Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2008-017 April 7, 2008

# LIBPMK: A Pyramid Match Toolkit

John J. Lee

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

# LIBPMK: A Pyramid Match Toolkit

John J. Lee

April 7, 2008

# Contents

# 1 Overview

LIBPMK is a C++ implementation of Grauman and Darrell's pyramid match algorithm [2, 3]. This toolkit provides a flexible framework with which developers can quickly match sets of image features and run experiments. LIBPMK provides functionality for $k$-means and hierarchical clustering, dealing with data sets too large to fit in memory, building multi-resolution histograms, quickly performing pyramid matches, and training and testing support vector machines (SVMs). This report provides a tutorial on how to use the LIBPMK code (Section 2), and gives the specifications of the LIBPMK API (Appendices A through E).

# 2 Using LIBPMK

## 2.1 Point Sets

The input data we will be working with will be organized in a `PointSetList` (the format of this file is specified in the class reference). As the name suggests, it is simply a list of `PointSet` objects, which, in turn, are simply lists of `Point` objects. `PointSet` and `PointSetList` require that all of the `Point`s contained within them have the same dimension.

`PointSetList` is an abstract class. LIBPMK provides two implementations of this class: `MutablePointSetList` and `OnDiskPointSetList`. A `MutablePointSetList` resides entirely in memory, and the user is free to add and remove `PointSet`s and `Point`s. The following code segment shows how to load data into a `MutablePointSetList` and print out the number of points there are in each point set.

```
// Load the point set list
MutablePointSetList psl;
psl.ReadFromFile("file_to_load.psl");

for (int ii = 0; ii < psl.size(); ++ii) {
   printf("Point set %d has %d points\n",
   ii, psl.point_set(ii).size());
}
```

The `OnDiskPointSetList` is read-only and is suitable for loading data that is too large to fit in memory. For performance, `OnDiskPointSetList` operates using two caches: an LRU cache which remembers the most recently used `PointSet`s, and what we call an "area" cache, which caches a contiguous block of `PointSet`s. In general, if it is known that a particular application access data in a sequential manner, it is generally more useful to make the LRU cache small (size 1) and the area cache large. If the application uses mostly random access, it is better to have a large LRU cache and a very small area cache.

```
string input_file = "file_to_load.psl";

// We are printing things sequentially, so make the
// LRU cache small (size 1)
OnDiskPointSetList psl(input_file, 1, 100);

for (int ii = 0; ii < psl.size(); ++ii) {
   printf("Point set %d has %d points\n",
          ii, psl.point_set(ii).size());
}
```

## 2.2 Clustering

The clustering implementation given by this library is not specific to LIBPMK, but it does provide the functionality required to create the pyramids and may be useful for other related tasks. LIBPMK currently only provides $k$-means clustering, but it is simple to implement one's own clustering method in this framework by subclassing the abstract `Clusterer` class.

All of the clustering code operates on lists of `PointRef` objects (rather than `PointSet`s or `PointSetList`s). This is because we may want to cluster more than just the points in one `PointSetList`, or we may only want to run the clustering on a subset of the points. A `PointRef` can be thought of as simply a pointer referencing a `Point`. The `PointSetList` class provides functionality to generate `PointRef`s for the data contained in it.

The following example shows the use of `PointSetList::ReadFromStream()` which gives some flexibility in where the data is (it does not necessarily have to be stored in a file, although in this case it uses a file stream).

```
// Load a point set list (may also use OnDiskPointSetList)
MutablePointSetList psl;
psl.ReadFromFile(input_file.c_str());

// Get a list of PointRefs to all of the points in psl
vector<PointRef> point_refs;
psl.GetPointRefs(&point_refs);

// Initialize the distance function we're using
// for the clustering
L2DistanceComputer dist_computer;

// Create a clusterer which will terminate after 500 iterations
KMeansClusterer clusterer(NUM_CLUSTERS, 500, dist_computer);
clusterer.Cluster(point_refs);
```

In the above example, we used `L2DistanceComputer`, which is built into LIBPMK. You may want to substitute your own distance computer for this. To do so, simply subclass the `DistanceComputer` (abstract) class and implement the `ComputeDistance()` function, which computes the distance between two `Point` objects.

Once the clustering is finished, the `Clusterer` object can be queried to find out which points belong in which cluster. You may also choose to save the data and read it at a later time:

```
string output_file = "k-means-output.cluster";
clusterer.WriteToFile(output_file.c_str());
clusterer.ReadFromFile(output_file.c_str());
```

This scheme works for `HierarchicalClusterer` as well, which actually uses `KMeansClusterer` to perform each level of clustering. The following code fragment will run hierarchical clustering on a `PointSetList` and save the output to a file:

```
vector<PointRef> point_refs;
psl.GetPointRefs(&point_refs);

HierarchicalClusterer clusterer;
L2DistanceComputer dist_computer;
clusterer.Cluster(num_levels, branch_factor,
                  point_refs, dist_computer);
clusterer.WriteToFile(output_file.c_str());
```

## 2.3 Creating and Using Pyramids

LIBPMK provides three different methods of turning `PointSet`s into pyramids (known also as `MultiResolutionHistogram`s). The three methods all share a common interface for creating new pyramids from `PointSet`s, but they are prepared in different ways. For the following examples, suppose we have loaded a `PointSetList` called `psl`.

### 2.3.1 Uniform bin sizes

```
UniformPyramidMaker upm;
upm.Preprocess(psl, finest_side_length, side_length_factor,
               discretize_order, true, true);
```

There are a number of parameters that `UniformPyramidMaker` requires. `finest_side_length` is the length of one side of the smallest bin. `side_length_factor`

is the amount the length of a side of a bin increases from one level to the next. The `discretize_order` parameter is particularly important because it can greatly affect the shape of the resulting pyramid. This parameter multiplies all of the feature values by $10^{\texttt{discretize\_order}}$. We typically set it in such a way that the resulting features are on the order of $10^2$. The final two boolean parameters specify whether we want to use random shifts, and whether to apply the same shifts to each level, respectively.

Just like `Clusterer`, it is possible to avoid doing redundant computations. `UniformPyramidMaker` supports `ReadFromStream()` and `WriteToStream()` functions. `ReadFromStream()` may be called on already-preprocessed data in lieu of calling `Preprocess()`.

### 2.3.2 Vocabulary-guided: global weights

```
HierarchicalClusterer clusterer;
L2DistanceComputer dist;
clusterer.Cluster(...);

GlobalVGPyramidMaker vgpm(clusterer, dist);
vgpm.Preprocess(psl);
```

The vocabulary-guided pyramids require data from `HierarchicalClusterer`. The above code will cluster a `PointSetList` and initialize a `PyramidMaker` with global weights. Again, since `GlobalVGPyramidMaker` has a `ReadFromStream()` and `WriteToStream()` function, data can be saved to a file and read later to avoid performing redundant computations.

### 2.3.3 Vocabulary-guided: input-specific weights

```
HierarchicalClusterer clusterer;
L2DistanceComputer dist;
clusterer.Cluster(...);
InputSpecificVGPyramidMaker vgpm(clusterer, dist);
```

With input-specific weights, no preprocessing is needed, since the weights for each bin depend on the `PointSet` we are converting.

### 2.3.4 Building pyramids with a `PyramidMaker`

Once we have set up a `PyramidMaker`, we can start creating pyramids from `PointSets`:

```
MultiResolutionHistogram* pyramid =
   pyramid_maker.MakePyramid(point_set);
```

Or, given a `PointSetList`, we can build them all at once:

```
vector<MultiResolutionHistogram*> vec =
   pyramid_maker.MakePyramids(psl);
```

Note that `PyramidMaker` allocates memory for these pyramids and returns pointers to them. It is the caller's responsibility to free the memory.

### 2.3.5 Matching Pyramids

The pyramid matching framework is simple to use. Suppose we have two `MultiResolutionHistogram` objects `mrh1` and `mrh2` (read from disk, computed by `PyramidMaker`, or by any other means):

```
double value = PyramidMatcher::GetPyramidMatchCost(
   mrh1, mrh2, BIN_WEIGHT_GLOBAL);
```

The third argument should reflect the method of creating the pyramids (`BIN_WEIGHT_INPUT_SPECIFIC` should be used for input-specific weights). For uniform pyramids, use `BIN_WEIGHT_GLOBAL`. `GetPyramidMatchCost()` takes the weight of the bin to be its size.

We compute the similarity by taking the weight of the bin to be the reciprocal of the bin size. This is also built-in:

```
double value = PyramidMatcher::GetPyramidMatchSimilarity(
   mrh1, mrh2, BIN_WEIGHT_GLOBAL);
```

## 2.4 Selectors and Experiments

To facilitate running experiments, LIBPMK provides the `ExampleSelector` and `Experiment` classes. `ExampleSelector` is an abstract class which will choose training and testing examples. `Experiment` will train a model and make predictions on the test examples. The actual implementations of `ExampleSelector` and `Experiment` will vary according to the experiment, but these classes are meant to provide a general framework in which they can be run. In the following example, we will be using `RandomSelector`, which chooses a random sample of the data to use as training examples, and `SVMExperiment`, which wraps around LIBSVM [1].

Suppose we have a database of $n$ labeled images. Let `labels` be a vector of integers of size $n$, where each element in the vector is the label of that image. We use `RandomSelector` as follows:

```
RandomSelector selector(labels, num_training_examples_per_class);
vector<LabeledIndex> train_ex(selector.GetTrainingExamples());
vector<LabeledIndex> test_ex(selector.GetTestingExamples());
```

If your needs are different, it is easy to create your own subclass of `ExampleSelector`. The only function to implement is `SelectExamples()`; see the class reference for details on how to do this. It is also instructive to review the implementation of `RandomSelector`.

Once the training and test examples are selected, it can be used with `SVMExperiment` as follows:

```
SVMExperiment experiment(train_ex, test_ex, kernel, C);
experiment.Train();
int num_correct = experiment.Test();
```

`Test()` reports the total number of test examples that were correctly classified. However, for more detailed information, we can query the `SVMExperiment` object for more data, such as what each prediction was.

# A LIBPMK Hierarchical Index

## A.1 LIBPMK Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# B LIBPMK Class Index

## B.1 LIBPMK Class List

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

# C LIBPMK Directory Documentation

## C.1 clustering/ Directory Reference

### Files

- file clusterer.cc
- file clusterer.h
- file hierarchical-clusterer.cc
- file hierarchical-clusterer.h
- file k-means-clusterer.cc
- file k-means-clusterer.h

## C.2 experiment/ Directory Reference

### Files

- file example-selector.cc
- file example-selector.h
- file experiment.cc
- file experiment.h
- file random-selector.cc
- file random-selector.h
- file svm-experiment.cc
- file svm-experiment.h

## C.3 histograms/ Directory Reference

### Files

- file bin.cc

- file bin.h
- file histogram.cc
- file histogram.h
- file multi-resolution-histogram.cc
- file multi-resolution-histogram.h

## C.4 kernel/ Directory Reference

### Files

- file kernel-matrix.cc
- file kernel-matrix.h
- file matrix.cc
- file matrix.h

## C.5 point_set/ Directory Reference

### Files

- file mutable-point-set-list.cc
- file mutable-point-set-list.h
- file on-disk-point-set-list.cc
- file on-disk-point-set-list.h
- file point-ref.cc
- file point-ref.h
- file point-set-list.cc
- file point-set-list.h
- file point-set.cc
- file point-set.h
- file point.cc
- file point.h

## C.6 pyramids/ Directory Reference

### Files

- file global-vg-pyramid-maker.cc
- file global-vg-pyramid-maker.h

- file input-specific-vg-pyramid-maker.cc
- file input-specific-vg-pyramid-maker.h
- file normalized-uniform-pyramid-maker.cc
- file normalized-uniform-pyramid-maker.h
- file pyramid-maker.cc
- file pyramid-maker.h
- file pyramid-matcher.cc
- file pyramid-matcher.h
- file uniform-pyramid-maker.cc
- file uniform-pyramid-maker.h
- file vg-pyramid-maker.cc
- file vg-pyramid-maker.h

## C.7 tools/ Directory Reference

### Files

- file average-exponent-kernel.cc
- file caltech101-splitter.cc
- file clean-psl.cc
- file clusters-to-pyramids.cc
- file dbg-disk-ptsets.cc
- file distance-kernel.cc
- file exponent-kernel.cc
- file exponential-pyramid-match-kernel.cc
- file extract-classes-from-psl.cc
- file extract-subset-from-psl.cc
- file feature-value-discretizer.cc
- file hierarchical-cluster-point-set.cc
- file hierarchical-histogram-maker.cc
- file kernel-adder.cc
- file kernel-weighter.cc
- file large-upmk-example.cc
- file normalize-kernel.cc
- file partial-pyramid-match-kernel.cc
- file pointset-merger.cc
- file pointsets-to-uniform-pyramids.cc
- file psl-sampler.cc

- file pyramid-match-kernel.cc
- file pyramid-match-self-similarity.cc
- file pyramid-match-split.cc
- file random-sampler.cc
- file random-sampling-merger.cc

## C.8 tree/ Directory Reference

### Files

- file point-tree-node.cc
- file point-tree-node.h
- file sparse-tree-node.cc
- file sparse-tree-node.h
- file sparse-tree.cc
- file sparse-tree.h
- file tree-node.cc
- file tree-node.h
- file tree.cc
- file tree.h

## C.9 util/ Directory Reference

### Files

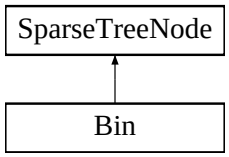
- file bin-weight-scheme.h
- file distance-computer.cc
- file distance-computer.h
- file labeled-index.cc
- file labeled-index.h
- file sparse-vector.cc
- file sparse-vector.h

# D LIBPMK Class Documentation

## D.1 Bin Class Reference

`#include <bin.h>`

Inheritance diagram for Bin::




### D.1.1 Detailed Description

Encapsulates a histogram bin.

A Bin is implemented as a SparseTreeNode with two data members: a size and a count.

#### Public Member Functions

- Bin ()
- Bin (const LargeIndex &index)
- virtual ~Bin ()
- void set_size (double size)

  *Sets the size of the bin.*

- double size () const

  *Get the size of the bin.*

- void set_count (double count)

  *Set the count.*

- double count () const

  *Get the current count.*

- virtual void ReadData (istream &input_stream)

  *Read the data, excluding the index, from a stream.*

- virtual void WriteData (ostream &output_stream) const

  *Write the data, excluding the index, to a stream.*

- virtual void Combine (const SparseTreeNode &other)

  *Combine data from two Bins.*

- const LargeIndex & index () const
- SparseTreeNode * parent ()
- SparseTreeNode * prev_sibling ()
- SparseTreeNode * next_sibling ()
- SparseTreeNode * first_child ()
- SparseTreeNode * last_child ()
- void set_parent (SparseTreeNode *parent)
- void set_prev_sibling (SparseTreeNode *sibling)
- void set_next_sibling (SparseTreeNode *sibling)
- void set_first_child (SparseTreeNode *child)
- void set_last_child (SparseTreeNode *child)
- void set_index (const LargeIndex &index)
- bool has_parent () const
- bool has_prev_sibling () const
- bool has_next_sibling () const
- bool has_child () const
- void ReadFromStream (istream &input_stream)

  *Read the data, including the index, from a stream.*

- void WriteToStream (ostream &output_stream) const

  *Write the data, including the index, to a stream.*

- bool operator< (const SparseTreeNode &other) const

  *Returns true if this node's index is smaller than that of <other>.*

### Static Public Member Functions

- static bool CompareNodes (const SparseTreeNode *one, const SparseTreeNode *two)

  *Returns true if <one>'s index is smaller than that of <two>.*

### D.1.2 Constructor & Destructor Documentation

#### D.1.2.1 Bin ()

#### D.1.2.2 Bin (const LargeIndex & *index*)

#### D.1.2.3 virtual ~Bin () [inline, virtual]

### D.1.3 Member Function Documentation

#### D.1.3.1 void set_size (double *size*) [inline]

Sets the size of the bin.

#### D.1.3.2 double size () const [inline]

Get the size of the bin.

#### D.1.3.3 void set_count (double *count*) [inline]

Set the count.

#### D.1.3.4 double count () const [inline]

Get the current count.

#### D.1.3.5 void ReadData (istream & *input_stream*) [virtual]

Read the data, excluding the index, from a stream.

Implements SparseTreeNode.

#### D.1.3.6 void WriteData (ostream & *output_stream*) const [virtual]

Write the data, excluding the index, to a stream.

Implements SparseTreeNode.

#### D.1.3.7 void Combine (const SparseTreeNode & *other*) [virtual]

Combine data from two Bins.

To combine bins, we take the max of the sizes, and add the counts.

Implements SparseTreeNode.

#### D.1.3.8 const LargeIndex& index () const [inline, inherited]

#### D.1.3.9 SparseTreeNode* parent () [inline, inherited]

#### D.1.3.10 SparseTreeNode* prev_sibling () [inline, inherited]

#### D.1.3.11 SparseTreeNode* next_sibling () [inline, inherited]

#### D.1.3.12 SparseTreeNode* first_child () [inline, inherited]

#### D.1.3.13 SparseTreeNode* last_child () [inline, inherited]

#### D.1.3.14 void set_parent (SparseTreeNode * *parent*) [inline, inherited]

#### D.1.3.15 void set_prev_sibling (SparseTreeNode * *sibling*) [inline, inherited]

#### D.1.3.16 void set_next_sibling (SparseTreeNode * *sibling*) [inline, inherited]

#### D.1.3.17 void set_first_child (SparseTreeNode * *child*) [inline, inherited]

#### D.1.3.18 void set_last_child (SparseTreeNode * *child*) [inline, inherited]

#### D.1.3.19 void set_index (const LargeIndex & *index*) [inline, inherited]

#### D.1.3.20 bool has_parent () const [inline, inherited]

#### D.1.3.21 bool has_prev_sibling () const [inline, inherited]

#### D.1.3.22 bool has_next_sibling () const [inline, inherited]

#### D.1.3.23 bool has_child () const [inline, inherited]

#### D.1.3.24 void ReadFromStream (istream & *input_stream*) [inherited]

Read the data, including the index, from a stream.

Calling this will override the node's current index. The format is:

- (int32_t) index_size
- (index_size * int32_t) the index

It will then call ReadData().

**See also:**

ReadData()

#### D.1.3.25 void WriteToStream (ostream & *output_stream*) const [inherited]

Write the data, including the index, to a stream.

This will simply write the index to the stream, followed by WriteData().

**See also:**

WriteData().

#### D.1.3.26 bool operator< (const SparseTreeNode & *other*) const [inherited]

Returns true if this node's index is smaller than that of <other>.

#### D.1.3.27 bool CompareNodes (const SparseTreeNode * *one*, const SparseTreeNode * *two*) [static, inherited]

Returns true if <one>'s index is smaller than that of <two>.

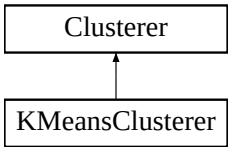
The documentation for this class was generated from the following files:

- histograms/bin.h
- histograms/bin.cc

## D.2 Clusterer Class Reference

`#include <clusterer.h>`

Inheritance diagram for Clusterer::




### D.2.1 Detailed Description

Abstract interface for a flat clusterer.

There are two ways for a Clusterer to get its data: either by (1) performing an actual clustering computation, or (2) by reading pre-clustered data from a stream. This setup allows one to cluster some data, and then save the results to a file so it can be later read and further processed.

To create your own clusterer, all you need to do is implement the DoClustering() method. The I/O is handled automatically.

#### Public Member Functions

- Clusterer ()
- virtual ∼Clusterer ()
- void Cluster (const vector< PointRef > &data)

  *Performs the clustering and stores the result internally.*

- void Cluster (const vector< const Point * > &data)

  *Performs the clustering and stores the result internally.*

- const PointSet & centers () const

  *Output the cluster centers.*

- int centers_size () const

  *Return the number of cluster centers.*

- int membership (int index) const

  *Return the membership of the <index>th point.*

- int membership_size () const

  *Return the number of members. Equivalent to the number of points that were clustered.*

- void WriteToStream (ostream &output_stream) const

*Write the clustering data to a stream.*

- void WriteToFile (const char *output_filename) const

  *Write the clustering data to a file.*

- void ReadFromStream (istream &input_stream)

  *Read clustering data from a stream.*

- void ReadFromFile (const char *input_filename)

  *Read clustering data from a file.*

**Protected Member Functions**

- virtual void DoClustering (const vector< const Point * > &data)=0

  *Performs the actual clustering.*

**Protected Attributes**

- auto_ptr< PointSet > cluster_centers_
- vector< int > membership_
- bool done_

### D.2.2 Constructor & Destructor Documentation

#### D.2.2.1 Clusterer ()

#### D.2.2.2 virtual ~Clusterer () [inline, virtual]

### D.2.3 Member Function Documentation

#### D.2.3.1 void Cluster (const vector< PointRef > & *data*)

Performs the clustering and stores the result internally.

To avoid potential memory problems, Clusterers do not operate on PointSetLists or PointSets directly. Rather, they simply shuffle PointRefs around.

**See also:**

PointSetList::GetPointRefs()

#### D.2.3.2 void Cluster (const vector< const Point * > & *data*)

Performs the clustering and stores the result internally.

