johnny Mariethoz (Johnny.Mariethoz@idiap.ch) Samy Bengio (bengio@idiap.ch)

108.1

SeqExample* **examples**

the vector of examples of the dataset

the vector of examples of the dataset

108.2

char** **file_names**

names of data files

names of data files

108.3

int **n_file_names**

the number of data files to read

the number of data files to read

108.4

bool **norm_inputs**

do we normalize the inputs?

do we normalize the inputs?

108.5

bool **norm_observations**

do we normalize the observations?

do we normalize the observations?

108.6

real* **mean_in**

if we normalize the inputs, this vector contains the mean of the inputs

if we normalize the inputs, this vector contains the mean of the inputs

108.7

real* **mean_ob**

if we normalize the inputs, this vector contains the mean of the observations

if we normalize the inputs, this vector contains the mean of the observations

108.8

real* **stdv_in**

if we normalize the inputs, this vector contains the stdev of the inputs

if we normalize the inputs, this vector contains the stdev of the inputs

108.9

real* **stdv_ob**

if we normalize the inputs, this vector contains the stdev of the observations

if we normalize the inputs, this vector contains the stdev of the observations

108.10

int **current_frame**

the current frame we are working on

the current frame we are working on

108.11

int **n_frames**

the number of frames of the current example

the number of frames of the current example

108.12

int **n_observations**

the number of observations of the examples

the number of observations of the examples

108.13

int **tot_n_frames**

the total number of frames in the dataset

the total number of frames in the dataset

108.14

char* **kind**

some dataset keeps an information of the kind of the data.

some dataset keeps an information of the kind of the data. this is used in Htk data files for instance

108.15

int **n_seqtargets**

if there are targets, the length of the seqtargets matrix.

if there are targets, the length of the seqtargets matrix. Note that its width is equal to n_targets.

108.16

real** **seqtargets**

the targets of the current example

the targets of the current example

108.17

virtual void **freeMemory** ()

each Subclass of `SeqDataSet` *is forced to allocate/free its own memory.*

each Subclass of `SeqDataSet` is forced to allocate/free its own memory. This is forced upon them for historical reasons eg IOTorch allocates/frees its own memory which is used in `StaticMatSeqDataSet`

108.18

int **removeUnlikelyFrames** (Distribution* likely_distr, Distribution* unlikely_distr, int obs_offset = 0, int range=5)

Remove the frames considering as unlinkely and return the number of removed frames

Remove the frames considering as unlinkely and return the number of removed frames

108.19

void **removeUnusedData** (bool* mask)

Remove the column of data unused

Remove the column of data unused

108.20

void **normalizeUsingDataSet** (SeqDataSet*)

Normalize the `DataSet` *using the normalization values of another* `DataSet`*.*

Normalize the `DataSet` using the normalization values of another `DataSet`. If the other `DataSet` isn't normalized, nothing happens...

108.21

void **normalize** ()

normalize the database

normalize the database

108.22

void **setFrame** (int)

set the current frame of the current example

set the current frame of the current example

108.23

void **totNFrames** ()

compute the total number of frames in the database.

compute the total number of frames in the database. keep this value in tot_n_frames

108.24

void **toOneFramePerExample** ()

transform the database such that there is one frame per example.

transform the database such that there is one frame per example. by default, each file corresponds to one sequence.

108.25

virtual void **readTargets** (char* file)

read seqtargets from a file

read seqtargets from a file

108.26

virtual void **readAlignments** (char* file, bool needs_all_examples = false)

read alignments from a file

read alignments from a file

108.27

void **setSelectedFrames** (int* selected_frames, int n_selected_frames)

for the current example, consider only the `selected_frames`

for the current example, consider only the `selected_frames`

108.28

void **unsetSelectedFrames** ()

for the current example, consider all the frames

for the current example, consider all the frames

108.29

void **display** ()

print the database, for debugging purposes only

print the database, for debugging purposes only

108.30

virtual void **loadFILE** (FILE* file)

Load the normalization parameters

Load the normalization parameters

108.31

virtual void **saveFILE** (FILE* file)