To avoid potential memory problems, Clusterers do not operate on PointSetLists or PointSets directly. Rather, they simply shuffle pointers around.

#### D.2.3.3 const PointSet & centers () const

Output the cluster centers.

This requires Cluster() or ReadFromStream() to have been called first. It returns the set of all cluster centers as Points.

#### D.2.3.4 int centers_size () const

Return the number of cluster centers.

This requires Cluster() or ReadFromStream() to have been called first. It reutnrs the number of cluster centers.

#### D.2.3.5 int membership (int *index*) const

Return the membership of the <index>th point.

The return value gives the ID of the cluster that this point belongs to. "ID" in this sense means an index into the PointSet returned by centers().

#### D.2.3.6 int membership_size () const

Return the number of members. Equivalent to the number of points that were clustered.

#### D.2.3.7 void WriteToStream (ostream & *output_stream*) const

Write the clustering data to a stream.

Requires Cluster() or ReadFromStream() to have been called first. Output format:

- (int32) C, the number of cluster centers
- (int32) P, the total number of clustered points
- (int32) D, the dimension of the Points

- (Point) center 0
- (Point) center 1
- (Point) ...
- (Point) center C-1
- (int32) the cluster to which point 1 belongs
- (int32) the cluster to which point 2 belongs
- (int32) ...
- (int32) the cluster to which point P belongs

  The clustered points themselves are not written to the stream, only the centers and membership data. It is assumed that the caller of Cluster() already has access to those points anyway. This function aborts if the stream is bad.

#### D.2.3.8 void WriteToFile (const char * *output_filename*) const

Write the clustering data to a file.

#### D.2.3.9 void ReadFromStream (istream & *input_stream*)

Read clustering data from a stream.

Can be called in lieu of Cluster(). If this is called after Cluster(), all of the previous data is cleared before reading the new data. For the file format, see WriteToStream. This function aborts if the stream is bad.

**See also:**

WriteToStream.

#### D.2.3.10 void ReadFromFile (const char * *input_filename*)

Read clustering data from a file.

#### D.2.3.11 virtual void DoClustering (const vector< const Point * > & *data*) [protected, pure virtual]

Performs the actual clustering.

DoClustering is responsible for three things:

1. Filling up cluster_centers_
2. Filling up membership_ with a number of elements equal to data.size()
3. Setting done_ to true.

Implemented in KMeansClusterer.

#### D.2.4 Member Data Documentation

##### D.2.4.1 auto_ptr<PointSet> cluster_centers_ [protected]

##### D.2.4.2 vector<int> membership_ [protected]

##### D.2.4.3 bool done_ [protected]

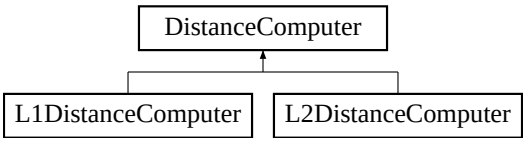
The documentation for this class was generated from the following files:

- clustering/clusterer.h
- clustering/clusterer.cc

### D.3 DistanceComputer Class Reference

`#include <distance-computer.h>`

Inheritance diagram for DistanceComputer::




#### D.3.1 Detailed Description

An abstract interface for computing distance between two Points.

#### Public Member Functions

- virtual ~DistanceComputer ()
- virtual double ComputeDistance (const Point &f1, const Point &f2) const=0

*Compute distance.*

- virtual double ComputeDistance (const Point &f1, const Point &f2, double max_distance) const

  *Quickly compute min(actual_distance(f1, f2), max_distance).*

### D.3.2 Constructor & Destructor Documentation

#### D.3.2.1 ~DistanceComputer () [virtual]

### D.3.3 Member Function Documentation

#### D.3.3.1 virtual double ComputeDistance (const Point & *f1*, const Point & *f2*) const [pure virtual]

Compute distance.

Implemented in L1DistanceComputer, and L2DistanceComputer.

#### D.3.3.2 double ComputeDistance (const Point & *f1*, const Point & *f2*, double *max_distance*) const [virtual]

Quickly compute min(actual_distance(f1, f2), max_distance).

This is the same as ComputeDistance, except it will stop calculating things when it knows the distance will be greater than max_distance.

Reimplemented in L1DistanceComputer, and L2DistanceComputer.

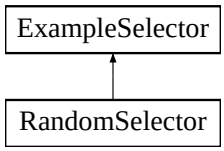
The documentation for this class was generated from the following files:

- util/distance-computer.h
- util/distance-computer.cc

## D.4 ExampleSelector Class Reference

`#include <example-selector.h>`

Inheritance diagram for ExampleSelector::




### D.4.1 Detailed Description

Splits a list of labels into a training and test set.

ExampleSelector does not specify any particular method of picking a training and test set, nor does it care about the data itself. The only input is a vector of labels, where each element in the vector corresponds to one image (or other data point).

To use ExampleSelector, simply subclass it and implement SelectExamples(). Within this function, designate training and testing examples by using AddTrainingExample() and AddTesting-Example(). There are a number of protected member functions at your disposal that could help make things easier.

#### Public Member Functions

- ExampleSelector (const vector< int > &labels)
- virtual ~ExampleSelector ()
- vector< LabeledIndex > GetTrainingExamples ()

  *Returns a vector of LabeledIndices of training instances sorted by index.*

- vector< LabeledIndex > GetTestingExamples ()

  *Returns a vector of LabeledIndices of testing instances sorted by index.*

- vector< int > GetUniqueLabels ()

  *Returns a vector containing all the unique labels seen. Sorting is NOT guaranteed.*

- int GetNumExamples () const

  *Returns the total number of examples there are.*

#### Protected Member Functions

- const vector< LabeledIndex > & GetExamples () const

  *Get all of the examples (in the same order as given when constructed).*

- vector< LabeledIndex > InvertSelection (const vector< LabeledIndex > &selection) const

  *Returns all instances are NOT in <selection>.*

- vector< LabeledIndex > InvertSelection (const vector< LabeledIndex > &selection, const vector< LabeledIndex > &superset) const

  *Returns all instances in <superset> that are NOT in <selection>.*

- vector< LabeledIndex > RandomSample (int n, const vector< LabeledIndex > &selection) const

  *Return a random sample of size <n> of <selection>.*

- vector< LabeledIndex > GetInstancesWithLabel (int label, const vector< LabeledIndex > &selection) const

  *Gets all elements in <selection> with the given <label>.*

- void AddTrainingExample (LabeledIndex index)

  *Add a LabeledIndex to the list of designated training examples.*

- void AddTestingExample (LabeledIndex index)

  *Add a LabeledIndex to the list of designated testing examples.*

- virtual void SelectExamples ()=0

  *Responsible for calling AddTrainingExample and AddTestingExample to generate the lists.*

### D.4.2 Constructor & Destructor Documentation

#### D.4.2.1 ExampleSelector (const vector< int > & *labels*)

#### D.4.2.2 virtual ~ExampleSelector () `[inline, virtual]`

### D.4.3 Member Function Documentation

#### D.4.3.1 vector< LabeledIndex > GetTrainingExamples ()

Returns a vector of LabeledIndices of training instances sorted by index.

#### D.4.3.2 vector< LabeledIndex > GetTestingExamples ()

Returns a vector of LabeledIndices of testing instances sorted by index.

#### D.4.3.3 vector< int > GetUniqueLabels ()

Returns a vector containing all the unique labels seen. Sorting is NOT guaranteed.

#### D.4.3.4 int GetNumExamples () const

Returns the total number of examples there are.

#### D.4.3.5 const vector< LabeledIndex > & GetExamples () const [protected]

Get all of the examples (in the same order as given when constructed).

#### D.4.3.6 vector< LabeledIndex > InvertSelection (const vector< LabeledIndex > & *selection*) const [protected]

Returns all instances are NOT in <selection>.

<selection> must be sorted by index. The output will be sorted by index.

#### D.4.3.7 vector< LabeledIndex > InvertSelection (const vector< LabeledIndex > & *selection*, const vector< LabeledIndex > & *superset*) const [protected]

Returns all instances in <superset> that are NOT in <selection>.

<selection> and <superset> must be sorted by index. The output will be sorted by index.

#### D.4.3.8 vector< LabeledIndex > RandomSample (int *n*, const vector< LabeledIndex > & *selection*) const [protected]

Return a random sample of size <n> of <selection>.

<selection> must be sorted by index. If the size of <selection> is less than n, this function will return a vector of size selection.size(). Output is sorted by index.

#### D.4.3.9 vector< LabeledIndex > GetInstancesWithLabel (int *label*, const vector< LabeledIndex > & *selection*) const [protected]

Gets all elements in <selection> with the given <label>.

<selection> must be sorted. The output will be sorted.

#### D.4.3.10 void AddTrainingExample (LabeledIndex *index*) [protected]

Add a LabeledIndex to the list of designated training examples.

#### D.4.3.11 void AddTestingExample (LabeledIndex *index*) [protected]

Add a LabeledIndex to the list of designated testing examples.

#### D.4.3.12 virtual void SelectExamples () [protected, pure virtual]

Responsible for calling AddTrainingExample and AddTestingExample to generate the lists.

Implemented in RandomSelector.

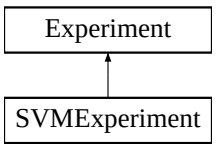
The documentation for this class was generated from the following files:

- experiment/example-selector.h
- experiment/example-selector.cc

## D.5 Experiment Class Reference

`#include <experiment.h>`

Inheritance diagram for Experiment::




### D.5.1 Detailed Description

Encapsulates training/testing of any method involving a kernel.

All that you need to implement are Train() and Test(). When implementing Test(), the way you report a new prediction is via SetPrediction(). Make sure to call SetPrediction() on each testing example.

#### Public Member Functions

- Experiment (vector< LabeledIndex > training, vector< LabeledIndex > testing, const KernelMatrix &kernel)

- Experiment (vector< LabeledIndex > training, const KernelMatrix &training_matrix, vector< LabeledIndex > testing, const Matrix &testing_matrix)
- virtual ~Experiment ()
- virtual void Train ()=0

  *Train a model (if applicable).*

- virtual int Test ()=0

  *Make predictions for each testing example.*

- int GetPrediction (int test_index) const

  *Returns the predicted value of the <test_index>th test example.*

- int GetNumCorrect () const

  *Get the total number of testing examples that were correctly classified.*

- int GetNumCorrect (int label) const

  *Get the number of testing examples that had label <label> that were also correctly classified.*

- int GetNumTestExamples () const

  *Get the total number of test examples.*

- int GetNumTestExamples (int label) const

  *Get the number of text examples with label <label>.*

- double GetAccuracy () const

  *Same as GetNumCorrect() / GetNumTestExamples().*

- double GetAccuracy (int label) const

  *Same as GetNumCorrect(label) / GetNumTestExamples(label).*

### Protected Member Functions

- double GetKernelValue (int row, int col) const

  *Get a kernel value (wrapper for KernelMatrix).*

- double GetKernelValue (const LabeledIndex &row, const LabeledIndex &col) const

  *Get the kernel value corresponding to the given LabeledIndices.*

- void SetPrediction (int test_index, int prediction)

**Protected Attributes**

- vector< LabeledIndex > training_
- vector< LabeledIndex > testing_

### D.5.2 Constructor & Destructor Documentation

#### D.5.2.1 Experiment (vector< LabeledIndex > *training*, vector< LabeledIndex > *testing*, const KernelMatrix & *kernel*)

<kernel> includes pairwise kernel values for all data (both training and testing). The LabeledIndices in <training> and <testing> specify which row of the kernel to look at.

#### D.5.2.2 Experiment (vector< LabeledIndex > *training*, const KernelMatrix & *training_matrix*, vector< LabeledIndex > *testing*, const Matrix & *testing_matrix*)

<training_matrix> is a kernel matrix for training examples only. Let N be the number of training examples. Then <testing_matrix> is a NxM Matrix where M is the number of test examples, and the testing[i][j] is the kernel value between the i'th training example and the j'th test example. <training> must be N-dimensional and <testing> must be M-dimensional.

#### D.5.2.3 virtual ~Experiment () `[inline, virtual]`

### D.5.3 Member Function Documentation

#### D.5.3.1 virtual void Train () `[pure virtual]`

Train a model (if applicable).

Implemented in SVMExperiment.

#### D.5.3.2 virtual int Test () `[pure virtual]`

Make predictions for each testing example.

Returns the number of test examples that were correct. When you implement Test(), you must report results via SetPrediction().

Implemented in SVMExperiment.

#### D.5.3.3 int GetPrediction (int *test_index*) const

Returns the predicted value of the <test_index>th test example.

Can only call this after Test() is called.

#### D.5.3.4 int GetNumCorrect () const

Get the total number of testing examples that were correctly classified.

#### D.5.3.5 int GetNumCorrect (int *label*) const

Get the number of testing examples that had label <label> that were also correctly classified.

#### D.5.3.6 int GetNumTestExamples () const

Get the total number of test examples.

#### D.5.3.7 int GetNumTestExamples (int *label*) const

Get the number of text examples with label <label>.

#### D.5.3.8 double GetAccuracy () const

Same as GetNumCorrect() / GetNumTestExamples().

#### D.5.3.9 double GetAccuracy (int *label*) const

Same as GetNumCorrect(label) / GetNumTestExamples(label).

#### D.5.3.10 double GetKernelValue (int *row*, int *col*) const [protected]

Get a kernel value (wrapper for KernelMatrix).

#### D.5.3.11 double GetKernelValue (const LabeledIndex & *row*, const LabeledIndex & *col*) const [protected]

Get the kernel value corresponding to the given LabeledIndices.

#### D.5.3.12 void SetPrediction (int *test_index*, int *prediction*) [protected]

Call this to tell Experiment's internals that the test example at <test_index> was classified as <prediction>.

### D.5.4 Member Data Documentation

#### D.5.4.1 vector<LabeledIndex> training_ [protected]

#### D.5.4.2 vector<LabeledIndex> testing_ [protected]

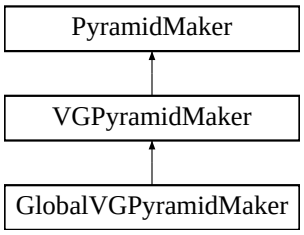
The documentation for this class was generated from the following files:

- experiment/experiment.h
- experiment/experiment.cc

## D.6 GlobalVGPyramidMaker Class Reference

`#include <global-vg-pyramid-maker.h>`

Inheritance diagram for GlobalVGPyramidMaker::




### D.6.1 Detailed Description

Makes pyramids with bin sizes determined by a particular set of points.

### Public Member Functions

- GlobalVGPyramidMaker (const HierarchicalClusterer &clusterer, const DistanceComputer &distance_computer)
- ~GlobalVGPyramidMaker ()
- void Preprocess (const PointSetList &point_sets)

  *Initialize the global bin sizes.*

- virtual MultiResolutionHistogram * MakePyramid (const PointSet &point_set)

  *Turn a single PointSet into a MultiResolutionHistogram.*

- void ReadFromStream (istream &input_stream)

  *Initialize the global bin sizes.*

- void ReadFromFile (const char *filename)

  *Initialize the global bin sizes.*

- void WriteToStream (ostream &output_stream) const

  *Write the global bin sizes to a stream.*

- void WriteToFile (const char *filename) const

  *Write the global bin sizes to a file.*

- vector< MultiResolutionHistogram * > MakePyramids (const PointSetList &psl)

  *Turn a list of PointSets into a bunch of MultiResolutionHistograms.*

**Protected Member Functions**

- virtual bool GetMembershipPath (const Point &f, LargeIndex *out_path, vector< double > *out_distances)

  *Get the membership data relative to the global pyramid data.*

**Protected Attributes**

- const HierarchicalClusterer & clusterer_
- const Tree< PointTreeNode > & centers_

  *The centers extracted from clusterer_.*

- const DistanceComputer & distance_computer_

### D.6.2 Constructor & Destructor Documentation

#### D.6.2.1 GlobalVGPyramidMaker (const HierarchicalClusterer & *clusterer*, const DistanceComputer & *distance_computer*)

#### D.6.2.2 ~GlobalVGPyramidMaker ()

### D.6.3 Member Function Documentation

#### D.6.3.1 void Preprocess (const PointSetList & *point_sets*)

Initialize the global bin sizes.

This takes O(n^2) time, where n is the total number of points in <point_sets>. This is because at the top level bin, it computes the furthest pair between any of the two points to set the bin size.

Rather than doing this every single time, you may also call ReadFromStream() which will read already-preprocessed data.

#### D.6.3.2 MultiResolutionHistogram * MakePyramid (const PointSet & *point_set*) [virtual]

Turn a single PointSet into a MultiResolutionHistogram.

This function allocates memory on its own. It is up to the caller to free it.

Implements VGPyramidMaker.

#### D.6.3.3 void ReadFromStream (istream & *input_stream*)

Initialize the global bin sizes.

Can be called in lieu of Preprocess(). Aborts if the stream is bad.

#### D.6.3.4 void ReadFromFile (const char * *filename*)

Initialize the global bin sizes.

**See also:**

ReadFromStream

#### D.6.3.5 void WriteToStream (ostream & *output_stream*) const

Write the global bin sizes to a stream.

**This output file format is not compatible with that of libpmk-1.x.**

Stream format:

- (int32) The number of nodes, n
- For each node n:
  - (int32) The node's ID (same as that in the HierarchicalClusterer)

– (double) The size of the node

Requires Preprocess() or ReadFromStream() to be called first. Aborts if the stream is bad.

#### D.6.3.6 void WriteToFile (const char * *filename*) const

Write the global bin sizes to a file.

**See also:**

WriteToStream

#### D.6.3.7 bool GetMembershipPath (const Point & *f*, LargeIndex * *out_path*, vector< double > * *out_distances*) [protected, virtual]

Get the membership data relative to the global pyramid data.

This function is overridden so that only paths that were present in the point sets that this was constructed with can appear. Otherwise, this function does the same thing as its parent VGPyramidMaker::GetMembershipPath().

**See also:**

VGPyramidMaker::GetMembershipPath()

Reimplemented from VGPyramidMaker.

#### D.6.3.8 vector< MultiResolutionHistogram * > MakePyramids (const PointSetList & *psl*) [inherited]

Turn a list of PointSets into a bunch of MultiResolutionHistograms.

It is up to the caller to free the memory.

### D.6.4 Member Data Documentation

#### D.6.4.1 const HierarchicalClusterer& clusterer_ [protected, inherited]

#### D.6.4.2 const Tree<PointTreeNode>& centers_ [protected, inherited]

The centers extracted from clusterer_.

#### D.6.4.3 const DistanceComputer& distance_computer_ [protected, inherited]

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

- pyramids/global-vg-pyramid-maker.h
- pyramids/global-vg-pyramid-maker.cc

## D.7 HierarchicalClusterer Class Reference

`#include <hierarchical-clusterer.h>`

### D.7.1 Detailed Description

Hierarchical K-Means clusterer.

This clusterer does not use the Clusterer interface because its output involves multiple levels. However, similar to Clusterer, HierarchicalClusterer has the ability to get its data either by actually doing hierarchical K-means, or by reading already-processed data. The difference is that the returned cluster centers are a PointSetList, where each element in the list gives the centers for a particular level.

### Public Member Functions

- HierarchicalClusterer ()
- const Tree< PointTreeNode > & centers () const

  *Get the cluster centers.*

- int membership (int index) const

  *Report which leaf the <index>th point belongs to.*

- int membership_size () const

  *Return the number of points that were clustered.*

- void IdentifyMemberIDPath (int index, list< int > *out) const

  *Get the set of node IDs in the hierarchy that a point belongs to.*

- void IdentifyMemberTreePath (int index, list< int > *out) const

  *Get a path down the hierarchy that a point belongs to.*

- void Cluster (int num_levels, int branch_factor, const vector< PointRef > &points, const DistanceComputer &distance_computer)

  *Performs the actual clustering and stores the result internally.*

- void WriteToStream (ostream &output_stream) const

  *Write the clustered data to a stream.*

- void WriteToFile (const char *output_filename) const

  *Write the clustered data to a file.*

- void ReadFromStream (istream &input_stream)

  *Read clustered data from a stream.*

- void ReadFromFile (const char *input_filename)

  *Read the clustered data from a file.*

**Static Public Attributes**

- static const int MAX_ITERATIONS

  *Default 100.*

### D.7.2 Constructor & Destructor Documentation

#### D.7.2.1 HierarchicalClusterer ()

### D.7.3 Member Function Documentation

#### D.7.3.1 const Tree< PointTreeNode > & centers () const

Get the cluster centers.

Must be called after Cluster() or ReadFromStream(). Returns a const ref to the tree of cluster centers. A PointTreeNode simply has a Point in it, which represents the cluster center.

**See also:**

Tree

#### D.7.3.2 int membership (int *index*) const

Report which leaf the <index>th point belongs to.

Returns the node ID of the leaf that this point belongs to. This ID can be used for lookups in the centers() tree.

**See also:**

IdentifyMemberIDPath()
IdentifyMemberTreePath()

#### D.7.3.3 int membership_size () const [inline]

Return the number of points that were clustered.

#### D.7.3.4 void IdentifyMemberIDPath (int *index*, list< int > * *out*) const

Get the set of node IDs in the hierarchy that a point belongs to.

Outputs a list with size equal to the depth of the tree (or if the tree is not balanced, a size equal to the depth of the leaf node associated with a point). The elements of the returned list are the IDs of the nodes (starting with the root, ending at the leaf) this point belongs to at that level. The last element will be the ID of the root, which currently will always be 0.

**See also:**

IdentifyMemberTreePath()

#### D.7.3.5 void IdentifyMemberTreePath (int *index*, list< int > * *out*) const

Get a path down the hierarchy that a point belongs to.

Outputs a list representing a path down the tree. If the leaf is at depth n, then the output of this list has size (n-1). The first element of the list specifies which child (its index) of the root to go down. The second element specifies which child to go to after that, and so on. Obviously, this value is at most B (the branch factor), since each node will only have at most B children. The value tells you the index of the child link to traverse in the tree.

This method is useful for converting to sparse representations, such as Pyramids.

**See also:**

IdentifyMemberIDPath()

#### D.7.3.6 void Cluster (int *num_levels*, int *branch_factor*, const vector< PointRef > & *points*, const DistanceComputer & *distance_computer*)

Performs the actual clustering and stores the result internally.

#### D.7.3.7 void WriteToStream (ostream & *output_stream*) const

Write the clustered data to a stream.

Must be called after Cluster() or ReadFromStream(). File format:

- (int32) N, the number of levels
- (int32) P, the total number of clustered points
- (int32) D, the feature dim
- (Tree<PointTreeNode>) A tree representing the cluster centers
- For each point:
  - (int32) the node ID of the leaf node that the point belongs to.

This function will abort if the stream is bad.

#### D.7.3.8 void WriteToFile (const char * *output_filename*) const

Write the clustered data to a file.

#### D.7.3.9 void ReadFromStream (istream & *input_stream*)

Read clustered data from a stream.

Can be called in lieu of Cluster() to load preprocessed data. See WriteToStream() for the format. This function will abort if the stream is bad.

**See also:**

WriteToStream()

#### D.7.3.10 void ReadFromFile (const char * *input_filename*)

Read the clustered data from a file.

### D.7.4 Member Data Documentation

#### D.7.4.1 const int MAX_ITERATIONS [static]

Default 100.

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

- clustering/hierarchical-clusterer.h
- clustering/hierarchical-clusterer.cc

## D.8 Histogram Class Reference

`#include <histogram.h>`

### D.8.1 Detailed Description

A sparse representation for a flat (1-D) histogram.

Note that this implementation sorts and consolidates bins lazily, meaning that there can be bins with duplicate indices in the internal structure. However, the public interface behaves as though the internal data is always sorted and consolidated, so there is no need to worry about duplicate bins or unsorted bins when accessing elements using any of the public methods.

#### Public Member Functions

- Histogram ()
- ~Histogram ()
- Bin * add_bin (const Bin &new_bin)

  *Adds a copy of <new_bin> to the histogram.*

- void SortBins () const

  *Sorts and consolidates bins.*

- int size () const

  *Gets the total number of bins with unique index.*

- const Bin * bin (int ii)

  *Assuming the Bins are sorted, gets the <ii>th bin in the ordering.*

- const Bin * bin (const LargeIndex &index)

*Gets the bin with the given index.*

- const Bin * bin (const LargeIndex &index, int finger)

  *Gets the Bin with the given index after the <finger>th Bin.*

- void Normalize ()

  *Normalizes the histogram by number of entries.*

### Static Public Member Functions

- static double ComputeIntersection (Histogram *first, Histogram *second)

  *Compute the histogram intersection.*

- static double ComputeChiSquaredDistance (Histogram *first, Histogram *second)

  *Computes the chi-squared distance between two Histograms.*

- static double ComputeSumSquaredDistance (Histogram *first, Histogram *second)
- static void WriteToStream (ostream &output_stream, const vector< Histogram * > &histograms)

  *Write a vector of Histograms to a stream.*

- static void WriteSingleHistogramToStream (ostream &output_stream, Histogram *h)

  *Write a single Histogram to a stream.*

- static vector< Histogram * > ReadFromStream (istream &input_stream)

  *Read a vector of Histograms from a stream.*

- static Histogram * ReadSingleHistogramFromStream (istream &input_stream)

  *Read a single Histogram from a stream.*

## D.8.2 Constructor & Destructor Documentation

### D.8.2.1 Histogram ()

### D.8.2.2 ~Histogram ()

### D.8.3 Member Function Documentation

#### D.8.3.1 Bin * add_bin (const Bin & *new_bin*)

Adds a copy of <new_bin> to the histogram.

By "adding a bin", we mean looking for the Bin in this histogram that has the same index (the LargeIndex identifier) as <new_bin> and increasing the count by that much. It also will set the size of the bin to be the **maximum** of the two bin sizes. The parent, child, and sibling pointers of <new_bin> are completely ignored.

#### D.8.3.2 void SortBins () const

Sorts and consolidates bins.

It is not necessary to call this manually, as all of the sorting and consolidation is done automatically (lazily). However, for performance reasons, you may call this manually.

#### D.8.3.3 int size () const

Gets the total number of bins with unique index.

#### D.8.3.4 const Bin * bin (int *ii*)

Assuming the Bins are sorted, gets the <ii>th bin in the ordering.

The Bins are sorted in order of increasing index.

#### D.8.3.5 const Bin * bin (const LargeIndex & *index*)

Gets the bin with the given index.

This will return NULL if no such Bin is found.

#### D.8.3.6 const Bin * bin (const LargeIndex & *index*, int *finger*)

Gets the Bin with the given index after the <finger>th Bin.

This does a linear search for the Bin with the given index, but begins the search from the <finger>th Bin in the sorted list of all Bins.

#### D.8.3.7 double ComputeIntersection (Histogram * *first*, Histogram * *second*) [static]

Compute the histogram intersection.

#### D.8.3.8 double ComputeChiSquaredDistance (Histogram * *first*, Histogram * *second*) [static]

Computes the chi-squared distance between two Histograms.

#### D.8.3.9 double ComputeSumSquaredDistance (Histogram * *first*, Histogram * *second*) [static]

#### D.8.3.10 void WriteToStream (ostream & *output_stream*, const vector< Histogram * > & *histograms*) [static]

Write a vector of Histograms to a stream.

The output format is as follows:

- (int32) N, the number of histograms
- (N * Histogram) the histograms.

#### D.8.3.11 void WriteSingleHistogramToStream (ostream & *output_stream*, Histogram * *h*) [static]

Write a single Histogram to a stream.

The output format is as follows:

- (int32) B, the number of bins
- For each bin:
  - (int32) S, the size of the LargeIndex identifier
  - (S * int32) the index
  - (double) size
  - (double) count

    The bins are sorted before they are written. They are sorted in order of increasing index.

#### D.8.3.12 vector< Histogram * > ReadFromStream (istream & *input_stream*) [static]

Read a vector of Histograms from a stream.

This function allocates new memory for the histograms. The caller is responsible for freeing it. For the file format, see WriteToStream.

**See also:**

WriteToStream

#### D.8.3.13 Histogram * ReadSingleHistogramFromStream (istream & *input_stream*) [static]

Read a single Histogram from a stream.

This function allocates new memory for the histogram. The caller is responsible for freeing it. For the file format, see ReadFromStream.

**See also:**

ReadFromStream

#### D.8.3.14 void Normalize ()

Normalizes the histogram by number of entries.

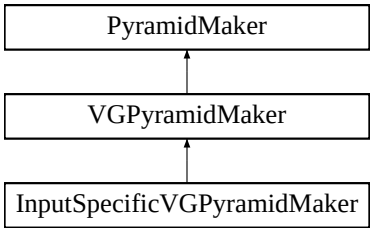
The documentation for this class was generated from the following files:

- histograms/histogram.h
- histograms/histogram.cc

## D.9 InputSpecificVGPyramidMaker Class Reference

`#include <input-specific-vg-pyramid-maker.h>`

Inheritance diagram for InputSpecificVGPyramidMaker::




### D.9.1 Detailed Description

Makes pyramids with bin sizes that are specific to each input.

**Public Member Functions**

- InputSpecificVGPyramidMaker (const HierarchicalClusterer &c, const DistanceComputer &distance_computer)
- virtual MultiResolutionHistogram * MakePyramid (const PointSet &point_set)

  *Turn a single PointSet into a MultiResolutionHistogram.*

- vector< MultiResolutionHistogram * > MakePyramids (const PointSetList &psl)

  *Turn a list of PointSets into a bunch of MultiResolutionHistograms.*

**Protected Member Functions**

- bool GetMembershipPath (const Point &f, LargeIndex *out_path, vector< double > *out_distances)

  *For a new feature, find which bin it would be placed in according to the HierarchicalClusterer.*

**Protected Attributes**

- const HierarchicalClusterer & clusterer_
- const Tree< PointTreeNode > & centers_

  *The centers extracted from clusterer_.*

- const DistanceComputer & distance_computer_

### D.9.2 Constructor & Destructor Documentation

#### D.9.2.1 InputSpecificVGPyramidMaker (const HierarchicalClusterer & *c*, const DistanceComputer & *distance_computer*)

### D.9.3 Member Function Documentation

#### D.9.3.1 MultiResolutionHistogram * MakePyramid (const PointSet & *point_set*) [virtual]

Turn a single PointSet into a MultiResolutionHistogram.

This function allocates memory on its own. It is up to the caller to free it.

Implements VGPyramidMaker.

#### D.9.3.2 bool GetMembershipPath (const Point & *f*, LargeIndex * *out_path*, vector< double > * *out_distances*) [protected, inherited]

For a new feature, find which bin it would be placed in according to the HierarchicalClusterer.

The LargeIndex returned is the same size as the number of levels in the tree. Each element of the LargeIndex tells you which child index to traverse. The first element is 0 by default, since the root of the tree has no parent. For instance, if the returned LargeIndex is [0 3 9], then the path down the tree is: root R, 3rd child of R, followed by 9th child of that node.

The returned vector of doubles gives the distance to the corresponding center at each level, according to distance_computer_. Thus, the first element is the distance from the point <f> to the center of the root node, etc.

The pointers must not be NULL.

Returns true on success, which is always.

Reimplemented in GlobalVGPyramidMaker.

#### D.9.3.3 vector< MultiResolutionHistogram * > MakePyramids (const PointSetList & *psl*) [inherited]

Turn a list of PointSets into a bunch of MultiResolutionHistograms.

It is up to the caller to free the memory.

### D.9.4 Member Data Documentation

#### D.9.4.1 const HierarchicalClusterer& clusterer_ [protected, inherited]

#### D.9.4.2 const Tree<PointTreeNode>& centers_ [protected, inherited]

The centers extracted from clusterer_.

#### D.9.4.3 const DistanceComputer& distance_computer_ [protected, inherited]

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

- pyramids/input-specific-vg-pyramid-maker.h
- pyramids/input-specific-vg-pyramid-maker.cc

## D.10 KernelMatrix Class Reference

`#include <kernel-matrix.h>`

### D.10.1 Detailed Description

Data structure for a square symmetric matrix.

#### Public Member Functions

- KernelMatrix ()

  *Creates an empty (0x0) matrix.*

- KernelMatrix (int size)

  *Creates a (size x size) identity matrix.*

- int size () const

  *Gets the current size of the matrix.*

- void Normalize ()

  *Normalizes by dividing k[r][c] by sqrt(k[r][r] ∗ k[c][c]).*

- void NormalizeByMinCardinality (const vector< int > &cards)

  *Normalize by min cardinality.*

- void Resize (int new_size)

  *Resizes the matrix.*

- double & at (int row, int col)

  *Get a ref to the kernel value at row, col.*

- double at (int row, int col) const

  *Get the kernel value at row, col.*

- void WriteToStream (ostream &output_stream) const

  *Write the matrix to a stream.*

- void WriteToFile (const char ∗filename) const

  *Write the matrix to a stream.*

- void ReadFromStream (istream &input_stream)

  *Replaces all data in this matrix with new data from the stream.*

- void ReadFromFile (const char *filename)

  *Replaces all data in this matrix with new data from the file.*

### D.10.2 Constructor & Destructor Documentation

#### D.10.2.1 KernelMatrix () [inline]

Creates an empty (0x0) matrix.

#### D.10.2.2 KernelMatrix (int *size*)

Creates a (size x size) identity matrix.

### D.10.3 Member Function Documentation

#### D.10.3.1 int size () const

Gets the current size of the matrix.

#### D.10.3.2 void Normalize ()

Normalizes by dividing k[r][c] by sqrt(k[r][r] * k[c][c]).

#### D.10.3.3 void NormalizeByMinCardinality (const vector< int > & *cards*)

Normalize by min cardinality.

Requires cards.size() == size(). cards[i] is the cardinality of the ith set. This function will divide k[r][c] by the minimum of cards[r] and cards[c].

**See also:**

PointSetList::GetSetCardinalities()

#### D.10.3.4 void Resize (int *new_size*)

Resizes the matrix.

Warning: this can cause data loss if you are shrinking the matrix. If rows are added, any new elements on the diagonal are initialized to 1, all other new elements are 0.

#### D.10.3.5 double & at (int *row*, int *col*)

Get a ref to the kernel value at row, col.

This allows you to do things like kernel.at(1, 3) = 37; KernelMatrix will automatically maintain symmetry, so that after you do this, if you call kernel.at(3, 1), it will be 37.

#### D.10.3.6 double at (int *row*, int *col*) const

Get the kernel value at row, col.

#### D.10.3.7 void WriteToStream (ostream & *output_stream*) const

Write the matrix to a stream.

File format:

- (int32) N, the number of rows (and cols)
- (1 ∗ double) row 0 (just k[0][0])
- (2 ∗ double) row 1
- ...
- (N ∗ double) row N-1

Aborts if the stream is bad.

#### D.10.3.8 void WriteToFile (const char ∗ *filename*) const

Write the matrix to a stream.

**See also:**

WriteToStream

#### D.10.3.9 void ReadFromStream (istream & *input_stream*)

Replaces all data in this matrix with new data from the stream.

Aborts if the stream is bad. See WriteToStream() for the file format.

**See also:**

WriteToStream()

#### D.10.3.10 void ReadFromFile (const char * *filename*)

Replaces all data in this matrix with new data from the file.

**See also:**

ReadFromStream

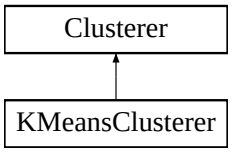
The documentation for this class was generated from the following files:

- kernel/kernel-matrix.h
- kernel/kernel-matrix.cc

## D.11 KMeansClusterer Class Reference

`#include <k-means-clusterer.h>`

Inheritance diagram for KMeansClusterer::




### D.11.1 Detailed Description

Implements K-Means clustering.

This implementation may not always return K clusters. There are two cases where we will return fewer than K clusters:

1. If the number of points provided (N) is less than K, then KMeansClusterer will return N clusters, where each cluster center is one of the points.
2. If the data contains duplicate points, and there are fewer than K unique points, then KMeans-Clusterer will return M points, where M is the number of unique points in the data.

Both of these situations are generally unlikely, but you should be careful about the assumptions your code makes about the number of returned clusters.

### Public Member Functions

- KMeansClusterer (int num_clusters, int max_iters, const DistanceComputer &distance_computer)
- void Cluster (const vector< PointRef > &data)

  *Performs the clustering and stores the result internally.*

- void Cluster (const vector< const Point * > &data)

  *Performs the clustering and stores the result internally.*

- const PointSet & centers () const

  *Output the cluster centers.*

- int centers_size () const

  *Return the number of cluster centers.*

- int membership (int index) const

  *Return the membership of the <index>th point.*

- int membership_size () const

  *Return the number of members. Equivalent to the number of points that were clustered.*

- void WriteToStream (ostream &output_stream) const

  *Write the clustering data to a stream.*

- void WriteToFile (const char *output_filename) const

  *Write the clustering data to a file.*

- void ReadFromStream (istream &input_stream)

  *Read clustering data from a stream.*

- void ReadFromFile (const char *input_filename)

  *Read clustering data from a file.*

### Protected Member Functions

- virtual void DoClustering (const vector< const Point * > &data)

  *Perform K-means.*

**Protected Attributes**

- auto_ptr< PointSet > cluster_centers_
- vector< int > membership_
- bool done_

### D.11.2 Constructor & Destructor Documentation

#### D.11.2.1 KMeansClusterer (int *num_clusters*, int *max_iters*, const DistanceComputer & *distance_computer*)

### D.11.3 Member Function Documentation

#### D.11.3.1 void DoClustering (const vector< const Point * > & *data*) [protected, virtual]

Perform K-means.

Uses the DistanceComputer it was constructed with to fill up cluster_centers_ with K Point representing the K-means cluster centers. K is assigned by the constructor of KMeansClusterer. If there are fewer data points than K, then the total number of clusters returned is simply the total number of data points (not K).

Implements Clusterer.

#### D.11.3.2 void Cluster (const vector< PointRef > & *data*) [inherited]

Performs the clustering and stores the result internally.

To avoid potential memory problems, Clusterers do not operate on PointSetLists or PointSets directly. Rather, they simply shuffle PointRefs around.

**See also:**

PointSetList::GetPointRefs()

#### D.11.3.3 void Cluster (const vector< const Point * > & *data*) [inherited]

Performs the clustering and stores the result internally.

To avoid potential memory problems, Clusterers do not operate on PointSetLists or PointSets directly. Rather, they simply shuffle pointers around.

#### D.11.3.4 const PointSet & centers () const [inherited]

Output the cluster centers.

This requires Cluster() or ReadFromStream() to have been called first. It returns the set of all cluster centers as Points.

#### D.11.3.5 int centers_size () const [inherited]

Return the number of cluster centers.

This requires Cluster() or ReadFromStream() to have been called first. It reutnrs the number of cluster centers.

#### D.11.3.6 int membership (int *index*) const [inherited]

Return the membership of the <index>th point.

The return value gives the ID of the cluster that this point belongs to. "ID" in this sense means an index into the PointSet returned by centers().

#### D.11.3.7 int membership_size () const [inherited]

Return the number of members. Equivalent to the number of points that were clustered.

#### D.11.3.8 void WriteToStream (ostream & *output_stream*) const [inherited]

Write the clustering data to a stream.

Requires Cluster() or ReadFromStream() to have been called first. Output format:

- (int32) C, the number of cluster centers
- (int32) P, the total number of clustered points
- (int32) D, the dimension of the Points
- (Point) center 0
- (Point) center 1
- (Point) ...
- (Point) center C-1
- (int32) the cluster to which point 1 belongs
- (int32) the cluster to which point 2 belongs

- (int32) ...
- (int32) the cluster to which point P belongs

  The clustered points themselves are not written to the stream, only the centers and membership data. It is assumed that the caller of Cluster() already has access to those points anyway. This function aborts if the stream is bad.

#### D.11.3.9 void WriteToFile (const char * *output_filename*) const [inherited]

Write the clustering data to a file.

#### D.11.3.10 void ReadFromStream (istream & *input_stream*) [inherited]

Read clustering data from a stream.

Can be called in lieu of Cluster(). If this is called after Cluster(), all of the previous data is cleared before reading the new data. For the file format, see WriteToStream. This function aborts if the stream is bad.

**See also:**

WriteToStream.

#### D.11.3.11 void ReadFromFile (const char * *input_filename*) [inherited]

Read clustering data from a file.

### D.11.4 Member Data Documentation

#### D.11.4.1 auto_ptr<PointSet> cluster_centers_ [protected, inherited]

#### D.11.4.2 vector<int> membership_ [protected, inherited]

#### D.11.4.3 bool done_ [protected, inherited]

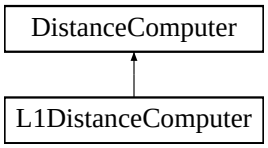
The documentation for this class was generated from the following files:

- clustering/k-means-clusterer.h
- clustering/k-means-clusterer.cc

## D.12 L1DistanceComputer Class Reference

`#include <distance-computer.h>`

Inheritance diagram for L1DistanceComputer::




### D.12.1 Detailed Description

Computes L1 distance between two Points.

### Public Member Functions

- virtual double ComputeDistance (const Point &f1, const Point &f2) const

  *Compute distance.*

- virtual double ComputeDistance (const Point &f1, const Point &f2, double max_distance) const

  *Quickly compute min(actual_distance(f1, f2), max_distance).*

### D.12.2 Member Function Documentation

#### D.12.2.1 double ComputeDistance (const Point & *f1*, const Point & *f2*) const `[virtual]`

Compute distance.

Implements DistanceComputer.

#### D.12.2.2 double ComputeDistance (const Point & *f1*, const Point & *f2*, double *max_distance*) const `[virtual]`

Quickly compute min(actual_distance(f1, f2), max_distance).

This is the same as ComputeDistance, except it will stop calculating things when it knows the distance will be greater than max_distance.

Reimplemented from DistanceComputer.

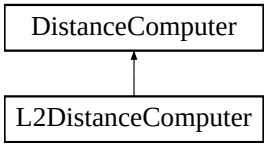
The documentation for this class was generated from the following files:

- util/distance-computer.h
- util/distance-computer.cc

## D.13 L2DistanceComputer Class Reference

`#include <distance-computer.h>`

Inheritance diagram for L2DistanceComputer::




### D.13.1 Detailed Description

Computes the **square** of the L2 distance between two Points.