Save the normalization parameters

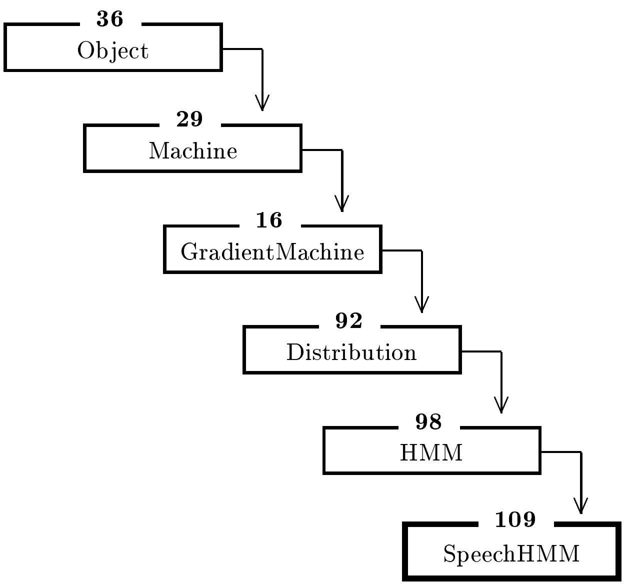
Save the normalization parameters

109

class **SpeechHMM** : public HMM

This class implements a special case of Hidden Markov Models that can be used to do connected word speech recognition for small vocabulary, using embedded training.

Inheritance



Public Members

This class implements a special case of Hidden Markov Models that can be used to do connected word speech recognition for small vocabulary, using embedded training.

It contains a set of phoneme models (represented by HMMs), a dictionary of words (which are sequences of phonemes) and a grammar (which states the legal sentences of the langage).

The decoding is done by creating the whole transition matrix and hence is not adapted to large vocabulary problems.

Author: Samy Bengio (bengio@idiap.ch)

109.1

int **n_models**

the number of basic phoneme models

the number of basic phoneme models

109.2

HMM** **models**

the basic phoneme models

the basic phoneme models

109.3

char** **model_names**

for each model, a unique name (used for translation)

for each model, a unique name (used for translation)

109.4

EMTrainer* **model_trainer**

if an initial alignment is given and an emtrainer for each model then it is used to train the models after kmeans during reset

if an initial alignment is given and an emtrainer for each model then it is used to train the models after kmeans during reset

109.5

Dictionary* **dict**

the acceptable dictionary, using the indices instead of the names

the acceptable dictionary, using the indices instead of the names

109.6

Grammar* **grammar**

the acceptable grammar

the acceptable grammar

109.7

real **word_entrance_penalty**

word entrance penalty: during viterbi, penalizes large sentences

word entrance penalty: during viterbi, penalizes large sentences

109.8

bool** **word_transitions**

true if the given transition is a transition between words

true if the given transition is a transition between words

109.9

int **max_n_states**

the maximum number of states in the graph (used for allocation)

the maximum number of states in the graph (used for allocation)

109.10

int* **states_to_model_states**

the relation between model states and SpeechHMM states

the relation between model states and SpeechHMM states

109.11

int* **states_to_model**

the relation between models and SpeechHMM states

the relation between models and SpeechHMM states

109.12

int* **states_to_word**

the relation between words and SpeechHMM states

the relation between words and SpeechHMM states

109.13

int* **word_sequence**

the word sequence corresponding to the state sequence

the word sequence corresponding to the state sequence

109.14

int* **target_word_sequence**

the target word sequence

the target word sequence

109.15

int **target_word_sequence_size**

the length of the target word sequence

the length of the target word sequence

109.16

int **target_word_sequence_max_size**

the length of the longest target word sequence

the length of the longest target word sequence

109.17

EditDistance* **edit_distance**

this object is used to compute the decoding error

this object is used to compute the decoding error

109.18

SpeechHMM (int n_models_, HMM** models_, char** model_names_, Dictionary* dict_, Grammar* grammar_, real word_entrance_penalty_ = LOG_ONE, EMTrainer* model_trainer_ = NULL)

In order to create a SpeechHMM, we need to give a vector of `n_models_` `HMM`*s as well as their corresponding name, a dictionary and a grammar, an optional word_entrance_penalty and an optional trainer that can be used to initialize each model independently*

In order to create a SpeechHMM, we need to give a vector of `n_models_` `HMM`s as well as their corresponding name, a dictionary and a grammar, an optional word_entrance_penalty and an optional trainer that can be used to initialize each model independently

109.19

virtual void **decode** (List* input)

this method returns the sentence associated to the input

this method returns the sentence associated to the input

109.20

virtual void **prepareTrainModel** (List* input)

this method prepare the transition graph associated with a given training sentence

this method prepare the transition graph associated with a given training sentence

109.21

virtual void **prepareTestModel** (List* input)

this method prepare the transition graph associated with a given test sentence

this method prepare the transition graph associated with a given test sentence

109.22

virtual int **addWordToModel** (int word, int current_state)

this method is used by `prepareTrainModel` *and* `prepareTestModel` *to prepare the model.*

this method is used by `prepareTrainModel` and `prepareTestModel` to prepare the model. It adds a given word to the current graph.

109.23

virtual void **addConnectionsBetweenWordsToModel** (int word, int next_word, int current_state, int next_current_state, real log_n_next)

this method is used by `prepareTrainModel` *and* `prepareTestModel` *to prepare the model.*

this method is used by `prepareTrainModel` and `prepareTestModel` to prepare the model. It adds the connections between words.