#### Public Member Functions

- virtual double ComputeDistance (const Point &f1, const Point &f2) const

  *Compute distance.*

- virtual double ComputeDistance (const Point &f1, const Point &f2, double max_distance) const

  *Quickly compute min(actual_distance(f1, f2), max_distance).*

### D.13.2 Member Function Documentation

#### D.13.2.1 double ComputeDistance (const Point & *f1*, const Point & *f2*) const [virtual]

Compute distance.

Implements DistanceComputer.

#### D.13.2.2 double ComputeDistance (const Point &*f1*, const Point &*f2*, double *max_distance*) const [virtual]

Quickly compute min(actual_distance(f1, f2), max_distance).

This is the same as ComputeDistance, except it will stop calculating things when it knows the distance will be greater than max_distance.

Reimplemented from DistanceComputer.

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

- util/distance-computer.h
- util/distance-computer.cc

## D.14 LabeledIndex Class Reference

`#include <labeled-index.h>`

### D.14.1 Detailed Description

An index-label pair.

This is a pair of two integers: the *index*, which will usually refer to an image, and a *label* which refers to its class.

#### Public Member Functions

- bool operator< (const LabeledIndex &other) const

  *Compares the index only.*

- bool operator== (const LabeledIndex &other) const

  *Checks both the label and the index.*

#### Public Attributes

- int label
- int index

### D.14.2 Member Function Documentation

#### D.14.2.1 bool operator< (const LabeledIndex & *other*) const

Compares the index only.

#### D.14.2.2 bool operator== (const LabeledIndex & *other*) const

Checks both the label and the index.

### D.14.3 Member Data Documentation

#### D.14.3.1 int label

#### D.14.3.2 int index

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

- util/labeled-index.h
- util/labeled-index.cc

## D.15 Matrix Class Reference

`#include <matrix.h>`

### D.15.1 Detailed Description

Data structure for a matrix of doubles.

#### Public Member Functions

- Matrix ()

  *Creates an empty (0x0) matrix.*

- Matrix (int rows, int cols)

  *Creates a (rows x cols) matrix full of 0s.*

- int num_rows () const

  *Gets the number of rows.*

- int num_cols () const

  *Gets the number of columns.*

- void Resize (int new_rows, int new_cols)

  *Resizes the matrix.*

- double & at (int row, int col)

  *Get a ref to the kernel value at row, col.*

- double at (int row, int col) const

  *Get the kernel value at row, col.*

- void WriteToStream (ostream &output_stream) const

  *Write the matrix to a stream.*

- void WriteToFile (const char *filename) const

  *Write the matrix to a file.*

- void ReadFromStream (istream &input_stream)

  *Replaces all data in this matrix with new data from the stream.*

- void ReadFromFile (const char *filename)

  *Replaces all data in this matrix with new data from the file.*

### D.15.2 Constructor & Destructor Documentation

#### D.15.2.1 Matrix () `[inline]`

Creates an empty (0x0) matrix.

#### D.15.2.2 Matrix (int *rows*, int *cols*)

Creates a (rows x cols) matrix full of 0s.

### D.15.3 Member Function Documentation

#### D.15.3.1 int num_rows () const

Gets the number of rows.

#### D.15.3.2 int num_cols () const

Gets the number of columns.

#### D.15.3.3 void Resize (int *new_rows*, int *new_cols*)

Resizes the matrix.

Warning: this can cause data loss if you are shrinking the matrix. If rows or columns are added, any new elements are 0.

#### D.15.3.4 double & at (int *row*, int *col*)

Get a ref to the kernel value at row, col.

#### D.15.3.5 double at (int *row*, int *col*) const

Get the kernel value at row, col.

#### D.15.3.6 void WriteToStream (ostream & *output_stream*) const

Write the matrix to a stream.

File format:

- (int32) R, the number of rows
- (int32) C, the number of cols
- (double * C) k[0][0], k[0][1], ...
- ...
- (double * C) k[R-1][0], k[R-1][1], ...

Aborts if the stream is bad.

#### D.15.3.7 void WriteToFile (const char * *filename*) const

Write the matrix to a file.

**See also:**

WriteToStream

#### D.15.3.8 void ReadFromStream (istream & *input_stream*)

Replaces all data in this matrix with new data from the stream.

Aborts if the stream is bad. See WriteToStream() for the file format.

**See also:**

WriteToStream()

#### D.15.3.9 void ReadFromFile (const char * *filename*)

Replaces all data in this matrix with new data from the file.

**See also:**

ReadFromStream

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

- kernel/matrix.h
- kernel/matrix.cc

## D.16 MultiResolutionHistogram Class Reference

`#include <multi-resolution-histogram.h>`

### D.16.1 Detailed Description

A data structure for a pyramid of histograms, with a link structure between levels.

A wrapper class around SparseTree. This data structure encapsulates a SparseTree of Bin objects, where a Bin just contains a weight (size) and a count.

### Public Member Functions

- MultiResolutionHistogram ()
- int size () const
- double total_counts () const
- Bin * bin (const LargeIndex &index)

  *Get a pointer to the bin with the specified index.*

- Bin * bin (const LargeIndex &index, Bin *finger)

  *Get a pointer to the bin with specified index.*

- Bin * root ()

  *Get a pointer to the root bin.*

- Bin *const root () const
- Bin * add_bin (const Bin &new_bin)

  *Insert a copy of the given bin into the tree.*

- Bin * add_bin (const Bin &new_bin, Bin *finger)

  *Insert a copy of the given bin into the tree.*

- void remove_bin (Bin *finger)

  *Remove a bin, and all of its children, from the pyramid.*

- void Normalize ()

  *Normalizes the histogram in-place so that total_count() == 1.*

- void Normalize (double divisor)

  *Normalizes the histogram in-place by dividing every count by the given number.*

### Static Public Member Functions

- static vector< MultiResolutionHistogram * > ReadFromStream (istream &input_stream)

  *Reads all the MultiResolutionHistograms in a stream.*

- static int ReadHeaderFromStream (istream &input_stream)

  *Reads just the header from a MultiResolutionHistogram file.*

- static vector< MultiResolutionHistogram * > ReadFromFile (const char *filename)

  *Reads all the MultiResolutionHistograms in a file.*

- static vector< MultiResolutionHistogram * > ReadSelectionFromStream (istream &input_stream, int start, int selection_size)

  *Reads some MultiResolutionHistograms from a stream.*

- static vector< MultiResolutionHistogram * > ReadSelectionFromFile (const char *filename, int start, int selection_size)

  *Reads some MultiResolutionHistograms from a file.*

- static void WriteToStream (ostream &output_stream, const vector< MultiResolution-Histogram * > &hists)

  *Writes all of the MultiResolutionHistograms to a stream.*

- static void WriteHeaderToStream (ostream &output_stream, int num_hists)

  *Writes just a header for a MultiResolutionHistogram file to stream.*

- static void WriteToFile (const char *filename, const vector< MultiResolutionHistogram * > &hists)

  *Writes all of the MultiResolutionHistograms to a file.*

- static MultiResolutionHistogram * ReadSingleHistogramFromStream (istream &input_-stream)

  *Reads just one histogram from a stream.*

- static void IgnoreSingleHistogramFromStream (istream &input_stream)

  *Throws out one histogram's worth of data from the stream.*

- static void WriteSingleHistogramToStream (ostream &output_stream, const Multi-ResolutionHistogram *h)

  *Write just one histogram to a stream.*

### D.16.2 Constructor & Destructor Documentation

#### D.16.2.1 MultiResolutionHistogram () `[inline]`

### D.16.3 Member Function Documentation

#### D.16.3.1 int size () const `[inline]`

#### D.16.3.2 double total_counts () const

#### D.16.3.3 Bin * bin (const LargeIndex & *index*)

Get a pointer to the bin with the specified index.

Returns NULL if the bin is not found.

#### D.16.3.4 Bin * bin (const LargeIndex & *index*, Bin * *finger*)

Get a pointer to the bin with specified index.

Same as GetBin(LargeIndex), but localizes the search to the subtree given by finger. Returns NULL if there is no such Bin in a subtree of <finger>.

#### D.16.3.5 Bin* root () [inline]

Get a pointer to the root bin.

#### D.16.3.6 Bin* const root () const [inline]

#### D.16.3.7 Bin * add_bin (const Bin & *new_bin*)

Insert a copy of the given bin into the tree.

Returns a pointer to the newly-added bin in the tree. This function completely ignores any parent/child/sibling pointers in <new_bin>.

This function requires the parent bin to exist. It will not create new bins along the way (it will abort if there is no parent bin, i.e., a bin whose index is a prefix of that of <new_bin> and whose index size is exactly 1 less than the new bin's index size. The insertion happens such that the sibling relationships remain sorted by index.

If there is already is a bin with the given index, we add the counts and keep the larger of the two sizes. This applies to the root bin as well.

#### D.16.3.8 Bin * add_bin (const Bin & *new_bin*, Bin * *finger*)

Insert a copy of the given bin into the tree.

Same as add_bin(const Bin&), except it starts the search for the bin at <finger>. A parent to <new_bin> must already exist, and must be present in a sub-branch of <finger> (it may also be <finger>).

#### D.16.3.9 void remove_bin (Bin * *finger*)

Remove a bin, and all of its children, from the pyramid.

#### D.16.3.10 void Normalize ()

Normalizes the histogram in-place so that total_count() == 1.

#### D.16.3.11 void Normalize (double *divisor*)

Normalizes the histogram in-place by dividing every count by the given number.

#### D.16.3.12 vector< MultiResolutionHistogram * > ReadFromStream (istream & *input_stream*) [static]

Reads all the MultiResolutionHistograms in a stream.

File format:

- (int32) N, the number of MultiResolutionHistograms.
- (N * MultiResolutionHistogram) The histograms.

Aborts if input_stream cannot be read.

#### D.16.3.13 int ReadHeaderFromStream (istream & *input_stream*) [static]

Reads just the header from a MultiResolutionHistogram file.

The header just contains one integer, the number of histograms in the rest of the stream. So this method will return the value of that integer.

#### D.16.3.14 vector< MultiResolutionHistogram * > ReadFromFile (const char * *filename*) [static]

Reads all the MultiResolutionHistograms in a file.

**See also:**

ReadFromStream

#### D.16.3.15 vector< MultiResolutionHistogram * > ReadSelectionFromStream (istream & *input_stream*, int *start*, int *selection_size*) [static]

Reads some MultiResolutionHistograms from a stream.

<start> specifies the index of the first histogram to read in the stream of data. If <selection_size> is large enough so that this would want to read past the end of the file, ReadSelectionFromStream will just stop reading. Observe that you will need to adjust indices (i.e., if you choose to read Y histograms starting from index X, then result[0] will be the 6th histogram, as opposed to this[5]).

Aborts if the input_stream cannot be read.

#### D.16.3.16 vector< MultiResolutionHistogram * > ReadSelectionFromFile (const char * *filename*, int *start*, int *selection_size*) [static]

Reads some MultiResolutionHistograms from a file.

**See also:**

ReadSelectionFromStream

#### D.16.3.17 void WriteToStream (ostream & *output_stream*, const vector< MultiResolutionHistogram * > & *hists*) [static]

Writes all of the MultiResolutionHistograms to a stream.

Aborts if the stream is bad. See ReadFromStream for the file format.

**See also:**

ReadFromStream

#### D.16.3.18 void WriteHeaderToStream (ostream & *output_stream*, int *num_hists*) [static]

Writes just a header for a MultiResolutionHistogram file to stream.

**See also:**

ReadHeaderFromStream

#### D.16.3.19 void WriteToFile (const char * *filename*, const vector< MultiResolutionHistogram * > & *hists*) [static]

Writes all of the MultiResolutionHistograms to a file.

**See also:**

WriteToFile

#### D.16.3.20 MultiResolutionHistogram * ReadSingleHistogramFromStream (istream & *input_stream*) [static]

Reads just one histogram from a stream.

File format:

- (int32) The total number of bins, including the root
- For each bin:
  - (int32) L, the size of its index (0 for the root, etc.)
  - (L ∗ int32) the index
  - (double) size
  - (double) count
    The ordering of the bins is a depth-first traversal of the tree.
    Aborts if the stream is bad.

#### D.16.3.21 void IgnoreSingleHistogramFromStream (istream & *input_stream*) [static]

Throws out one histogram's worth of data from the stream.

#### D.16.3.22 void WriteSingleHistogramToStream (ostream & *output_stream*, const MultiResolutionHistogram ∗ *h*) [static]

Write just one histogram to a stream.

Aborts if the stream is bad. See ReadSingleHistogramFromStream for the format.

**See also:**

ReadSingleHistogramFromStream

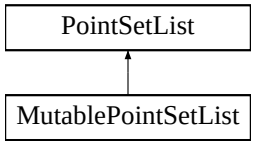
The documentation for this class was generated from the following files:

- histograms/multi-resolution-histogram.h
- histograms/multi-resolution-histogram.cc

## D.17 MutablePointSetList Class Reference

`#include <mutable-point-set-list.h>`

Inheritance diagram for MutablePointSetList::



### D.17.1 Detailed Description

A mutable, in-memory PointSetList.

This class adds AddPointSet to the interface, allowing one to (as the name suggests) add new PointSets. You may also modify PointSets or the Points they contain, and can replace PointSets in the list with other PointSets.

#### Public Member Functions

- virtual ~MutablePointSetList ()
- void ReadFromStream (istream &input_stream)

  *Reads an entire data set from a stream.*

- void ReadFromFile (const char *filename)

  *Reads an entire data set from a file.*

- void ReadSelectionFromStream (istream &input_stream, int start, int selection_size)

  *Reads a segment of the data from a stream.*

- virtual int point_set_size () const

  *Get the total number of PointSets in this PointSetList.*

- virtual int point_size () const

  *Get the total number of Points in this PointSetList.*

- virtual int point_dim () const

  *Get the dim of every Point in this PointSetList.*

- const PointSet & point_set (int index) const

  *Returns a const ref to the <index>th PointSet.*

- PointSet * mutable_point_set (int index)

  *Returns a pointer to the <index>th PointSet.*

- PointSet * add_point_set (const PointSet &point_set)

  *Adds a copy of <point_set> to the end of the list.*

- PointSet * add_point_set (int point_dim)

*Adds a new, empty PointSet to the end of the list.*

- virtual bool GetSetCardinalities (vector< int > *destination) const

  *Get the number of Points in each PointSet.*

- const PointSet & operator[ ] (int index)

  *Returns a const ref to the <index>th PointSet.*

- const Point & point (int index) const

  *Returns the <index>th Point in the PointSetList.*

- bool GetPointIndices (int index, int *which_point_set, int *which_point) const

  *Locate a Point in a PointSet.*

- void GetPointRefs (vector< PointRef > *out_refs) const

  *Get PointRefs to every Point in a particular image.*

- void GetPointRefs (int index, vector< PointRef > *out_refs) const

  *Get PointRefs to every Point in this PointSetList.*

- void WriteToStream (ostream &output_stream) const

  *Writes the entire PointSetList to a stream.*

- void WriteHeaderToStream (ostream &output_stream) const

  *Writes just the serialized header to a stream.*

- void WritePointSetsToStream (ostream &output_stream) const

  *Writes the point sets (without a header) sequentially to a stream.*

- void WriteToFile (const char *output_file) const

  *Writes the entire PointSetList to a file.*

### D.17.2 Constructor & Destructor Documentation

#### D.17.2.1 ~MutablePointSetList () [virtual]

### D.17.3 Member Function Documentation

#### D.17.3.1 void ReadFromStream (istream & *input_stream*)

Reads an entire data set from a stream.

Aborts if the input_stream cannot be read.

#### D.17.3.2 void ReadFromFile (const char * *filename*)

Reads an entire data set from a file.

Aborts if the file cannot be read.

#### D.17.3.3 void ReadSelectionFromStream (istream & *input_stream*, int *start*, int *selection_size*)

Reads a segment of the data from a stream.

<start> specifies the index of the first PointSet to read in the stream of data. If <selection_size> is large enough so that this would want to read past the end of the file, ReadSelectionFromStream will just stop reading. Observe that you will need to adjust indices (i.e., if you choose to read Y PointSets starting from index X, then this[0] will be the 6th PointSet, as opposed to this[5]).

Aborts if the input_stream cannot be read.

#### D.17.3.4 int point_set_size () const [virtual]

Get the total number of PointSets in this PointSetList.

Implements PointSetList.

#### D.17.3.5 int point_size () const [virtual]

Get the total number of Points in this PointSetList.

This is the sum of PointSet::size() over all PointSets in in this PointSetList.

Implements PointSetList.

#### D.17.3.6 int point_dim () const [virtual]

Get the dim of every Point in this PointSetList.

Defined to be 0 if the PointSetList is empty.

Implements PointSetList.

#### D.17.3.7 const PointSet & point_set (int *index*) const [virtual]

Returns a const ref to the <index>th PointSet.

Implements PointSetList.

#### D.17.3.8 PointSet * mutable_point_set (int *index*)

Returns a pointer to the <index>th PointSet.

#### D.17.3.9 PointSet * add_point_set (const PointSet & *point_set*)

Adds a copy of <point_set> to the end of the list.

Returns a pointer to the newly added point set.

#### D.17.3.10 PointSet * add_point_set (int *point_dim*)

Adds a new, empty PointSet to the end of the list.

Returns a pointer to the newly added point set. If this PointSetList already has data in it, it will check to make sure that <point_dim> matches the dim of those points. If the current PointSetList is empty, it will add a new PointSet initialized with <point_dim> as the point dim.

#### D.17.3.11 bool GetSetCardinalities (vector< int > * *destination*) const [virtual, inherited]

Get the number of Points in each PointSet.

Fills <destination> with a vector of size this.point_set_size(), where the nth element is the number of Points in the nth PointSet in this PointSetList. It is cleared first, and must not be NULL.

Returns true on success and false otherwise, although currently there is no reason that this method will fail.

Reimplemented in OnDiskPointSetList.

#### D.17.3.12 ] const PointSet& operator[ ] (int *index*) [inline, inherited]

Returns a const ref to the <index>th PointSet.

#### D.17.3.13 const Point & point (int *index*) const [inherited]

Returns the <index>th Point in the PointSetList.

We can also think of a PointSetList as a long vector of Points if we ignore the PointSet boundaries, so you can reference individual points of a PointSetList through this.

#### D.17.3.14 bool GetPointIndices (int *index*, int * *which_point_set*, int * *which_point*) const [inherited]

Locate a Point in a PointSet.

Given an index specifying which point we are referring to, report which PointSet it belongs to, as well as the index into that PointSet that the PointSet belongs to.

Returns true if <index> is a valid Point and the data was successfully loaded, false otherwise.

#### D.17.3.15 void GetPointRefs (vector< PointRef > * *out_refs*) const [inherited]

Get PointRefs to every Point in a particular image.

Requires out_refs != NULL. Makes out_refs a vector of size this->point_size(), where the nth element is a PointRef pointing to the nth Point (i.e., points to this->point(n)). If out_refs has something in it beforehand, it will be cleared prior to filling it.

**See also:**

PointRef

#### D.17.3.16 void GetPointRefs (int *index*, vector< PointRef > * *out_refs*) const [inherited]

Get PointRefs to every Point in this PointSetList.

Requires out_refs != NULL. Makes out_refs a vector of size this->point_set(index).size(), where the nth element is a PointRef pointing to the nth Point in the <index>th image. If out_refs has something in it beforehand, it will be cleared prior to filling it.

**See also:**

PointRef

#### D.17.3.17 void WriteToStream (ostream & *output_stream*) const [inherited]

Writes the entire PointSetList to a stream.

See the detailed description of PointSetList for the format.

#### D.17.3.18 void WriteHeaderToStream (ostream & *output_stream*) const [inherited]

Writes just the serialized header to a stream.

#### D.17.3.19 void WritePointSetsToStream (ostream & *output_stream*) const [inherited]

Writes the point sets (without a header) sequentially to a stream.

#### D.17.3.20 void WriteToFile (const char * *output_file*) const [inherited]

Writes the entire PointSetList to a file.

See detailed description of PointSetList for the format.

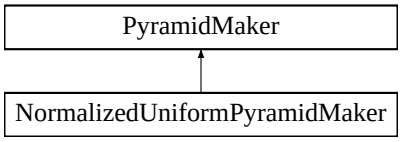
The documentation for this class was generated from the following files:

- point_set/mutable-point-set-list.h
- point_set/mutable-point-set-list.cc

## D.18 NormalizedUniformPyramidMaker Class Reference

`#include <normalized-uniform-pyramid-maker.h>`

Inheritance diagram for NormalizedUniformPyramidMaker::




### D.18.1 Detailed Description

Creates MultiResolutionHistograms with bins of uniform size at each level.

The difference between this class and UniformPyramidMaker is that the image dimensions are normalized– that is, regardless of the minimum and maximum values of a particular feature value, the feature space is always partitioned into a particular number of pieces. Each dimension is always halved at each level in the pyramid.