109.24

virtual void **realloc** (int n_frames, int n_states_)

this methods reallocate the structure to accomodate a new sequence

this methods reallocate the structure to accomodate a new sequence

109.25

virtual int **nStatesInGrammar** ()

this methods returns the number of states in the grammar

this methods returns the number of states in the grammar

109.26

virtual int **nStatesInWord** (int word)

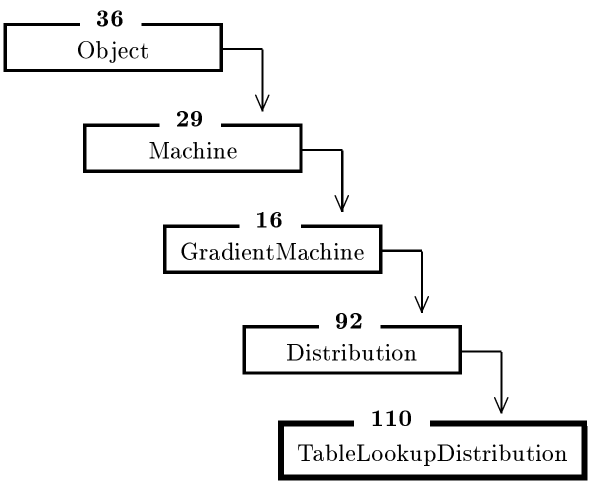
this methods returns the number of states in a given word

this methods returns the number of states in a given word

110 class **TableLookupDistribution** : public Distribution

This class outputs one of the observations as the logProbability.

Inheritance



Public Members

110.1	int	**column**	*The column in the observation vector that corresponds to the logProbability*	310
110.2	bool	**apply_log**	*do we apply a log transformation*	310
110.3	real	**prior**	*do we normalize by a given prior*	310
110.4		**TableLookupDistribution** (int column_ = 0, bool apply_log_ = true, real prior_ = 1.)	*The column number corresponds to the logProbability which can be normalized by an eventual prior*	311

This class outputs one of the observations as the logProbability. It can eventually apply a log transformation and/or normalize by a given prior. It can therefore be used in conjunction with HMMs to implement the HMM/ANN hybrid model...

Author: Samy Bengio (bengio@idiap.ch)

110.1

int **column**

The column in the observation vector that corresponds to the logProbability

The column in the observation vector that corresponds to the logProbability

110.2

bool **apply_log**

do we apply a log transformation

do we apply a log transformation

110.3

real **prior**

do we normalize by a given prior

do we normalize by a given prior

110.4

TableLookupDistribution (int column_ = 0, bool apply_log_ = true, real prior_ = 1.)

The column number corresponds to the logProbability which can be normalized by an eventual prior

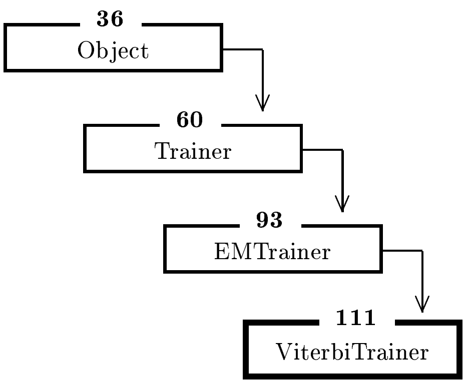
The column number corresponds to the logProbability which can be normalized by an eventual prior

111

class **ViterbiTrainer** : public EMTrainer

This class is used to train any distribution using the Viterbi algorithm.

Inheritance



Public Members

111.1 **ViterbiTrainer** (Distribution* distribution_, SeqDataSet* data_)
creates a Viterbi trainer given a distribution and a dataset 313

This class is used to train any distribution using the Viterbi algorithm. This means that it is most useful for distributions over sequences, such as HMMs.

Author: Samy Bengio (bengio@idiap.ch)

111.1

ViterbiTrainer (Distribution* distribution_, SeqDataSet* data_)

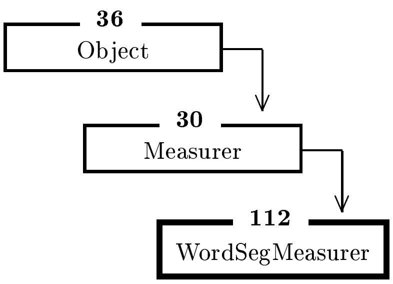
creates a Viterbi trainer given a distribution and a dataset creates a Viterbi trainer given a distribution and a dataset

112

class **WordSegMeasurer** : public Measurer

This class can be used to save the word segmentation of a `SpeechHMM` *in a file.*

Inheritance



Public Members

112.1	SpeechHMM*	**shmm**	*the pointer to the SpeechHMM object*	315
112.2	bool	**print_targets**	*a boolean stating whether we also print the corresponding targets*	315
112.3		**WordSegMeasurer**	(SpeechHMM* shmm_, DataSet* data_, FILE* file_, bool print_targets=true)	315
112.4		**WordSegMeasurer**	(SpeechHMM* shmm_, DataSet* data_, const char* filename, bool print_targets=true)	315

This class can be used to save the word segmentation of a `SpeechHMM` in a file. Optionally, it also prints the corresponding target segmentation.

Author: Samy Bengio (bengio@idiap.ch)

112.1

SpeechHMM* **shmm**

the pointer to the SpeechHMM object

the pointer to the SpeechHMM object

112.2

bool **print_targets**

a boolean stating whether we also print the corresponding targets

a boolean stating whether we also print the corresponding targets

112.3

WordSegMeasurer (SpeechHMM* shmm_, DataSet* data_, FILE* file_, bool print_targets=true)

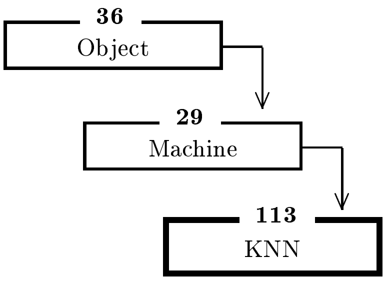
112.4

WordSegMeasurer (SpeechHMM* shmm_, DataSet* data_, const char* filename, bool print_targets=true)

113 class **KNN** : public Machine

This machine implements the K-nearest-neighbors (KNN) algorithm.

Inheritance



Public Members

113.1	int	**K**	*The number of nearest neighbors.*	317
113.2	real*	**distances**	*For each nearest neighbor, keeps its distance to the current input*	317
113.3	int*	**indices**	*For each nearest neighbor, keeps its index in the dataset*	317
113.4	DataSet*	**data**	*The dataset that contains the potential neaghbors*	317
113.5	int*	**real_examples**	*the indices of the training examples*	318
113.6		**KNN** (DataSet* data_, int K_)		318
113.7	virtual void	**setK** (int K_)	*change the value of K*	318

This machine implements the K-nearest-neighbors (KNN) algorithm. Given a dataset (in the constructor), the `forward` method returns for a given input the average of the outputs of the K nearest examples (in the input space, using the Euclidean distance). As a side effect, the machine also keep the table of distances of the K-nearest-neighbors.

Author: Samy Bengio (bengio@idiap.ch)

113.1

int **K**

The number of nearest neighbors.

The number of nearest neighbors. Controls the capacity of the machine

113.2

real* **distances**

For each nearest neighbor, keeps its distance to the current input

For each nearest neighbor, keeps its distance to the current input

113.3

int* **indices**

For each nearest neighbor, keeps its index in the dataset

For each nearest neighbor, keeps its index in the dataset

113.4

DataSet* **data**

The dataset that contains the potential neaghbors

The dataset that contains the potential neaghbors

113.5

int* **real_examples**

the indices of the training examples

the indices of the training examples

113.6

KNN (DataSet* data_, int K_)

113.7

virtual void **setK** (int K_)

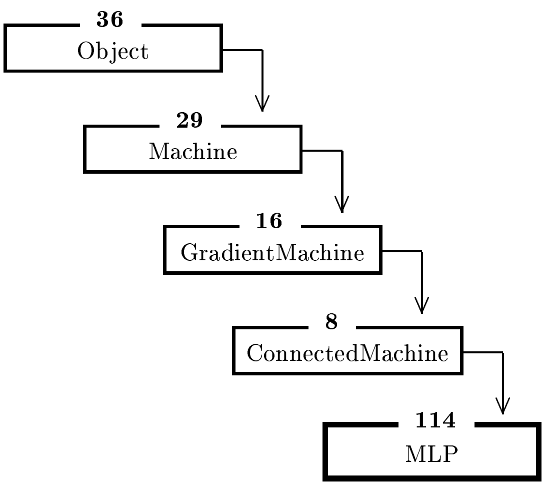
change the value of K

change the value of K

114 class **MLP** : public ConnectedMachine

This class is a simple interface to the `ConnectedMachine` *class that ca be used to build the well-known Multi Layer Perceptron type of neural networks.*

Inheritance



Public Members

This class is a simple interface to the `ConnectedMachine` class that ca be used to build the well-known Multi Layer Perceptron type of neural networks. It contains a layer of `Linear` followed by a layer of `Tanh`, followed by a layer of `Linear` and optionally

- a layer of softmax
- or a layer of sigmoid
- or a layer of log-softmax
- or a layer of tanh

Optionally, it also contains a direct connection from the inputs to the linear layer, and if you want, you can choose sparse inputs.