The bin sizes are normalized too– that is, the top-level bin has size 10000, and bins under that have bin sizes scaled appropriately (i.e., if it is n levels down, the size is (1 / 2^d) * (1 / 2^n) where d is the dimension of the points).

### Public Member Functions

- NormalizedUniformPyramidMaker (int num_levels)

- int num_levels () const
- virtual MultiResolutionHistogram * MakePyramid (const PointSet &ps)

  *Creates a normalized uniform pyramid from a PointSet.*

- virtual MultiResolutionHistogram * MakePyramid (const vector< PointRef > &points)
- void ReadFromStream (istream &input_stream)

  *Read parameters and preprocessed data from a stream.*

- void ReadFromFile (const char *filename)

  *Read parameters and preprocessed data from a file.*

- void WriteToStream (ostream &output_stream) const

  *Write parameters and preprocessed data to a stream.*

- void WriteToFile (const char *filename) const

  *Write parameters and preprocessed data to a file.*

- vector< MultiResolutionHistogram * > MakePyramids (const PointSetList &psl)

  *Turn a list of PointSets into a bunch of MultiResolutionHistograms.*

**Static Public Attributes**

- static const double TOP_LEVEL_BIN_SIZE = 10000

### D.18.2 Constructor & Destructor Documentation

#### D.18.2.1 NormalizedUniformPyramidMaker (int *num_levels*)

### D.18.3 Member Function Documentation

#### D.18.3.1 int num_levels () const

Must be called after Preprocess() or ReadFromStream().

#### D.18.3.2 MultiResolutionHistogram * MakePyramid (const PointSet & *ps*) [virtual]

Creates a normalized uniform pyramid from a PointSet.

This function allocates memory on its own; it is up to the caller to free the memory.

Implements PyramidMaker.

#### D.18.3.3 MultiResolutionHistogram * MakePyramid (const vector< PointRef > & *points*) [virtual]

#### D.18.3.4 void ReadFromStream (istream & *input_stream*)

Read parameters and preprocessed data from a stream.

Aborts if the stream is bad.

#### D.18.3.5 void ReadFromFile (const char * *filename*)

Read parameters and preprocessed data from a file.

**See also:**

ReadFromStream

#### D.18.3.6 void WriteToStream (ostream & *output_stream*) const

Write parameters and preprocessed data to a stream.

Aborts if the stream is bad.

#### D.18.3.7 void WriteToFile (const char * *filename*) const

Write parameters and preprocessed data to a file.

**See also:**

WriteToStream

#### D.18.3.8 vector< MultiResolutionHistogram * > MakePyramids (const PointSetList & *psl*) [inherited]

Turn a list of PointSets into a bunch of MultiResolutionHistograms.

It is up to the caller to free the memory.

### D.18.4 Member Data Documentation

#### D.18.4.1 const double TOP_LEVEL_BIN_SIZE = 10000 [static]

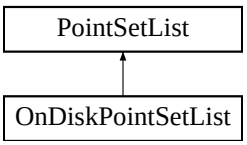
The documentation for this class was generated from the following files:

- pyramids/normalized-uniform-pyramid-maker.h
- pyramids/normalized-uniform-pyramid-maker.cc

## D.19 OnDiskPointSetList Class Reference

`#include <on-disk-point-set-list.h>`

Inheritance diagram for OnDiskPointSetList::




### D.19.1 Detailed Description

A read-only PointSetList which reads information from disk.

This class is suitable for loading data that is too large to fit in memory. For performance, OnDiskPointSetList operates using two caches: an LRU cache (of sorts) which remembers the most recently used PointSets, and what we call an "area" cache, which caches a contiguous block of PointSets. In general, if it is known that a particular application access data in a **sequential** manner, it is generally more useful to make the LRU cache small (size 1) and the area cache large. If the application uses mostly **random access**, it is better to have a large LRU cache and a very small area cache.

#### Public Member Functions

- OnDiskPointSetList (string filename)

  *Load from a particular file, using default cache sizes.*

- OnDiskPointSetList (string filename, int lru_cache_size, int area_cache_size)

  *Load from a particular file, with specified cache sizes.*

- virtual ~OnDiskPointSetList ()

- virtual int point_set_size () const

  *Get the total number of PointSets in this PointSetList.*

- virtual int point_size () const

  *Get the total number of Points in this PointSetList.*

- virtual int point_dim () const

  *Get the dim of every Point in this PointSetList.*

- virtual bool GetSetCardinalities (vector< int > *destination) const

  *Get the number of Points in each PointSet.*

- virtual const PointSet & point_set (int index) const

  *Return a reference to the <index>th PointSet.* **READ WARNING BELOW!**.

- const PointSet & operator[ ] (int index)

  *Returns a const ref to the <index>th PointSet.*

- const Point & point (int index) const

  *Returns the <index>th Point in the PointSetList.*

- bool GetPointIndices (int index, int *which_point_set, int *which_point) const

  *Locate a Point in a PointSet.*

- void GetPointRefs (vector< PointRef > *out_refs) const

  *Get PointRefs to every Point in a particular image.*

- void GetPointRefs (int index, vector< PointRef > *out_refs) const

  *Get PointRefs to every Point in this PointSetList.*

- void WriteToStream (ostream &output_stream) const

  *Writes the entire PointSetList to a stream.*

- void WriteHeaderToStream (ostream &output_stream) const

  *Writes just the serialized header to a stream.*

- void WritePointSetsToStream (ostream &output_stream) const

  *Writes the point sets (without a header) sequentially to a stream.*

- void WriteToFile (const char *output_file) const

  *Writes the entire PointSetList to a file.*

**Static Public Attributes**

- static const int DEFAULT_LRU_CACHE_SIZE

  *Default value: 1500.*

- static const int DEFAULT_AREA_CACHE_SIZE

  *Default value: 1500.*

### D.19.2 Constructor & Destructor Documentation

#### D.19.2.1 OnDiskPointSetList (string *filename*)

Load from a particular file, using default cache sizes.

Makes one pass through the entire data set for preprocessing. Aborts if <filename> is not valid or not readable.

#### D.19.2.2 OnDiskPointSetList (string *filename*, int *lru_cache_size*, int *area_cache_size*)

Load from a particular file, with specified cache sizes.

Makes one pass through the entire data set for preprocessing. Aborts if <filename> is not valid or not readable.

#### D.19.2.3 ~OnDiskPointSetList () `[virtual]`

### D.19.3 Member Function Documentation

#### D.19.3.1 int point_set_size () const `[virtual]`

Get the total number of PointSets in this PointSetList.

Implements PointSetList.

#### D.19.3.2 int point_size () const [virtual]

Get the total number of Points in this PointSetList.

This is the sum of PointSet::size() over all PointSets in in this PointSetList.

Implements PointSetList.

#### D.19.3.3 int point_dim () const [virtual]

Get the dim of every Point in this PointSetList.

Defined to be 0 if the PointSetList is empty.

Implements PointSetList.

#### D.19.3.4 bool GetSetCardinalities (vector< int > * *destination*) const [virtual]

Get the number of Points in each PointSet.

Fills <destination> with a vector of size this.point_set_size(), where the nth element is the number of Points in the nth PointSet in this PointSetList. It is cleared first, and must not be NULL.

Returns true on success and false otherwise, although currently there is no reason that this method will fail.

Reimplemented from PointSetList.

#### D.19.3.5 const PointSet & point_set (int *index*) const [virtual]

Return a reference to the <index>th PointSet. **READ WARNING BELOW!**.

Let A be the size of the area cache, and L be the size of the LRU cache. First, check to see if it's in the area cache, and return it if it's there(O(1)). If it's not there, we check the LRU cache (O(L)). If it's there, move it to the front of the LRU cache and return it. Otherwise, we insert it into the LRU cache, and also fill up the area cache starting at the element we've just retrieved (O(A)).

**Warning**: The reference returned by this operator is no longer valid the next time operator[ ] is called. This means that whenever you want to access more than one PointSet at a time, you must make a copy of them!

Implements PointSetList.

#### D.19.3.6 ] const PointSet& operator[ ] (int *index*) [inline, inherited]

Returns a const ref to the <index>th PointSet.

#### D.19.3.7 const Point & point (int *index*) const [inherited]

Returns the <index>th Point in the PointSetList.

We can also think of a PointSetList as a long vector of Points if we ignore the PointSet boundaries, so you can reference individual points of a PointSetList through this.

#### D.19.3.8 bool GetPointIndices (int *index*, int * *which_point_set*, int * *which_point*) const [inherited]

Locate a Point in a PointSet.

Given an index specifying which point we are referring to, report which PointSet it belongs to, as well as the index into that PointSet that the PointSet belongs to.

Returns true if <index> is a valid Point and the data was successfully loaded, false otherwise.

#### D.19.3.9 void GetPointRefs (vector< PointRef > * *out_refs*) const [inherited]

Get PointRefs to every Point in a particular image.

Requires out_refs != NULL. Makes out_refs a vector of size this->point_size(), where the nth element is a PointRef pointing to the nth Point (i.e., points to this->point(n)). If out_refs has something in it beforehand, it will be cleared prior to filling it.

**See also:**

PointRef

#### D.19.3.10 void GetPointRefs (int *index*, vector< PointRef > * *out_refs*) const [inherited]

Get PointRefs to every Point in this PointSetList.

Requires out_refs != NULL. Makes out_refs a vector of size this->point_set(index).size(), where the nth element is a PointRef pointing to the nth Point in the <index>th image. If out_refs has something in it beforehand, it will be cleared prior to filling it.

**See also:**

PointRef

#### D.19.3.11 void WriteToStream (ostream & *output_stream*) const [inherited]

Writes the entire PointSetList to a stream.

See the detailed description of PointSetList for the format.

#### D.19.3.12 void WriteHeaderToStream (ostream & *output_stream*) const [inherited]

Writes just the serialized header to a stream.

#### D.19.3.13 void WritePointSetsToStream (ostream & *output_stream*) const [inherited]

Writes the point sets (without a header) sequentially to a stream.

#### D.19.3.14 void WriteToFile (const char * *output_file*) const [inherited]

Writes the entire PointSetList to a file.

See detailed description of PointSetList for the format.

### D.19.4 Member Data Documentation

#### D.19.4.1 const int DEFAULT_LRU_CACHE_SIZE [static]

Default value: 1500.

#### D.19.4.2 const int DEFAULT_AREA_CACHE_SIZE [static]

Default value: 1500.

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

- point_set/on-disk-point-set-list.h
- point_set/on-disk-point-set-list.cc

## D.20 Point Class Reference

`#include <point.h>`

### D.20.1 Detailed Description

A generic abstract data structure storing a weighted vector of floats.

The size of the vector is determined at construction and can not be changed. The elements in the vector and weight are mutable.

### Public Member Functions

- Point (int dim)

  *Initially sets all elements in the vector to 0 and the weight to 1.*

- Point (const Point &other)
- virtual ~Point ()
- virtual void CopyFrom (const Point &other)
- int size () const

  *Returns the size of the vector.*

- int point_dim () const

  *Same as size().*

- float feature (int index) const

  *Returns the <index>th element with bounds checking.*

- void set_feature (int index, float value)

  *Sets the <index>th element to value, with bounds checking.*

- float weight () const
- void set_weight (float new_weight)
- float & operator[ ] (int index)

  *Returns a reference to the specified index. No bounds checking.*

- float operator[ ] (int index) const

  *Returns a reference to the specified index. No bounds checking.*

- virtual void WriteToStream (ostream &output_stream) const

  *Serialize to a stream.*

- virtual void ReadFromStream (istream &input_stream)

  *Read a vector of <dim> floats from a stream.*

### Protected Attributes

- float weight_
- vector< float > features_

### D.20.2 Constructor & Destructor Documentation

#### D.20.2.1 Point (int *dim*)

Initially sets all elements in the vector to 0 and the weight to 1.

#### D.20.2.2 Point (const Point & *other*)

#### D.20.2.3 virtual ~Point () [inline, virtual]

### D.20.3 Member Function Documentation

#### D.20.3.1 void CopyFrom (const Point & *other*) [virtual]

#### D.20.3.2 int size () const [inline]

Returns the size of the vector.

#### D.20.3.3 int point_dim () const [inline]

Same as size().

#### D.20.3.4 float feature (int *index*) const

Returns the <index>th element with bounds checking.

#### D.20.3.5 void set_feature (int *index*, float *value*)

Sets the <index>th element to value, with bounds checking.

#### D.20.3.6 float weight () const [inline]

#### D.20.3.7 void set_weight (float *new_weight*) [inline]

#### D.20.3.8 ] float & operator[ ] (int *index*)

Returns a reference to the specified index. No bounds checking.

#### D.20.3.9 ] float operator[ ] (int *index*) const

Returns a reference to the specified index. No bounds checking.

#### D.20.3.10 void WriteToStream (ostream & *output_stream*) const [virtual]

Serialize to a stream.

Simply writes the vector of floats to a stream. Note that this does NOT write the weight or the feature dim. Writing the weights and feature dim are handled by PointSetList. When overriding this in your own Point classes, you should do it such that you write all of the data except the weight or feature dim. It's OK to include them if you want, but it will be redundant.

#### D.20.3.11 void ReadFromStream (istream & *input_stream*) [virtual]

Read a vector of <dim> floats from a stream.

### D.20.4 Member Data Documentation

#### D.20.4.1 float weight_ [protected]

#### D.20.4.2 vector<float> features_ [protected]

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

- point_set/point.h
- point_set/point.cc

## D.21 PointRef Class Reference

`#include <point-ref.h>`

### D.21.1 Detailed Description

A way to reference Points in a particular PointSetList.

PointRef is basically a pair of things: (1) a pointer to a PointSetList, and (2) an index (or set of indices) into that PointSetList that references one particular Point. Since there are two ways to reference a particular Point in a PointSetList (either by specifying which PointSet it's in followed by the index of that point within that PointSet, or just by specifying a single index as if you were ignoring PointSet boundaries), PointRef will transparently convert between these two methods of describing a Point, making it easy to use PointSetLists for different applications.

Note that PointRefs become invalid if the PointSetList is modified. PointRef by itself does not perform any consistency checking.

**Public Member Functions**

- PointRef (const PointSetList *data, int point_index)

  *Refer to a point initially by specifying a single index.*

- PointRef (const PointSetList *data, int which_point_set, int which_point)

  *Refer to a point initially by specifying which PointSet and index.*

- PointRef (const PointSetList *data, int point_index, int which_point_set, int which_point)

  *Refer to a point by providing both methods of accessing a Feature.*

- PointRef (const PointRef &other)
- const Point & point () const

  *Get the Feature referred to by this PointRef.*

- int which_point_set () const

  *Find the index of the PointSet that this point is in.*

- int which_point () const

  *Find the index of the point within the PointSet that it's in.*

- int point_index () const

  *Get the index of the point in the list of all Points in this PointSetList.*

### D.21.2 Constructor & Destructor Documentation

#### D.21.2.1 PointRef (const PointSetList * *data*, int *point_index*)

Refer to a point initially by specifying a single index.

#### D.21.2.2 PointRef (const PointSetList * *data*, int *which_point_set*, int *which_point*)

Refer to a point initially by specifying which PointSet and index.

#### D.21.2.3 PointRef (const PointSetList * *data*, int *point_index*, int *which_point_set*, int *which_point*)

Refer to a point by providing both methods of accessing a Feature.

This constructor does not do any consistency checking i.e., to check whether feature_index refers to the same point as (which_point_set, which_point). You should only use this constructor if you know for sure; the reason for this constructor is to quickly build a PointRef if you already know all the indices.

#### D.21.2.4 PointRef (const PointRef & *other*)

### D.21.3 Member Function Documentation

#### D.21.3.1 const Point & point () const

Get the Feature referred to by this PointRef.

#### D.21.3.2 int which_point_set () const

Find the index of the PointSet that this point is in.

This will usually be used in conjunction with which_point().

#### D.21.3.3 int which_point () const

Find the index of the point within the PointSet that it's in.

This will usually be used in conjunction with which_point_set().

#### D.21.3.4 int point_index () const

Get the index of the point in the list of all Points in this PointSetList.

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

- point_set/point-ref.h
- point_set/point-ref.cc

## D.22 PointSet Class Reference

`#include <point-set.h>`

### D.22.1 Detailed Description

A data structure representing a list of Points.

All features in the set must have the same dimension, and this is determined when the PointSet is constructed. Points may be added, removed, and replaced by index. When points are inserted, PointSet will automatically check to make sure that the points all have the same dimension.

This is called a PointSet just to maintain name compatibility with previous versions, even though it is actually an array.

### Public Member Functions

- PointSet (int point_dim)

  *Creates an initially empty set (such that size() == 0).*

- ~PointSet ()
- PointSet (const PointSet &other)
- void CopyFrom (const PointSet &other)
- int size () const
- int point_dim () const
- void set_point_dim (int new_dim)

  *Set the point dim. Only works when size() == 0.*

- const Point & point (int index) const

  *Returns a const reference to the point at <index>.*

- Point * mutable_point (int index) const
- const Point & operator[ ] (int index) const
- Point * add_point (const Point &point)

  *Adds a copy of the point to the end of the list.*

- Point * add_point ()

  *Adds a new, empty (all zeroes) Point and returns a pointer to it.*

- void remove_point (int index)

  *Removes the point at <index>.*

- void clear ()

  *Deletes all of the points.*

- void WriteToStream (ostream &output_stream) const

  *Writes the point set to a stream.*

- void ReadFromStream (istream &input_stream)

  *Reads the point set from a stream.*

### D.22.2 Constructor & Destructor Documentation

#### D.22.2.1 PointSet (int *point_dim*)

Creates an initially empty set (such that size() == 0).

#### D.22.2.2 ~PointSet ()

#### D.22.2.3 PointSet (const PointSet & *other*)

### D.22.3 Member Function Documentation

#### D.22.3.1 void CopyFrom (const PointSet & *other*)

#### D.22.3.2 int size () const `[inline]`

#### D.22.3.3 int point_dim () const `[inline]`

#### D.22.3.4 void set_point_dim (int *new_dim*)

Set the point dim. Only works when size() == 0.

#### D.22.3.5 const Point & point (int *index*) const

Returns a const reference to the point at <index>.

#### D.22.3.6 Point * mutable_point (int *index*) const

#### D.22.3.7 ] const Point& operator[ ] (int *index*) const `[inline]`

#### D.22.3.8 Point * add_point (const Point & *point*)

Adds a copy of the point to the end of the list.

Returns a pointer to the Point that was just added.

#### D.22.3.9 Point * add_point ()

Adds a new, empty (all zeroes) Point and returns a pointer to it.

#### D.22.3.10 void remove_point (int *index*)

Removes the point at <index>.

#### D.22.3.11 void clear ()

Deletes all of the points.

#### D.22.3.12 void WriteToStream (ostream & *output_stream*) const

Writes the point set to a stream.

The format is as follows:

- (int32_t) num_features
- for each feature:
  - (float * point_dim) the feature vector
- then, again, for each feature:
  - (float) the feature vector's weight

  Note that the point dim is not written to the stream.

#### D.22.3.13 void ReadFromStream (istream & *input_stream*)

Reads the point set from a stream.

This function will clear any Features present currently. It will also assume that the incoming PointSet has the proper feature dim (observe that a serialized PointSet doesn't actually record feature dim– that is up to PointSetList to remember).

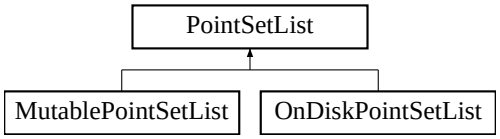
The documentation for this class was generated from the following files:

- point_set/point-set.h
- point_set/point-set.cc

## D.23 PointSetList Class Reference

`#include <point-set-list.h>`

Inheritance diagram for PointSetList::




### D.23.1 Detailed Description

Abstract interface for a list of PointSet.

All Points contained in any PointSetList must have the same dimension. When serialized, a PointSetList takes on the following format:

- (int32) N, the number of PointSets
- (int32) f, the dim of every point in this PointSetList
- For each PointSet:
  - (int32) The number of Points in this PointSet
  - For each point in this PointSet:
    - (Point) The point itself (generally, float * f)
  - Then, again for each Point in this PointSet:
    - (float) The weight of the Point

Equivalently, at a higher level:

- (int32) N, the number of PointSets
- (int32) f, the dim of every point in this PointSetList
- (N * PointSet) the N PointSets.

**See also:**

PointSet

### Public Member Functions

- virtual ~PointSetList ()
- virtual int point_set_size () const=0

  *Get the total number of PointSets in this PointSetList.*

- virtual int point_size () const=0

  *Get the total number of Points in this PointSetList.*

- virtual int point_dim () const=0

  *Get the dim of every Point in this PointSetList.*

- virtual bool GetSetCardinalities (vector< int > *destination) const

  *Get the number of Points in each PointSet.*

- const PointSet & operator[ ] (int index)

  *Returns a const ref to the <index>th PointSet.*

- virtual const PointSet & point_set (int index) const=0

  *Returns a const ref to the <index>th PointSet.*

- const Point & point (int index) const

  *Returns the <index>th Point in the PointSetList.*

- bool GetPointIndices (int index, int *which_point_set, int *which_point) const

  *Locate a Point in a PointSet.*

- void GetPointRefs (vector< PointRef > *out_refs) const

  *Get PointRefs to every Point in a particular image.*

- void GetPointRefs (int index, vector< PointRef > *out_refs) const

  *Get PointRefs to every Point in this PointSetList.*

- void WriteToStream (ostream &output_stream) const

  *Writes the entire PointSetList to a stream.*

- void WriteHeaderToStream (ostream &output_stream) const

  *Writes just the serialized header to a stream.*

- void WritePointSetsToStream (ostream &output_stream) const

  *Writes the point sets (without a header) sequentially to a stream.*

- void WriteToFile (const char *output_file) const

  *Writes the entire PointSetList to a file.*

### D.23.2 Constructor & Destructor Documentation

#### D.23.2.1 virtual ~PointSetList () [inline, virtual]

### D.23.3 Member Function Documentation

#### D.23.3.1 virtual int point_set_size () const [pure virtual]

Get the total number of PointSets in this PointSetList.

Implemented in MutablePointSetList, and OnDiskPointSetList.

#### D.23.3.2 virtual int point_size () const [pure virtual]

Get the total number of Points in this PointSetList.

This is the sum of PointSet::size() over all PointSets in in this PointSetList.

Implemented in MutablePointSetList, and OnDiskPointSetList.

#### D.23.3.3 virtual int point_dim () const [pure virtual]

Get the dim of every Point in this PointSetList.

Defined to be 0 if the PointSetList is empty.

Implemented in MutablePointSetList, and OnDiskPointSetList.

#### D.23.3.4 bool GetSetCardinalities (vector< int > * *destination*) const [virtual]

Get the number of Points in each PointSet.

Fills <destination> with a vector of size this.point_set_size(), where the nth element is the number of Points in the nth PointSet in this PointSetList. It is cleared first, and must not be NULL.

Returns true on success and false otherwise, although currently there is no reason that this method will fail.

Reimplemented in OnDiskPointSetList.

#### D.23.3.5 ] const PointSet& operator[ ] (int *index*) [inline]

Returns a const ref to the <index>th PointSet.

#### D.23.3.6 virtual const PointSet& point_set (int *index*) const [pure virtual]

Returns a const ref to the <index>th PointSet.

Implemented in MutablePointSetList, and OnDiskPointSetList.

#### D.23.3.7 const Point & point (int *index*) const

Returns the <index>th Point in the PointSetList.

We can also think of a PointSetList as a long vector of Points if we ignore the PointSet boundaries, so you can reference individual points of a PointSetList through this.

#### D.23.3.8 bool GetPointIndices (int *index*, int * *which_point_set*, int * *which_point*) const

Locate a Point in a PointSet.

Given an index specifying which point we are referring to, report which PointSet it belongs to, as well as the index into that PointSet that the PointSet belongs to.

Returns true if <index> is a valid Point and the data was successfully loaded, false otherwise.

#### D.23.3.9 void GetPointRefs (vector< PointRef > * *out_refs*) const

Get PointRefs to every Point in a particular image.

Requires out_refs != NULL. Makes out_refs a vector of size this->point_size(), where the nth element is a PointRef pointing to the nth Point (i.e., points to this->point(n)). If out_refs has something in it beforehand, it will be cleared prior to filling it.

**See also:**

PointRef

#### D.23.3.10 void GetPointRefs (int *index*, vector< PointRef > * *out_refs*) const

Get PointRefs to every Point in this PointSetList.

Requires out_refs != NULL. Makes out_refs a vector of size this->point_set(index).size(), where the nth element is a PointRef pointing to the nth Point in the <index>th image. If out_refs has something in it beforehand, it will be cleared prior to filling it.

**See also:**

PointRef

#### D.23.3.11 void WriteToStream (ostream & *output_stream*) const

Writes the entire PointSetList to a stream.

See the detailed description of PointSetList for the format.

#### D.23.3.12 void WriteHeaderToStream (ostream & *output_stream*) const

Writes just the serialized header to a stream.

#### D.23.3.13 void WritePointSetsToStream (ostream & *output_stream*) const

Writes the point sets (without a header) sequentially to a stream.

#### D.23.3.14 void WriteToFile (const char * *output_file*) const

Writes the entire PointSetList to a file.

See detailed description of PointSetList for the format.

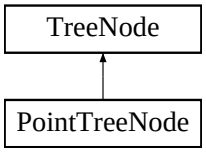
The documentation for this class was generated from the following files:

- point_set/point-set-list.h
- point_set/point-set-list.cc

## D.24 PointTreeNode Class Reference

`#include <point-tree-node.h>`

Inheritance diagram for PointTreeNode::




### D.24.1 Detailed Description

A TreeNode containing a Point.

A TreeNode has an integer identifier. It also contains the IDs of its parent and children, but makes no effort to maintain them.

To augment a TreeNode with your own data, there are three functions you need to provide: (1) reading the data from a stream, (2) writing the data to a stream, and (3) how to combine two nodes.

The definition of "combine" will vary depending on what you're doing, but the general idea is that sometimes you will have two TreeNodes with the same ID that you want to insert into a tree. It is advisable that you make the output of Combine() symmetric, since there are no guarantees on what Tree does with duplicate-ID bins other than calling Combine().

Remember that if you implement a copy constructor for your TreeNode, you should have your copy constructor call the base class's copy constructor:

```
MyNode::MyNode(const MyNode& other) : TreeNode(other) {
// your copy code
}
```

### Public Member Functions

- PointTreeNode ()
- PointTreeNode (int node_id)
- PointTreeNode (const PointTreeNode &other)

  *A copy constructor, which copies the ID and parent/child info.*

- const Point & point () const
- Point * mutable_point ()
- void set_point (const Point &point)
- bool has_point () const
- virtual ~PointTreeNode ()
- virtual void Combine (const TreeNode &other)

  *This should not happen for PointTreeNodes, so this method will always fail.*

- int id () const
- int parent () const
- int child (int child_index) const
- int child_size () const
- bool has_parent () const
- bool has_child () const
- void set_id (int id)
- void set_parent (int id)
- void add_child (int id)

  *Adds a child to the end of the child list.*

- void remove_child (int child_index)

*Removes the <child_index>th child.*

- void ReadFromStream (istream &input_stream)

  *Read the data, including the index, from a stream.*

- void WriteToStream (ostream &output_stream) const

  *Write the data, including the index, to a stream.*

### Static Public Attributes

- static const int kInvalidNodeID = -1

### Protected Member Functions

- virtual void ReadData (istream &input_stream)

  *Read the data, excluding the index, from a stream.*

- virtual void WriteData (ostream &output_stream) const

  *Write the data, excluding the index, to a stream.*

## D.24.2 Constructor & Destructor Documentation

### D.24.2.1 PointTreeNode ()

### D.24.2.2 PointTreeNode (int *node_id*)

### D.24.2.3 PointTreeNode (const PointTreeNode & *other*)

A copy constructor, which copies the ID and parent/child info.

### D.24.2.4 virtual ~PointTreeNode () `[inline, virtual]`

## D.24.3 Member Function Documentation

### D.24.3.1 const Point& point () const `[inline]`

#### D.24.3.2 Point* mutable_point () [inline]

#### D.24.3.3 void set_point (const Point & *point*) [inline]

#### D.24.3.4 bool has_point () const [inline]

#### D.24.3.5 void Combine (const TreeNode & *other*) [virtual]

This should not happen for PointTreeNodes, so this method will always fail.

Implements TreeNode.

#### D.24.3.6 void ReadData (istream & *input_stream*) [protected, virtual]

Read the data, excluding the index, from a stream.

Implements TreeNode.

#### D.24.3.7 void WriteData (ostream & *output_stream*) const [protected, virtual]

Write the data, excluding the index, to a stream.

Implements TreeNode.

#### D.24.3.8 int id () const [inline, inherited]

#### D.24.3.9 int parent () const [inline, inherited]

#### D.24.3.10 int child (int *child_index*) const [inline, inherited]

#### D.24.3.11 int child_size () const [inline, inherited]

#### D.24.3.12 bool has_parent () const [inline, inherited]

#### D.24.3.13 bool has_child () const [inline, inherited]

#### D.24.3.14 void set_id (int *id*) [inline, inherited]

#### D.24.3.15 void set_parent (int *id*) [inline, inherited]

#### D.24.3.16 void add_child (int *id*) [inline, inherited]

Adds a child to the end of the child list.

This does not check for dupes, i.e., it is possible for a node to have two children with the same ID using this, so be careful!

#### D.24.3.17 void remove_child (int *child_index*) [inherited]

Removes the <child_index>th child.

#### D.24.3.18 void ReadFromStream (istream & *input_stream*) [inherited]

Read the data, including the index, from a stream.

Calling this will override the node's current index. The format is:

- (int32_t) id
- (int32_t) parent_id
- (int32_t) num_children
- (num_children * int32_t) child IDs

It will then call ReadData().

**See also:**

ReadData()

#### D.24.3.19 void WriteToStream (ostream & *output_stream*) const [inherited]

Write the data, including the index, to a stream.

This will simply write the index to the stream, followed by WriteData().

**See also:**

WriteData().

### D.24.4 Member Data Documentation

#### D.24.4.1 const int kInvalidNodeID = -1 [static, inherited]

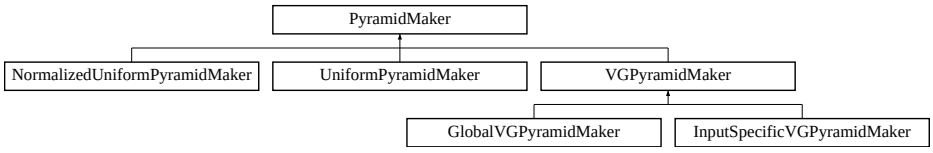
The documentation for this class was generated from the following files:

- tree/point-tree-node.h
- tree/point-tree-node.cc

## D.25 PyramidMaker Class Reference

`#include <pyramid-maker.h>`

Inheritance diagram for PyramidMaker::




### D.25.1 Detailed Description

Abstract interface for turning PointSets into MultiResolutionHistograms.

### Public Member Functions

- PyramidMaker ()
- virtual ~PyramidMaker ()
- virtual MultiResolutionHistogram * MakePyramid (const PointSet &ps)=0

  *Turn a single PointSet into a MultiResolutionHistogram.*

- vector< MultiResolutionHistogram * > MakePyramids (const PointSetList &psl)

  *Turn a list of PointSets into a bunch of MultiResolutionHistograms.*

### D.25.2 Constructor & Destructor Documentation

#### D.25.2.1 PyramidMaker () `[inline]`

#### D.25.2.2 virtual ~PyramidMaker () `[inline, virtual]`

### D.25.3 Member Function Documentation

#### D.25.3.1 virtual MultiResolutionHistogram* MakePyramid (const PointSet & *ps*) `[pure virtual]`

Turn a single PointSet into a MultiResolutionHistogram.

This function allocates memory on its own. It is up to the caller to free it.

Implemented in GlobalVGPyramidMaker, InputSpecificVGPyramidMaker, NormalizedUniformPyramidMaker, UniformPyramidMaker, and VGPyramidMaker.

#### D.25.3.2 vector< MultiResolutionHistogram * > MakePyramids (const PointSetList & *psl*)

Turn a list of PointSets into a bunch of MultiResolutionHistograms.

It is up to the caller to free the memory.

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

- pyramids/pyramid-maker.h
- pyramids/pyramid-maker.cc

## D.26 PyramidMatcher Class Reference

`#include <pyramid-matcher.h>`

### D.26.1 Detailed Description

Matches two MultiResolutionHistograms.

#### Static Public Member Functions

- static double GetPyramidMatchCost (const MultiResolutionHistogram &hist1, const MultiResolutionHistogram &hist2, BinWeightScheme bin_weight_scheme)

  *Returns the pyramid match cost/distance.*

- static double GetPyramidMatchSimilarity (const MultiResolutionHistogram &hist1, const MultiResolutionHistogram &hist2, BinWeightScheme bin_weight_scheme)

  *Returns the pyramid match similarity (kernel value).*

#### Classes

- class **MatchNode**

### D.26.2 Member Function Documentation

#### D.26.2.1 double GetPyramidMatchCost (const MultiResolutionHistogram & *hist1*, const MultiResolutionHistogram & *hist2*, BinWeightScheme *bin_weight_scheme*) [static]

Returns the pyramid match cost/distance.

**Parameters:**

***bin_weight_scheme*** depends on how the pyramid was generated.

**See also:**

GlobalVGPyramidMaker, InputSpecificVGPyramidMaker

#### D.26.2.2 double GetPyramidMatchSimilarity (const MultiResolutionHistogram & *hist1*, const MultiResolutionHistogram & *hist2*, BinWeightScheme *bin_weight_scheme*) [static]

Returns the pyramid match similarity (kernel value).

**Parameters:**

***bin_weight_scheme*** depends on how the pyramid was generated.

**See also:**

GlobalVGPyramidMaker, InputSpecificVGPyramidMaker

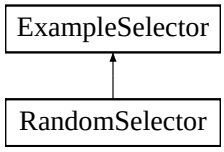
The documentation for this class was generated from the following files:

- pyramids/pyramid-matcher.h
- pyramids/pyramid-matcher.cc

## D.27 RandomSelector Class Reference

`#include <random-selector.h>`

Inheritance diagram for RandomSelector::



### D.27.1 Detailed Description

Chooses a random sample of each class to use as testing.

#### Public Member Functions

- RandomSelector (const vector< int > &labels, int num_train_per_class)
- vector< LabeledIndex > GetTrainingExamples ()

  *Returns a vector of LabeledIndices of training instances sorted by index.*

- vector< LabeledIndex > GetTestingExamples ()

  *Returns a vector of LabeledIndices of testing instances sorted by index.*

- vector< int > GetUniqueLabels ()

  *Returns a vector containing all the unique labels seen. Sorting is NOT guaranteed.*

- int GetNumExamples () const

  *Returns the total number of examples there are.*

#### Protected Member Functions

- virtual void SelectExamples ()

  *Responsible for calling AddTrainingExample and AddTestingExample to generate the lists.*

- const vector< LabeledIndex > & GetExamples () const

  *Get all of the examples (in the same order as given when constructed).*

- vector< LabeledIndex > InvertSelection (const vector< LabeledIndex > &selection) const

  *Returns all instances are NOT in <selection>.*

- vector< LabeledIndex > InvertSelection (const vector< LabeledIndex > &selection, const vector< LabeledIndex > &superset) const

  *Returns all instances in <superset> that are NOT in <selection>.*

- vector< LabeledIndex > RandomSample (int n, const vector< LabeledIndex > &selection) const

  *Return a random sample of size <n> of <selection>.*

- vector< LabeledIndex > GetInstancesWithLabel (int label, const vector< LabeledIndex > &selection) const

  *Gets all elements in <selection> with the given <label>.*

- void AddTrainingExample (LabeledIndex index)

  *Add a LabeledIndex to the list of designated training examples.*

- void AddTestingExample (LabeledIndex index)

  *Add a LabeledIndex to the list of designated testing examples.*

### D.27.2 Constructor & Destructor Documentation

#### D.27.2.1 RandomSelector (const vector< int > & *labels*, int *num_train_per_class*)

### D.27.3 Member Function Documentation

#### D.27.3.1 void SelectExamples () [protected, virtual]

Responsible for calling AddTrainingExample and AddTestingExample to generate the lists.

Implements ExampleSelector.

#### D.27.3.2 vector< LabeledIndex > GetTrainingExamples () [inherited]

Returns a vector of LabeledIndices of training instances sorted by index.

#### D.27.3.3 vector< LabeledIndex > GetTestingExamples () [inherited]

Returns a vector of LabeledIndices of testing instances sorted by index.

#### D.27.3.4 vector< int > GetUniqueLabels () [inherited]

Returns a vector containing all the unique labels seen. Sorting is NOT guaranteed.

#### D.27.3.5 int GetNumExamples () const [inherited]

Returns the total number of examples there are.

#### D.27.3.6 const vector< LabeledIndex > & GetExamples () const [protected, inherited]

Get all of the examples (in the same order as given when constructed).

#### D.27.3.7 vector< LabeledIndex > InvertSelection (const vector< LabeledIndex > & *selection*) const [protected, inherited]

Returns all instances are NOT in <selection>.

<selection> must be sorted by index. The output will be sorted by index.

#### D.27.3.8 vector< LabeledIndex > InvertSelection (const vector< LabeledIndex > & *selection*, const vector< LabeledIndex > & *superset*) const [protected, inherited]

Returns all instances in <superset> that are NOT in <selection>.

<selection> and <superset> must be sorted by index. The output will be sorted by index.

#### D.27.3.9 vector< LabeledIndex > RandomSample (int *n*, const vector< LabeledIndex > & *selection*) const [protected, inherited]

Return a random sample of size <n> of <selection>.

<selection> must be sorted by index. If the size of <selection> is less than n, this function will return a vector of size selection.size(). Output is sorted by index.

#### D.27.3.10 vector< LabeledIndex > GetInstancesWithLabel (int *label*, const vector< LabeledIndex > & *selection*) const [protected, inherited]

Gets all elements in <selection> with the given <label>.

<selection> must be sorted. The output will be sorted.

#### D.27.3.11 void AddTrainingExample (LabeledIndex *index*) [protected, inherited]

Add a LabeledIndex to the list of designated training examples.

#### D.27.3.12 void AddTestingExample (LabeledIndex *index*) [protected, inherited]

Add a LabeledIndex to the list of designated testing examples.

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

- experiment/random-selector.h

- experiment/random-selector.cc

## D.28 SparseTree Class Template Reference

`#include <sparse-tree.h>`

### D.28.1 Detailed Description

**template<class T> class libpmk::SparseTree< T >**

A data structure for sparsely representing trees containing SparseTreeNode objects.

This implementation allows one to quickly get the parent, child, or siblings of any node. All data put into a Tree must be a subclass of SparseTreeNode. SparseTreeNode will automatically manage all of the parent/child/sibling pointers as well as the LargeIndexes for you. For more information on how to implement a SparseTreeNode, see that page.

There are two ways to access nodes in these data structures: either by looking at the root node and traversing the tree manually by using the parent/child/sibling relationships, or by specifying a LargeIndex describing a path down the tree. These indices are set up and maintained such that at the nth level of the tree, the LargeIndex has size n (the root node has size 0 which is an empty LargeIndex). The ith element of the LargeIndex specifies the link traversed to get from the (i-1)st level to the ith level.

The sibling pointers are managed such that GetNextSibling() always returns a Bin with the next highest index in that branch of the tree. This multi-resolution histogram is stored sparsely: if you consider the Bins referred to by [0 1] and [0 3], it is possible that [0 3] is the sibling after [0 1] (as long as there is no such bin [0 2]).

Note that using LargeIndex *and* the link structure between Bins is redundant, as a LargeIndex specifies a path down the tree. When reading and writing the SparseTree to disk, we only persist the LargeIndexes– the link structure is inferred from the LargeIndex values upon reading from disk. The purpose of the link structure is to speed up computations and lookups.

** **If you have build problems:** remember that since SparseTree is a class template, when using your own SparseTreeNodes in a SparseTree, you'll run into linker errors if you don't do an explicit instantiation of SparseTree<YourSparseTreeNode>. To deal with this, you will need to do two things:

- include "tree/sparse-tree.cc" (yes, sparse-tree.cc, not sparse-tree.h)
- Add the line "template class Tree<YourTreeNode>;" in your code.

If you are still having problems:

- Make sure to implement two required `SparseTreeNode` constructors: the default one, which takes no arguments, and the one which takes a `const LargeIndex&`.
- If you are getting segmentation faults, it could be that you are trying to copy trees. Make sure that if you are doing so, and you have implemented a copy constructor for your Node class, be sure that it calls TreeNode's copy constructor during initialization. For example:

```
MyNode::MyNode(const MyNode& other) : SparseTreeNode(other) {
// your copy code
}
```

- Remember that `WriteData` is a `const` function.

**See also:**

SparseTreeNode

### Public Member Functions

- SparseTree ()
- ~SparseTree ()
- SparseTree (const SparseTree< T > &other)

  *Copy constructor.*

- SparseTree (const SparseTree< T > &one, const SparseTree< T > &two)
- T * root ()

  *Get a pointer to the root node.*

- T *const root () const

  *Get a pointer to the root node (const version).*

- int size () const

  *Get the total number of nodes in the tree.*

- T * node (const LargeIndex &index)

  *Get a node specified by the given index.*

- T * node (const LargeIndex &index, SparseTreeNode *finger)

  *Get a pointer to the node with the specified index.*

- T * add_node (const T &new_node)

*Insert a copy of the given node into the tree.*

- T * add_node (const T &new_node, SparseTreeNode *parent)

  *Insert a copy of the given bin into the tree.*

- void ReadFromStream (istream &input_stream)

  *Reads a tree from the stream.*

- void WriteToStream (ostream &output_stream) const

  *Write the tree to a stream.*

- void remove_node (SparseTreeNode *node)

  *Removes a node, and all of its children, from the tree.*

- PreorderIterator BeginPreorder () const

  *Start of a preorder depth-first traversal.*

- PostorderIterator BeginPostorder () const

  *Start of a postorder depth-first traversal.*

- BreadthFirstIterator BeginBreadthFirst () const

  *Start of a breadth-first traversal.*

### Static Public Member Functions

- static const PreorderIterator & EndPreorder ()

  *End of a preorder depth-first traversal.*

- static const PostorderIterator & EndPostorder ()

  *End of a postorder depth-first traversal.*

- static const BreadthFirstIterator & EndBreadthFirst ()

  *End of a breadth-first traversal.*

**Classes**

- class BreadthFirstIterator

  *Breadth-first iterator for SparseTrees.*

- class Iterator

  *An iterator for SparseTrees.*

- class PostorderIterator
- class PreorderIterator

  *Preorder depth-first iterator for SparseTrees.*

### D.28.2 Constructor & Destructor Documentation

#### D.28.2.1 SparseTree ()

#### D.28.2.2 ~SparseTree ()

#### D.28.2.3 SparseTree (const SparseTree< T > & *other*)

Copy constructor.