Options:	"inputs to outputs"	bool	connections from inputs to outputs	[false]
	"weight decay"	real	the weight decay	[0]
	"softmax outputs"	bool	softmax outputs	[false]
	"sigmoid outputs"	bool	sigmoid outputs	[false]
	"log-softmax outputs"	bool	log-softmax outputs	[false]
	"tanh outputs"	bool	tanh outputs	[false]
	"sparse inputs"	bool	sparse inputs (to use with SparseDataSet)	[false]

Author: Samy Bengio (bengio@idiap.ch) Ronan Collobert (collober@iro.umontreal.ca)

114.1

Linear* **hidden_layer**

the first `Linear` *layer*

the first `Linear` layer

114.2

SparseLinear* **sparse_hidden_layer**

the first `Linear` *layer for sparse mode*

the first `Linear` layer for sparse mode

114.3

Tanh* **hidden_tanh_layer**

the first `Tanh` *layer*

the first `Tanh` layer

114.4

Linear* **outputs_layer**

the second `Linear` *layer*

the second `Linear` layer

114.5

SparseLinear* **sparse_outputs_layer**

the second `Linear` *layer for sparse mode, if there is no hidden units*

the second `Linear` layer for sparse mode, if there is no hidden units

114.6

Softmax* **outputs_softmax_layer**

the optional second `Softmax` *layer*

the optional second `Softmax` layer

114.7

Sigmoid* **outputs_sigmoid_layer**

the optional second `Sigmoid` *layer*

the optional second `Sigmoid` layer

114.8

LogSoftmax* **outputs_log_softmax_layer**

the optional second `Softmax` *layer*

the optional second `Softmax` layer

114.9

Tanh* **outputs_tanh_layer**

the optional second `Tanh` *layer*

the optional second `Tanh` layer

114.10

int **n_hidden**

the number of hidden units

the number of hidden units

114.11

int **n_inputs**

the number of inputs

the number of inputs

114.12

int **n_outputs**

the number of outputs

the number of outputs

114.13

if this is false, add a #Tanh# layer

if this is false, add a `Tanh` layer

114.14

Flags (in order of priority if several are true) to know what will be the output

Flags (in order of priority if several are true) to know what will be the output

114.15

bool **is_sparse_inputs**

To know if the inputs are sparse

To know if the inputs are sparse

114.16

bool **inputs_to_outputs**

if this is true, add a direct connection from inputs to `Linear`

if this is true, add a direct connection from inputs to `Linear`

114.17

real **weight_decay**

the eventual weight_decay

the eventual weight_decay

114.18

Linear* **add_layer**

the direct `Linear` *layer*

the direct `Linear` layer

114.19

SumMachine* **sum_layer**

if `inputs_to_outputs` *is true, we also need a* `SumMachine`

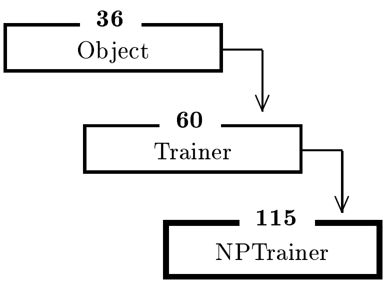
if `inputs_to_outputs` is true, we also need a `SumMachine`

115

class **NPTrainer** : public Trainer

Trainer for Non Parametric Machines.

Inheritance



Trainer for Non Parametric Machines. This trainer does nothing during training! But it can be used to test Non Parametric models such as KNN or ParzenDistributions or even select a correct hyper-parameter using cross-validation.

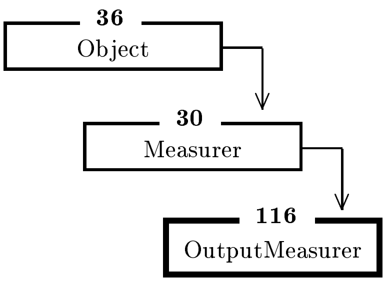
Author: Samy Bengio (bengio@idiap.ch)

116

class **OutputMeasurer** : public Measurer

This class can be used to save the outputs of a `Trainer` *in a file in such a format that it can be read again in* `Torch` *using the* `FileDataSet` *class.*

Inheritance



Public Members

116.1	List*	**inputs**	*the pointer the obtained outputs*	329
116.2	bool	**print_inputs**	*a boolean stating whether we also print the corresponding inputs* ..	329
116.3	bool	**print_targets**	*a boolean stating whether we also print the corresponding targets* .	329
116.4		**OutputMeasurer**	(List* inputs_, DataSet* data_, FILE* file_, bool print_inputs_=false, bool print_targets=true)	329
116.5		**OutputMeasurer**	(List* inputs_, DataSet* data_, const char* filename, bool print_inputs_=false, bool print_targets=true)	330

This class can be used to save the outputs of a `Trainer` in a file in such a format that it can be read again in `Torch` using the `FileDataSet` class. Optionally, it also prints the corresponding inputs and/or the corresponding targets.

Author: Samy Bengio (bengio@idiap.ch)

116.1

List* **inputs**

the pointer the obtained outputs

the pointer the obtained outputs

116.2

bool **print_inputs**

a boolean stating whether we also print the corresponding inputs

a boolean stating whether we also print the corresponding inputs

116.3

bool **print_targets**

a boolean stating whether we also print the corresponding targets

a boolean stating whether we also print the corresponding targets

116.4

OutputMeasurer (List* inputs_, DataSet* data_, FILE* file_, bool print_inputs_=false, bool print_targets=true)

116.5

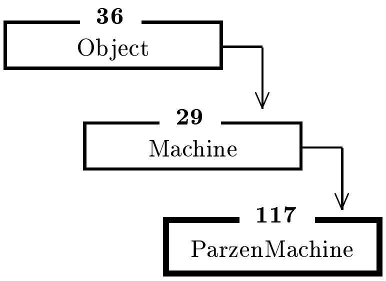
OutputMeasurer (List* inputs_, DataSet* data_, const char* filename, bool print_inputs_=false, bool print_targets=true)

117

class **ParzenMachine** : public Machine

This machine implements the Parzen Window estimator.

Inheritance



Public Members

117.1	real	**var**	*the variance used*	332
117.2	DataSet*	**data**	*The dataset that contains the training set*	332
117.3	int*	**real_examples**	*the indices of the training examples*	332
117.4	real	**denominator**	*keep the denominator*	332
117.5		**ParzenMachine** (DataSet* data_, real var_)		333
117.6	virtual void	**setVar** (real var_)	*change the value of var*	333

This machine implements the Parzen Window estimator. Given a dataset (in the constructor), the `forward` method returns for a given input the average of the outputs of the training set weighted by a Gaussian kernel distance:

$$y(x) = \frac{\sum_i t_i \exp(-\frac{||x-x_i||^2}{2var})}{\sum_i \exp(-\frac{||x-x_i||^2}{2var})}$$

The only parameter `var` is given in the constructor.

Author: Samy Bengio (bengio@idiap.ch)

117.1

real **var**

the variance used

the variance used

117.2

DataSet* **data**

The dataset that contains the training set

The dataset that contains the training set

117.3

int* **real_examples**

the indices of the training examples

the indices of the training examples

117.4

real **denominator**

keep the denominator

keep the denominator

117.5

ParzenMachine (DataSet* data_, real var_)

117.6

virtual void **setVar** (real var_)

change the value of var

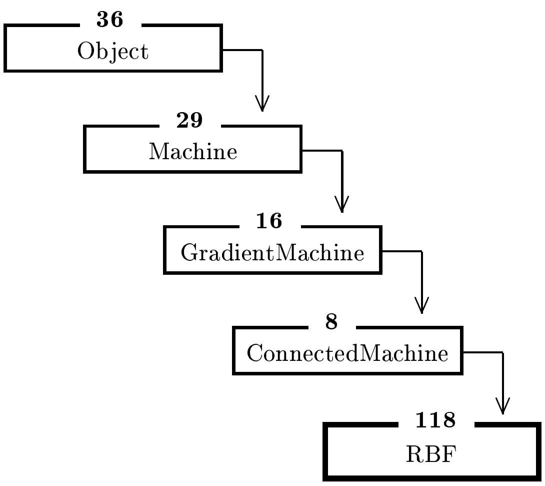
change the value of var

118

class **RBF** : public ConnectedMachine

This class is a simple interface to the `ConnectedMachine` *class that ca be used to build the well-known Radial Basis Function type of neural networks.*

Inheritance



Public Members

118.1	LogRBF*	**log_rbf_layer**	*the* `LogRBF` *layer*	335
118.2	Exp*	**exp_layer**	*the* `Exp` *layer*	335
118.3	Linear*	**outputs_layer**	*the* `Linear` *layer*	336
118.4	Tanh*	**outputs_tanh_layer**	*the optional* `Tanh` *layer*	336
118.5	int	**n_hidden**	*the number of basis functions* ...	336
118.6	bool	**is_linear_outputs**	*if this is false, add a* `Tanh` *layer*	336
118.7	bool	**inputs_to_outputs**	*if this is true, add a direct connection from inputs to* `Linear` ..	337
118.8	Linear*	**add_layer**	*the direct* `Linear` *layer*	337
118.9	SumMachine*			

This class is a simple interface to the `ConnectedMachine` class that ca be used to build the well-known Radial Basis Function type of neural networks. It contains a layer of `LogRBF` followed by a layer of `Exp`, followed by a layer of `Linear` and optionally a layer of `Tanh` for classification experiments. Optionally, it also contains a direct connection from the inputs to the linear layer.