#### D.28.2.4 SparseTree (const SparseTree< T > & *one*, const SparseTree< T > & *two*)

### D.28.3 Member Function Documentation

#### D.28.3.1 T* root () `[inline]`

Get a pointer to the root node.

#### D.28.3.2 T* const root () const `[inline]`

Get a pointer to the root node (const version).

#### D.28.3.3 int size () const `[inline]`

Get the total number of nodes in the tree.

#### D.28.3.4 T * node (const LargeIndex & *index*)

Get a node specified by the given index.

Returns NULL if the node is not found.

#### D.28.3.5 T * node (const LargeIndex & *index*, SparseTreeNode * *finger*)

Get a pointer to the node with the specified index.

Same as GetNode(LargeIndex), but localizes the search to the subtree given by <finger>. Returns NULL if there is no such node in a subtree of <finger>.

#### D.28.3.6 T * add_node (const T & *new_node*)

Insert a copy of the given node into the tree.

Returns a pointer to the newly-added node in the tree. This function completely ignores any parent/child/sibling pointers in <new_node>.

This function requires the parent bin to already exist in the tree. It will NOT create new bins along the way (it will abort if there is no parent node, i.e., a node whose index is a prefix of that of <new_node> and whose index size is exactly 1 less than the new node's index size. The insertion happens such that the sibling relationships remain sorted by index.

If there is already is a node with the given index (call it N), we call N.Combine(new_node), and no new node is inserted into the tree. This applies to the root node as well.

#### D.28.3.7 T * add_node (const T & *new_node*, SparseTreeNode * *parent*)

Insert a copy of the given bin into the tree.

Same as AddBin(const Node&), except it starts the search for the bin at <parent>. In this case, <parent> must be the direct parent of the node we are inserting (it cannot be a higher-up ancestor).

#### D.28.3.8 void ReadFromStream (istream & *input_stream*)

Reads a tree from the stream.

File format:

- (int32) The total number of bins, including the root
- For each bin:
  - (int32) L, the size of its index (0 for the root, etc.)
  - (L * int32) the index

– (sizeof(data)) the data, from T.WriteToStream().
The ordering of the bins is a depth-first traversal of the tree.
Aborts if the stream is bad.

#### D.28.3.9 void WriteToStream (ostream & *output_stream*) const

Write the tree to a stream.

Aborts if the stream is bad. See ReadFromStream for the format.

**See also:**

ReadFromStream

#### D.28.3.10 void remove_node (SparseTreeNode * *node*)

Removes a node, and all of its children, from the tree.

The node must be owned by this tree. SparseTree will not check for that.

#### D.28.3.11 SparseTree< T >::PreorderIterator BeginPreorder () const

Start of a preorder depth-first traversal.

#### D.28.3.12 const SparseTree< T >::PreorderIterator & EndPreorder () [static]

End of a preorder depth-first traversal.

#### D.28.3.13 SparseTree< T >::PostorderIterator BeginPostorder () const

Start of a postorder depth-first traversal.

#### D.28.3.14 const SparseTree< T >::PostorderIterator & EndPostorder () [static]

End of a postorder depth-first traversal.

#### D.28.3.15 SparseTree< T >::BreadthFirstIterator BeginBreadthFirst () const

Start of a breadth-first traversal.

#### D.28.3.16 const SparseTree< T >::BreadthFirstIterator & EndBreadthFirst () [static]

End of a breadth-first traversal.

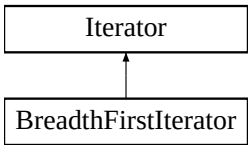
The documentation for this class was generated from the following files:

- tree/sparse-tree.h
- tree/sparse-tree.cc

## D.29 SparseTree::BreadthFirstIterator Class Reference

`#include <sparse-tree.h>`

Inheritance diagram for SparseTree::BreadthFirstIterator::




### D.29.1 Detailed Description

**template<class T> class libpmk::SparseTree< T >::BreadthFirstIterator**

Breadth-first iterator for SparseTrees.

### Public Member Functions

- BreadthFirstIterator (T *node)
- virtual ~BreadthFirstIterator ()
- virtual Iterator & operator++ ()
- bool operator== (const Iterator &other)

  *Returns true iff <other> points to the same node in memory.*

- bool operator!= (const Iterator &other)

  *Returns true iff <other> points to a different node in memory.*

- T * operator → ()

  *Accesses the pointer to the SparseTreeNode.*

- T * get ()

  *Returns a pointer to the SparseTreeNode.*

**Protected Attributes**

- SparseTreeNode * node_

### D.29.2 Constructor & Destructor Documentation

#### D.29.2.1 BreadthFirstIterator (T * *node*) [inline]

Creates a new iterator rooted at <node>. The resulting traversal will ignore any parents of <node>.

#### D.29.2.2 virtual ~BreadthFirstIterator () [inline, virtual]

### D.29.3 Member Function Documentation

#### D.29.3.1 SparseTree< T >::Iterator & operator++ () [virtual]

Implements SparseTree::Iterator.

#### D.29.3.2 bool operator== (const Iterator & *other*) [inline, inherited]

Returns true iff <other> points to the same node in memory.

#### D.29.3.3 bool operator!= (const Iterator & *other*) [inline, inherited]

Returns true iff <other> points to a different node in memory.

#### D.29.3.4 T* operator -> () [inline, inherited]

Accesses the pointer to the SparseTreeNode.

#### D.29.3.5 T* get () [inline, inherited]

Returns a pointer to the SparseTreeNode.

### D.29.4 Member Data Documentation

#### D.29.4.1 SparseTreeNode* node_ [protected, inherited]

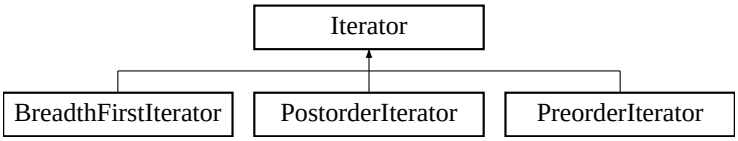
The documentation for this class was generated from the following files:

- tree/sparse-tree.h
- tree/sparse-tree.cc

## D.30 SparseTree::Iterator Class Reference

`#include <sparse-tree.h>`

Inheritance diagram for SparseTree::Iterator::




### D.30.1 Detailed Description

**template<class T> class libpmk::SparseTree< T >::Iterator**

An iterator for SparseTrees.

### Public Member Functions

- Iterator (SparseTreeNode *node)

  *Creates a new iterator pointing at the given node.*

- virtual ~Iterator ()
- Iterator & operator= (const Iterator &other)

  *Copy assignment operator.*

- bool operator== (const Iterator &other)

  *Returns true iff <other> points to the same node in memory.*

- bool operator!= (const Iterator &other)

  *Returns true iff <other> points to a different node in memory.*

- T * operator → ()

  *Accesses the pointer to the SparseTreeNode.*

- T * get ()

  *Returns a pointer to the SparseTreeNode.*

- virtual Iterator & operator++ ()=0

**Protected Attributes**

- SparseTreeNode * node_

### D.30.2 Constructor & Destructor Documentation

#### D.30.2.1 Iterator (SparseTreeNode * *node*) [inline]

Creates a new iterator pointing at the given node.

#### D.30.2.2 virtual ~Iterator () [inline, virtual]

### D.30.3 Member Function Documentation

#### D.30.3.1 Iterator& operator= (const Iterator & *other*) [inline]

Copy assignment operator.

#### D.30.3.2 bool operator== (const Iterator & *other*) [inline]

Returns true iff <other> points to the same node in memory.

#### D.30.3.3 bool operator!= (const Iterator & *other*) [inline]

Returns true iff <other> points to a different node in memory.

#### D.30.3.4 T* operator → () [inline]

Accesses the pointer to the SparseTreeNode.

#### D.30.3.5 T* get () [inline]

Returns a pointer to the SparseTreeNode.

#### D.30.3.6 virtual Iterator& operator++ () [pure virtual]

Implemented in SparseTree::PreorderIterator, SparseTree::PostorderIterator, and SparseTree::BreadthFirstIterator.

### D.30.4 Member Data Documentation

#### D.30.4.1 SparseTreeNode* node_ [protected]

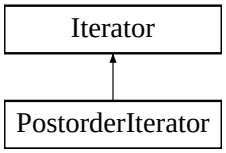
The documentation for this class was generated from the following file:

- tree/sparse-tree.h

## D.31 SparseTree::PostorderIterator Class Reference

`#include <sparse-tree.h>`

Inheritance diagram for SparseTree::PostorderIterator::




**template<class T> class libpmk::SparseTree< T >::PostorderIterator**

**Public Member Functions**

- PostorderIterator (T *node)
- virtual ~PostorderIterator ()
- virtual Iterator & operator++ ()
- bool operator== (const Iterator &other)

  *Returns true iff <other> points to the same node in memory.*

- bool operator!= (const Iterator &other)

  *Returns true iff <other> points to a different node in memory.*

- T ∗ operator → ()

  *Accesses the pointer to the SparseTreeNode.*

- T ∗ get ()

  *Returns a pointer to the SparseTreeNode.*

**Protected Attributes**

- SparseTreeNode ∗ node_

### D.31.1 Constructor & Destructor Documentation

#### D.31.1.1 PostorderIterator (T ∗ *node*)

Creates a new iterator rooted at <node>. The resulting traversal will ignore any parents of <node>.

#### D.31.1.2 virtual ∼PostorderIterator () `[inline, virtual]`

### D.31.2 Member Function Documentation

#### D.31.2.1 SparseTree< T >::Iterator & operator++ () `[virtual]`

Implements SparseTree::Iterator.

#### D.31.2.2 bool operator== (const Iterator & *other*) `[inline, inherited]`

Returns true iff <other> points to the same node in memory.

#### D.31.2.3 bool operator!= (const Iterator & *other*) `[inline, inherited]`

Returns true iff <other> points to a different node in memory.

#### D.31.2.4 T∗ operator → () `[inline, inherited]`

Accesses the pointer to the SparseTreeNode.

#### D.31.2.5 T∗ get () `[inline, inherited]`

Returns a pointer to the SparseTreeNode.

### D.31.3 Member Data Documentation

#### D.31.3.1 SparseTreeNode* node_ [protected, inherited]

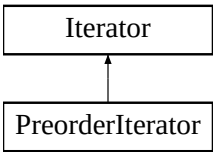
The documentation for this class was generated from the following files:

- tree/sparse-tree.h
- tree/sparse-tree.cc

## D.32 SparseTree::PreorderIterator Class Reference

`#include <sparse-tree.h>`

Inheritance diagram for SparseTree::PreorderIterator::




### D.32.1 Detailed Description

**template<class T> class libpmk::SparseTree< T >::PreorderIterator**

Preorder depth-first iterator for SparseTrees.

### Public Member Functions

- PreorderIterator (T *node)
- virtual ~PreorderIterator ()
- virtual Iterator & operator++ ()
- bool operator== (const Iterator &other)

  *Returns true iff <other> points to the same node in memory.*

- bool operator!= (const Iterator &other)

  *Returns true iff <other> points to a different node in memory.*

- T * operator → ()

  *Accesses the pointer to the SparseTreeNode.*

- T ∗ get ()

  *Returns a pointer to the SparseTreeNode.*

**Protected Attributes**

- SparseTreeNode ∗ node_

### D.32.2 Constructor & Destructor Documentation

#### D.32.2.1 PreorderIterator (T ∗ *node*) [inline]

Creates a new iterator rooted at <node>. The resulting traversal will ignore any parents of <node>.

#### D.32.2.2 virtual ∼PreorderIterator () [inline, virtual]

### D.32.3 Member Function Documentation

#### D.32.3.1 SparseTree< T >::Iterator & operator++ () [virtual]

Implements SparseTree::Iterator.

#### D.32.3.2 bool operator== (const Iterator & *other*) [inline, inherited]

Returns true iff <other> points to the same node in memory.

#### D.32.3.3 bool operator!= (const Iterator & *other*) [inline, inherited]

Returns true iff <other> points to a different node in memory.

#### D.32.3.4 T∗ operator → () [inline, inherited]

Accesses the pointer to the SparseTreeNode.

#### D.32.3.5 T∗ get () [inline, inherited]

Returns a pointer to the SparseTreeNode.

### D.32.4 Member Data Documentation

#### D.32.4.1 SparseTreeNode* node_ [protected, inherited]

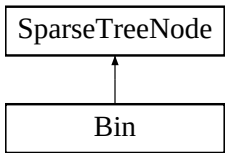
The documentation for this class was generated from the following files:

- tree/sparse-tree.h
- tree/sparse-tree.cc

## D.33 SparseTreeNode Class Reference

`#include <sparse-tree-node.h>`

Inheritance diagram for SparseTreeNode::




### D.33.1 Detailed Description

An indexed node, used by SparseTree.

A SparseTreeNode has an identifier of type LargeIndex (a vector of ints). SparseTreeNode also contains pointers to parents, children, and siblings, but makes no effort to maintain about them, and as such, we make no guarantees on it. For example, the "sibling" pointer can point to an arbitrary bin which may or may not be the real sibling. Tree handles the getting/setting of these pointers.

To augment a SparseTreeNode with your own data, there are three functions you need to provide: (1) reading the data from a stream, (2) writing the data to a stream, and (3) how to combine two nodes. The definition of "combine" will vary depending on what you're doing, but the general idea is that sometimes you will have two SparseTreeNodes with the same LargeIndex that you want to insert into a tree. It is advisable that you make the output of Combine() is symmetric, since there are no guarantees on what Tree does with duplicate-index bins other than calling Combine().

Remember that if you implement a copy constructor for your SparseTreeNode, you should have your copy constructor call the base class's copy constructor:

```
MyNode::MyNode(const MyNode& other) : SparseTreeNode(other) {
// your copy code
}
```

### Public Member Functions

- SparseTreeNode ()

  *Create a new node with an empty index.*

- SparseTreeNode (const LargeIndex &index)

  *Create a new node with the given index.*

- SparseTreeNode (const SparseTreeNode &other)

  *A 'copy' constructor, which only copies over the index.*

- virtual ~SparseTreeNode ()
- const LargeIndex & index () const
- SparseTreeNode * parent ()
- SparseTreeNode * prev_sibling ()
- SparseTreeNode * next_sibling ()
- SparseTreeNode * first_child ()
- SparseTreeNode * last_child ()
- void set_parent (SparseTreeNode *parent)
- void set_prev_sibling (SparseTreeNode *sibling)
- void set_next_sibling (SparseTreeNode *sibling)
- void set_first_child (SparseTreeNode *child)
- void set_last_child (SparseTreeNode *child)
- void set_index (const LargeIndex &index)
- bool has_parent () const
- bool has_prev_sibling () const
- bool has_next_sibling () const
- bool has_child () const
- void ReadFromStream (istream &input_stream)

  *Read the data, including the index, from a stream.*

- void WriteToStream (ostream &output_stream) const

  *Write the data, including the index, to a stream.*

- virtual void Combine (const SparseTreeNode &other)=0

  *Add the data from <other> to self.*

- bool operator< (const SparseTreeNode &other) const

  *Returns true if this node's index is smaller than that of <other>.*

### Static Public Member Functions

- static bool CompareNodes (const SparseTreeNode *one, const SparseTreeNode *two)

  *Returns true if <one>'s index is smaller than that of <two>.*

### Protected Member Functions

- virtual void ReadData (istream &input_stream)=0

  *Read the data, excluding the index, from a stream.*

- virtual void WriteData (ostream &output_stream) const=0

  *Write the data, excluding the index, to a stream.*

## D.33.2 Constructor & Destructor Documentation

### D.33.2.1 SparseTreeNode ()

Create a new node with an empty index.

### D.33.2.2 SparseTreeNode (const LargeIndex & *index*)

Create a new node with the given index.

### D.33.2.3 SparseTreeNode (const SparseTreeNode & *other*)

A 'copy' constructor, which only copies over the index.

The pointers to data will NOT be copied over; they will be NULL.

### D.33.2.4 ~SparseTreeNode () `[virtual]`

## D.33.3 Member Function Documentation

### D.33.3.1 const LargeIndex& index () const `[inline]`

### D.33.3.2 SparseTreeNode* parent () `[inline]`

### D.33.3.3 SparseTreeNode* prev_sibling () `[inline]`

#### D.33.3.4 SparseTreeNode* next_sibling () [inline]

#### D.33.3.5 SparseTreeNode* first_child () [inline]

#### D.33.3.6 SparseTreeNode* last_child () [inline]

#### D.33.3.7 void set_parent (SparseTreeNode * *parent*) [inline]

#### D.33.3.8 void set_prev_sibling (SparseTreeNode * *sibling*) [inline]

#### D.33.3.9 void set_next_sibling (SparseTreeNode * *sibling*) [inline]

#### D.33.3.10 void set_first_child (SparseTreeNode * *child*) [inline]

#### D.33.3.11 void set_last_child (SparseTreeNode * *child*) [inline]

#### D.33.3.12 void set_index (const LargeIndex & *index*) [inline]

#### D.33.3.13 bool has_parent () const [inline]

#### D.33.3.14 bool has_prev_sibling () const [inline]

#### D.33.3.15 bool has_next_sibling () const [inline]

#### D.33.3.16 bool has_child () const [inline]

#### D.33.3.17 void ReadFromStream (istream & *input_stream*)

Read the data, including the index, from a stream.

Calling this will override the node's current index. The format is:

- (int32_t) index_size
- (index_size * int32_t) the index

It will then call ReadData().

**See also:**

ReadData()

#### D.33.3.18 void WriteToStream (ostream & *output_stream*) const

Write the data, including the index, to a stream.

This will simply write the index to the stream, followed by WriteData().

**See also:**

WriteData().

#### D.33.3.19 virtual void Combine (const SparseTreeNode & *other*) [pure virtual]

Add the data from <other> to self.

Implemented in Bin.

#### D.33.3.20 bool operator< (const SparseTreeNode & *other*) const

Returns true if this node's index is smaller than that of <other>.

#### D.33.3.21 bool CompareNodes (const SparseTreeNode * *one*, const SparseTreeNode * *two*) [static]

Returns true if <one>'s index is smaller than that of <two>.

#### D.33.3.22 virtual void ReadData (istream & *input_stream*) [protected, pure virtual]

Read the data, excluding the index, from a stream.

Implemented in Bin.

#### D.33.3.23 virtual void WriteData (ostream & *output_stream*) const [protected, pure virtual]

Write the data, excluding the index, to a stream.

Implemented in Bin.

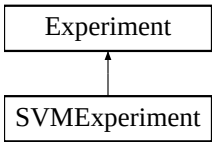
The documentation for this class was generated from the following files:

- tree/sparse-tree-node.h
- tree/sparse-tree-node.cc

## D.34 SVMExperiment Class Reference

`#include <svm-experiment.h>`

Inheritance diagram for SVMExperiment::




### D.34.1 Detailed Description

Runs an experiment using LIBSVM.

This will use a one-vs-all classifier.

For more information about LIBSVM, please see

Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

### Public Member Functions

- SVMExperiment (vector< LabeledIndex > training, vector< LabeledIndex > testing, const KernelMatrix &kernel, double c)
- SVMExperiment (vector< LabeledIndex > training, const KernelMatrix &training_matrix, vector< LabeledIndex > testing, const Matrix &testing_matrix, double c)
- virtual ~SVMExperiment ()
- virtual void Train ()

  *Train a model (if applicable).*

- virtual int Test ()

  *Make predictions for each testing example.*

- void ReadFromFile (const char *filename)
- void WriteToFile (const char *filename) const
- int GetPrediction (int test_index) const

  *Returns the predicted value of the <test_index>th test example.*

- int GetNumCorrect () const

*Get the total number of testing examples that were correctly classified.*

- int GetNumCorrect (int label) const

  *Get the number of testing examples that had label <label> that were also correctly classified.*

- int GetNumTestExamples () const

  *Get the total number of test examples.*

- int GetNumTestExamples (int label) const

  *Get the number of text examples with label <label>.*

- double GetAccuracy () const

  *Same as GetNumCorrect() / GetNumTestExamples().*

- double GetAccuracy (int label) const

  *Same as GetNumCorrect(label) / GetNumTestExamples(label).*

**Protected Member Functions**

- double GetKernelValue (int row, int col) const

  *Get a kernel value (wrapper for KernelMatrix).*

- double GetKernelValue (const LabeledIndex &row, const LabeledIndex &col) const

  *Get the kernel value corresponding to the given LabeledIndices.*

- void SetPrediction (int test_index, int prediction)

**Protected Attributes**

- vector< LabeledIndex > training_
- vector< LabeledIndex > testing_

### D.34.2 Constructor & Destructor Documentation

#### D.34.2.1 SVMExperiment (vector< LabeledIndex > *training*, vector< LabeledIndex > *testing*, const KernelMatrix & *kernel*, double *c*)

<kernel> includes pairwise kernel values for all data (both training and testing). The LabeledIndices in <training> and <testing> specify which row of the kernel to look at.