Author: Samy Bengio (bengio@idiap.ch)

118.1

LogRBF* **log_rbf_layer**

the `LogRBF` *layer*

the `LogRBF` layer

118.2

Exp* **exp_layer**

the `Exp` *layer*

the `Exp` layer

118.3

Linear* **outputs_layer**

the `Linear` *layer*

the `Linear` layer

118.4

Tanh* **outputs_tanh_layer**

the optional `Tanh` *layer*

the optional `Tanh` layer

118.5

int **n_hidden**

the number of basis functions

the number of basis functions

118.6

bool **is_linear_outputs**

if this is false, add a `Tanh` *layer*

if this is false, add a `Tanh` layer

118.7

bool **inputs_to_outputs**

if this is true, add a direct connection from inputs to `Linear`

if this is true, add a direct connection from inputs to `Linear`

118.8

Linear* **add_layer**

the direct `Linear` *layer*

the direct `Linear` layer

118.9

SumMachine* **sum_layer**

if `inputs_to_outputs` *is true, we also need a* `SumMachine`

if `inputs_to_outputs` is true, we also need a `SumMachine`

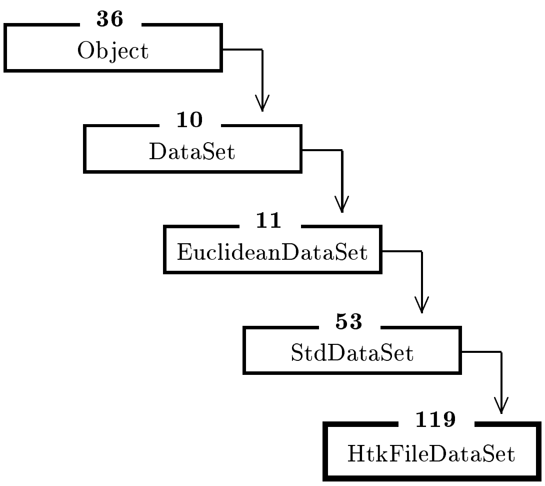
118.10

RBF (int n_inputs_, int n_hidden, int n_outputs_, bool is_linear_outputs_=true, bool inputs_to_outputs=false, EMTrainer* kmeans_trainer=NULL)

119 class **HtkFileDataSet** : public StdDataSet

Creates a `StdDataSet` *from a disk file in HTK format.*

Inheritance



Public Members

119.1	int	**n_files**	*the number of files to load*	340
119.2	IOHtk**	**htk**	*for each file, an object IOHtk is created*	340
119.3	int	**window_size**	*the size of the input window*	340
119.4		**HtkFileDataSet** (char* file, int window_size_ = 1, int max_load=-1)	*Load* `file` *in memory.*	340
119.5	virtual void	**write** (char* dir_to_save=NULL)	*save the data set on htk binary format in specifing directory*	341
119.6	virtual void			

Creates a `StdDataSet` from a disk file in HTK format. As there are no targets in HTK datasets, we decided that they point to the input vector (hence, can be used for Diabolo type of problems for instance). Moreover, a window_size can be given such that the real input vector x_t in the concatenation of the vectors from x_{t-d} to x_{t+d}, where `window_size` is equal to $2d + 1$.

Finally, it can also be used to load multiple files at once.

Author: Samy Bengio (bengio@idiap.ch)

119.1

int **n_files**

the number of files to load

the number of files to load

119.2

IOHtk** **htk**

for each file, an object IOHtk is created

for each file, an object IOHtk is created

119.3

int **window_size**

the size of the input window

the size of the input window

119.4

HtkFileDataSet (char* file, int window_size_ = 1, int max_load=-1)

Load `file` in memory.

Load `file` in memory.

- `window_size` gives the number of frames per input.
- if `max_load` > 0, it loads only the first `max_load` examples.

119.5

virtual void **write** (char* dir_to_save=NULL)

save the data set on htk binary format in specifing directory

save the data set on htk binary format in specifing directory

119.6

virtual void **prepareData** (bool* input_to_del=NULL)

Set pointers on datas stored on `IOHtk` *object and remove unused columns*

Set pointers on datas stored on `IOHtk` object and remove unused columns

119.7

virtual void **normalizeUsingDataSet** (StdDataSet* data_norm)

Normalize the `DataSet` *using the normalization values of another* `DataSet`*.*

Normalize the `DataSet` using the normalization values of another `DataSet`. If the other `DataSet` isn't normalized, nothing happens...

119.8

virtual void **normalize** ()

Normalize the database

Normalize the database

119.9

virtual int **removeUnlikelyFrames** (Distribution* likely_distr, Distribution* unlikely_distr, bool* mask = NULL)

Remove the frames considering as unlinkely and return the number of removed frames and if mask is not NULL remove unused data

Remove the frames considering as unlinkely and return the number of removed frames and if mask is not NULL remove unused data

119.10

virtual void **createMaskFromParam** (bool* mask)

Create a mask to remove log energy parameters

Create a mask to remove log energy parameters

119.11

virtual void **loadFILE** (FILE* file)

Load the normalization parameters

Load the normalization parameters

119.12

virtual void **saveFILE** (FILE* file)

Save the normalization parameters

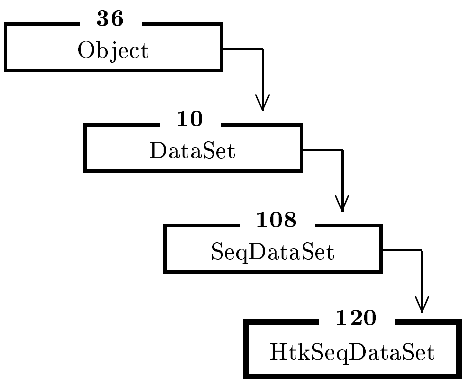
Save the normalization parameters

120

class **HtkSeqDataSet** : public SeqDataSet

This class is used to read Htk seqdatasets

Inheritance



Public Members

120.1	IOHtk**	**htk**	*the files, in htk format*	345
120.2	Dictionary*	**dict**	*used to read targets and alignments*	345
120.3	int	**n_per_frame**	*number of miliseconds per frame (used during alignment)*	345
120.4		**HtkSeqDataSet** (char* file, int max_load=-1)		345
120.5		**HtkSeqDataSet** (char** files, int n_files, int max_load=-1)		346
120.6	virtual void	**freeMemory** ()	*destroys data allocated by* `createData`	346
120.7	virtual void	**write** (char* dir_to_save)	*save the data set on htk binary format in specifing directory*	346

This class is used to read Htk seqdatasets

Author: Samy Bengio (bengio@idiap.ch) Johnny Mariethoz (Johnny.Mariethoz@idiap.ch)

120.1

IOHtk** **htk**

the files, in htk format

the files, in htk format

120.2

Dictionary* **dict**

used to read targets and alignments

used to read targets and alignments

120.3

int **n_per_frame**

number of miliseconds per frame (used during alignment)

number of miliseconds per frame (used during alignment)

120.4

HtkSeqDataSet (char* file, int max_load=-1)

120.5

HtkSeqDataSet (char** files, int n_files, int max_load=-1)

120.6

virtual void **freeMemory** ()

destroys data allocated by `createData`

destroys data allocated by `createData`

120.7

virtual void **write** (char* dir_to_save)

save the data set on htk binary format in specifing directory

save the data set on htk binary format in specifing directory

121

struct **HTKhdr**

HTK File Header

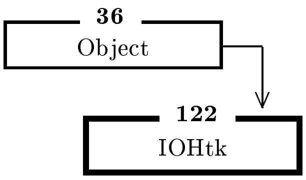
HTK File Header

122

class **IOHtk** : public Object

This class is used to read HTK objects (used for various datasets)

Inheritance



Public Members

This class is used to read HTK objects (used for various datasets)

Author: Samy Bengio (bengio@idiap.ch) Johnny Mariethoz (Johnny.Mariethoz@idiap.ch)

122.1

real* **data**

the data itself

the data itself

122.2

int **n_cols**

number of columns in the data

number of columns in the data

122.3

int **n_lines**

number of lines in the data

number of lines in the data

122.4

char* **kind**

kind of data

kind of data

122.5

char* **file_name**

file name

file name

122.6

long **samp_period**

sampling period

sampling period

122.7

IOHtk (char* file, int max_load=-1)

for each IOHtk object, specify the corresponding filename as well as the maximum number of examples to load

for each IOHtk object, specify the corresponding filename as well as the maximum number of examples to load

122.8

void **write** (char* dir_to_save=NULL)

write data on HTK binary format on the specifing directory

write data on HTK binary format on the specifing directory

122.9

virtual void **createMaskFromParam** (bool* mask)

create a mask to remove energy and corresponding delta, delta-delta

create a mask to remove energy and corresponding delta, delta-delta

122.10

char* **parmKind2Str** (ParmKind kind, char* buf)

HTK source code

HTK source code

Class Graph