#### D.34.2.2 SVMExperiment (vector< LabeledIndex > *training*, const KernelMatrix & *training_matrix*, vector< LabeledIndex > *testing*, const Matrix & *testing_matrix*, double *c*)

<training_matrix> is a kernel matrix for training examples only. Let N be the number of training examples. Then <testing_matrix> is a NxM Matrix where M is the number of test examples, and the testing[i][j] is the kernel value between the i'th training example and the j'th test example. <training> must be N-dimensional and <testing> must be M-dimensional.

#### D.34.2.3 ~SVMExperiment () [virtual]

### D.34.3 Member Function Documentation

#### D.34.3.1 void Train () [virtual]

Train a model (if applicable).

Implements Experiment.

#### D.34.3.2 int Test () [virtual]

Make predictions for each testing example.

Returns the number of test examples that were correct. When you implement Test(), you must report results via SetPrediction().

Implements Experiment.

#### D.34.3.3 void ReadFromFile (const char * *filename*)

#### D.34.3.4 void WriteToFile (const char * *filename*) const

#### D.34.3.5 int GetPrediction (int *test_index*) const [inherited]

Returns the predicted value of the <test_index>th test example.

Can only call this after Test() is called.

#### D.34.3.6 int GetNumCorrect () const [inherited]

Get the total number of testing examples that were correctly classified.

#### D.34.3.7 int GetNumCorrect (int *label*) const [inherited]

Get the number of testing examples that had label <label> that were also correctly classified.

#### D.34.3.8 int GetNumTestExamples () const [inherited]

Get the total number of test examples.

#### D.34.3.9 int GetNumTestExamples (int *label*) const [inherited]

Get the number of text examples with label <label>.

#### D.34.3.10 double GetAccuracy () const [inherited]

Same as GetNumCorrect() / GetNumTestExamples().

#### D.34.3.11 double GetAccuracy (int *label*) const [inherited]

Same as GetNumCorrect(label) / GetNumTestExamples(label).

#### D.34.3.12 double GetKernelValue (int *row*, int *col*) const [protected, inherited]

Get a kernel value (wrapper for KernelMatrix).

#### D.34.3.13 double GetKernelValue (const LabeledIndex & *row*, const LabeledIndex & *col*) const [protected, inherited]

Get the kernel value corresponding to the given LabeledIndices.

#### D.34.3.14 void SetPrediction (int *test_index*, int *prediction*) [protected, inherited]

Call this to tell Experiment's internals that the test example at <test_index> was classified as <prediction>.

### D.34.4 Member Data Documentation

#### D.34.4.1 vector<LabeledIndex> training_ [protected, inherited]

#### D.34.4.2 vector<LabeledIndex> testing_ [protected, inherited]

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

- experiment/svm-experiment.h
- experiment/svm-experiment.cc

## D.35 Tree Class Template Reference

`#include <tree.h>`

### D.35.1 Detailed Description

**template<class T> class libpmk::Tree< T >**

A data structure for representing trees containing TreeNode objects.

This implementation stores nodes indexed by ID (a single integer). The root will always have ID 0, but the IDs of other nodes are not in any particular order. The tree structure is obtained by asking any particular node what the IDs of its parent or children are.

Access to the tree is simple: give the Tree a node ID and it will return a pointer to the node with that ID.

** **If you have build problems:** remember that since Tree is a class template, when using your own TreeNodes in a Tree, you'll run into linker errors if you don't do an explicit instantiation of Tree<YourTreeNode>. To deal with this, you will need to do two things:

- include "tree/tree.cc" (yes, tree.cc, not tree.h)
- Add the line "template class Tree<YourTreeNode>;" in your code.

If you are still having problems:

- Make sure to implement two required `TreeNode` constructors: the default one, which takes no arguments, and the one which takes an int (ID).
- If you are getting segmentation faults, it could be that you are trying to copy trees. Make sure that if you are doing so, and you have implemented a copy constructor for your Node class, be sure that it calls TreeNode's copy constructor during initialization. For example:

```
MyNode::MyNode(const MyNode& other) : TreeNode(other) {
// your copy code
}
```

- Remember that `WriteData` is a `const` function.

**See also:**

TreeNode

### Public Member Functions

- Tree ()
- ~Tree ()
- Tree (const Tree< T > &other)

  *Copy constructor. It will do a deep copy of all of the TreeNodes.*

- void clear ()

  *Return the tree to a state as if it were just constructed by Tree().*

- T * root ()

  *Get a pointer to the root node.*

- const T * root () const
- int size () const

  *Get the total number of nodes in the tree.*

- T * node (int id)

  *Get a node specified by the given index.*

- const T * node (int id) const
- T * add_node (const T &new_node)

  *Insert a copy of the given node into the tree.*

- T * add_node ()

  *Insert a blank node into the tree.*

- void ReadFromStream (istream &input_stream)

  *Reads a tree from the stream.*

- void WriteToStream (ostream &output_stream) const

  *Write the tree to a stream.*

- void DeleteNode (int node_id)

  *Frees a node, and all of its children, from the tree.*

- PreorderIterator BeginPreorder () const

  *Start of a preorder depth-first traversal.*

- const PreorderIterator & EndPreorder () const

  *End of a preorder depth-first traversal.*

- PostorderIterator BeginPostorder () const

  *Start of a postorder depth-first traversal.*

- const PostorderIterator & EndPostorder () const

  *End of a postorder depth-first traversal.*

- BreadthFirstIterator BeginBreadthFirst () const

  *Start of a breadth-first traversal.*

- const BreadthFirstIterator & EndBreadthFirst () const

  *End of a breadth-first traversal.*

**Classes**

- class BreadthFirstIterator

  *Breadth-first iterator for Trees.*

- class Iterator

  *An iterator for Trees.*

- class PostorderIterator
- class PreorderIterator

  *Preorder depth-first iterator for Trees.*

### D.35.2 Constructor & Destructor Documentation

#### D.35.2.1 Tree ()

#### D.35.2.2 ~Tree ()

#### D.35.2.3 Tree (const Tree< T > & *other*)

Copy constructor. It will do a deep copy of all of the TreeNodes.

### D.35.3 Member Function Documentation

#### D.35.3.1 void clear ()

Return the tree to a state as if it were just constructed by Tree().

#### D.35.3.2 T* root () [inline]

Get a pointer to the root node.

#### D.35.3.3 const T* root () const [inline]

#### D.35.3.4 int size () const [inline]

Get the total number of nodes in the tree.

#### D.35.3.5 T * node (int *id*)

Get a node specified by the given index.

Returns NULL if the node is not found.

#### D.35.3.6 const T * node (int *id*) const

#### D.35.3.7 T * add_node (const T & *new_node*)

Insert a copy of the given node into the tree.

Returns a pointer to the newly-added node in the tree.

If the new node's ID is uninitialized or invalid, it will be assigned a new unique ID. The returned pointer is a pointer to the newly-added node, so the user should call set_parent() and so forth in order to link the new node to the tree.

If the new node's ID is valid, but the tree does not contain a node with that ID, a new node will be created. The user still needs to link it to the tree manually via set_parent() and set_child().

If there already is a node in the tree with the same ID as new_node.id() (call it N), this will call N.Combine(new_node). In this case, the already-existing tree structure will be used (no need to call set_parent() and so forth, since the node is already there).

If you are having problems with this, i.e., you are calling add_node() but the data seems to not be copied into the tree, make sure that you have provided a copy constructor for your TreeNode.

#### D.35.3.8 T * add_node ()

Insert a blank node into the tree.

Returns a pointer to the newly added node. The resulting node will have no parent/child pointers, so it is not actually linked to the tree. It is up to the caller to link it to the tree by calling TreeNode::add_child() and TreeNode::set_parent().

#### D.35.3.9 void ReadFromStream (istream & *input_stream*)

Reads a tree from the stream.

File format:

- (int32) The total number of nodes, including the root
- (num_nodes * TreeNode) The nodes, in no particular order.

  Aborts if the stream is bad. Recall that the root of the tree always has ID 0.

#### D.35.3.10 void WriteToStream (ostream & *output_stream*) const

Write the tree to a stream.

Aborts if the stream is bad. See ReadFromStream for the format.

**See also:**

ReadFromStream

#### D.35.3.11 void DeleteNode (int *node_id*)

Frees a node, and all of its children, from the tree.

#### D.35.3.12 Tree< T >::PreorderIterator BeginPreorder () const

Start of a preorder depth-first traversal.

#### D.35.3.13 const Tree< T >::PreorderIterator & EndPreorder () const

End of a preorder depth-first traversal.

#### D.35.3.14 Tree< T >::PostorderIterator BeginPostorder () const

Start of a postorder depth-first traversal.

#### D.35.3.15 const Tree< T >::PostorderIterator & EndPostorder () const

End of a postorder depth-first traversal.

#### D.35.3.16 Tree< T >::BreadthFirstIterator BeginBreadthFirst () const

Start of a breadth-first traversal.

#### D.35.3.17 const Tree< T >::BreadthFirstIterator & EndBreadthFirst () const

End of a breadth-first traversal.

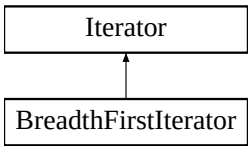
The documentation for this class was generated from the following files:

- tree/tree.h
- tree/tree.cc

## D.36 Tree::BreadthFirstIterator Class Reference

`#include <tree.h>`

Inheritance diagram for Tree::BreadthFirstIterator::




### D.36.1 Detailed Description

**template<class T> class libpmk::Tree< T >::BreadthFirstIterator**

Breadth-first iterator for Trees.

**Public Member Functions**

- BreadthFirstIterator (int node_id, const Tree< T > *tree)
- virtual ~BreadthFirstIterator ()
- virtual Iterator & operator++ ()
- bool operator== (const Iterator &other)

  *Returns true iff <other> points to the same node.*

- bool operator!= (const Iterator &other)

  *Returns true iff <other> points to a different node.*

- int operator * ()
- int id ()

**Protected Attributes**

- int node_id_
- const Tree< T > * source_tree_

### D.36.2 Constructor & Destructor Documentation

#### D.36.2.1 BreadthFirstIterator (int *node_id*, const Tree< T > * *tree*) `[inline]`

Creates a new iterator rooted at <node_id>. The resulting traversal will ignore any parents of <node_id>.

#### D.36.2.2 virtual ~BreadthFirstIterator () `[inline, virtual]`

### D.36.3 Member Function Documentation

#### D.36.3.1 Tree< T >::Iterator & operator++ () `[virtual]`

Implements Tree::Iterator.

#### D.36.3.2 bool operator== (const Iterator & *other*) `[inline, inherited]`

Returns true iff <other> points to the same node.

#### D.36.3.3 bool operator!= (const Iterator & *other*) `[inline, inherited]`

Returns true iff <other> points to a different node.

#### D.36.3.4 int operator ∗ () [inline, inherited]

#### D.36.3.5 int id () [inline, inherited]

### D.36.4 Member Data Documentation

#### D.36.4.1 int node_id_ [protected, inherited]

#### D.36.4.2 const Tree<T>∗ source_tree_ [protected, inherited]

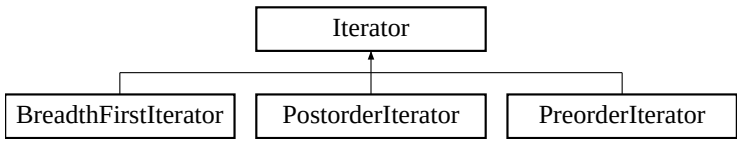
The documentation for this class was generated from the following files:

- tree/tree.h
- tree/tree.cc

## D.37 Tree::Iterator Class Reference

`#include <tree.h>`

Inheritance diagram for Tree::Iterator::




### D.37.1 Detailed Description

**template<class T> class libpmk::Tree< T >::Iterator**

An iterator for Trees.

### Public Member Functions

- Iterator (int node_id, const Tree< T > ∗source_tree)

  *Creates a new iterator pointing at the given ID.*

- virtual ∼Iterator ()
- Iterator & operator= (const Iterator &other)

*Copy assignment operator.*

- bool operator== (const Iterator &other)

  *Returns true iff <other> points to the same node.*

- bool operator!= (const Iterator &other)

  *Returns true iff <other> points to a different node.*

- int operator * ()
- int id ()
- virtual Iterator & operator++ ()=0

**Protected Attributes**

- int node_id_
- const Tree< T > * source_tree_

### D.37.2 Constructor & Destructor Documentation

#### D.37.2.1 Iterator (int *node_id*, const Tree< T > * *source_tree*) [inline]

Creates a new iterator pointing at the given ID.

#### D.37.2.2 virtual ~Iterator () [inline, virtual]

### D.37.3 Member Function Documentation

#### D.37.3.1 Iterator& operator= (const Iterator & *other*) [inline]

Copy assignment operator.

#### D.37.3.2 bool operator== (const Iterator & *other*) [inline]

Returns true iff <other> points to the same node.

#### D.37.3.3 bool operator!= (const Iterator & *other*) [inline]

Returns true iff <other> points to a different node.

#### D.37.3.4 int operator * () [inline]

#### D.37.3.5 int id () [inline]

#### D.37.3.6 virtual Iterator& operator++ () [pure virtual]

Implemented in Tree::PreorderIterator, Tree::PostorderIterator, and Tree::BreadthFirstIterator.

### D.37.4 Member Data Documentation

#### D.37.4.1 int node_id_ [protected]

#### D.37.4.2 const Tree<T>* source_tree_ [protected]

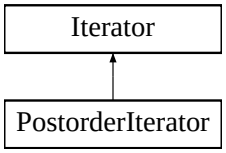
The documentation for this class was generated from the following file:

- tree/tree.h

## D.38 Tree::PostorderIterator Class Reference

`#include <tree.h>`

Inheritance diagram for Tree::PostorderIterator::




**template<class T> class libpmk::Tree< T >::PostorderIterator**

### Public Member Functions

- PostorderIterator (int node_id, const Tree< T > *tree)

  *Creates a new iterator rooted at <node_id>. The resulting traversal will ignore any parents of <node_id>.*

- virtual ~PostorderIterator ()
- virtual Iterator & operator++ ()
- bool operator== (const Iterator &other)

  *Returns true iff <other> points to the same node.*

- bool operator!= (const Iterator &other)

  *Returns true iff <other> points to a different node.*

- int operator ∗ ()
- int id ()

**Protected Attributes**

- int node_id_
- const Tree< T > ∗ source_tree_

### D.38.1 Constructor & Destructor Documentation

#### D.38.1.1 PostorderIterator (int *node_id*, const Tree< T > ∗ *tree*)

Creates a new iterator rooted at <node_id>. The resulting traversal will ignore any parents of <node_id>.

#### D.38.1.2 virtual ∼PostorderIterator () `[inline, virtual]`

### D.38.2 Member Function Documentation

#### D.38.2.1 Tree< T >::Iterator & operator++ () `[virtual]`

Implements Tree::Iterator.

#### D.38.2.2 bool operator== (const Iterator & *other*) `[inline, inherited]`

Returns true iff <other> points to the same node.

#### D.38.2.3 bool operator!= (const Iterator & *other*) `[inline, inherited]`

Returns true iff <other> points to a different node.

#### D.38.2.4 int operator ∗ () `[inline, inherited]`

#### D.38.2.5 int id () `[inline, inherited]`

### D.38.3 Member Data Documentation

#### D.38.3.1 int node_id_ [protected, inherited]

#### D.38.3.2 const Tree<T>* source_tree_ [protected, inherited]

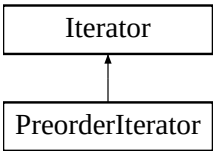
The documentation for this class was generated from the following files:

- tree/tree.h
- tree/tree.cc

## D.39 Tree::PreorderIterator Class Reference

`#include <tree.h>`

Inheritance diagram for Tree::PreorderIterator::




### D.39.1 Detailed Description

**template<class T> class libpmk::Tree< T >::PreorderIterator**

Preorder depth-first iterator for Trees.

### Public Member Functions

- PreorderIterator (int node_id, const Tree< T > *tree)

  *Creates a new iterator rooted at <node_id>. The resulting traversal will ignore any parents of <node_id>.*

- virtual ~PreorderIterator ()
- virtual Iterator & operator++ ()
- bool operator== (const Iterator &other)

  *Returns true iff <other> points to the same node.*

- bool operator!= (const Iterator &other)

*Returns true iff <other> points to a different node.*

- int operator ∗ ()
- int id ()

**Protected Attributes**

- int node_id_
- const Tree< T > ∗ source_tree_

### D.39.2 Constructor & Destructor Documentation

#### D.39.2.1 PreorderIterator (int *node_id*, const Tree< T > ∗ *tree*) [inline]

Creates a new iterator rooted at <node_id>. The resulting traversal will ignore any parents of <node_id>.

#### D.39.2.2 virtual ∼PreorderIterator () [inline, virtual]

### D.39.3 Member Function Documentation

#### D.39.3.1 Tree< T >::Iterator & operator++ () [virtual]

Implements Tree::Iterator.

#### D.39.3.2 bool operator== (const Iterator & *other*) [inline, inherited]

Returns true iff <other> points to the same node.

#### D.39.3.3 bool operator!= (const Iterator & *other*) [inline, inherited]

Returns true iff <other> points to a different node.

#### D.39.3.4 int operator ∗ () [inline, inherited]

#### D.39.3.5 int id () [inline, inherited]

### D.39.4 Member Data Documentation

#### D.39.4.1 int node_id_ [protected, inherited]

#### D.39.4.2 const Tree<T>* source_tree_ [protected, inherited]

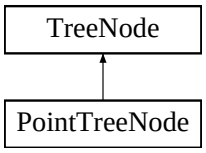
The documentation for this class was generated from the following files:

- tree/tree.h
- tree/tree.cc

## D.40 TreeNode Class Reference

`#include <tree-node.h>`

Inheritance diagram for TreeNode::




### D.40.1 Detailed Description

An indexed node, used by Tree.

A TreeNode has an integer identifier. It also contains the IDs of its parent and children, but makes no effort to maintain them.

To augment a TreeNode with your own data, there are three functions you need to provide: (1) reading the data from a stream, (2) writing the data to a stream, and (3) how to combine two nodes. The definition of "combine" will vary depending on what you're doing, but the general idea is that sometimes you will have two TreeNodes with the same ID that you want to insert into a tree. It is advisable that you make the output of Combine() symmetric, since there are no guarantees on what Tree does with duplicate-ID bins other than calling Combine().

Remember that if you implement a copy constructor for your TreeNode, you should have your copy constructor call the base class's copy constructor:

```
MyNode::MyNode(const MyNode& other) : TreeNode(other) {
// your copy code
}
```

#### Public Member Functions

- TreeNode ()

- TreeNode (int node_id)
- TreeNode (const TreeNode &other)

  *A copy constructor, which copies the ID and parent/child info.*

- virtual ~TreeNode ()
- int id () const
- int parent () const
- int child (int child_index) const
- int child_size () const
- bool has_parent () const
- bool has_child () const
- void set_id (int id)
- void set_parent (int id)
- void add_child (int id)

  *Adds a child to the end of the child list.*

- void remove_child (int child_index)

  *Removes the <child_index>th child.*

- void ReadFromStream (istream &input_stream)

  *Read the data, including the index, from a stream.*

- void WriteToStream (ostream &output_stream) const

  *Write the data, including the index, to a stream.*

- virtual void Combine (const TreeNode &other)=0

  *Add the data from <other> to self.*

### Static Public Attributes

- static const int kInvalidNodeID = -1

### Protected Member Functions

- virtual void ReadData (istream &input_stream)=0

  *Read the data, excluding the index, from a stream.*

- virtual void WriteData (ostream &output_stream) const=0

  *Write the data, excluding the index, to a stream.*

### D.40.2 Constructor & Destructor Documentation

#### D.40.2.1 TreeNode ()

#### D.40.2.2 TreeNode (int *node_id*)

#### D.40.2.3 TreeNode (const TreeNode & *other*)

A copy constructor, which copies the ID and parent/child info.

#### D.40.2.4 virtual ~TreeNode () [inline, virtual]

### D.40.3 Member Function Documentation

#### D.40.3.1 int id () const [inline]

#### D.40.3.2 int parent () const [inline]

#### D.40.3.3 int child (int *child_index*) const [inline]

#### D.40.3.4 int child_size () const [inline]

#### D.40.3.5 bool has_parent () const [inline]

#### D.40.3.6 bool has_child () const [inline]

#### D.40.3.7 void set_id (int *id*) [inline]

#### D.40.3.8 void set_parent (int *id*) [inline]

#### D.40.3.9 void add_child (int *id*) [inline]

Adds a child to the end of the child list.

This does not check for dupes, i.e., it is possible for a node to have two children with the same ID using this, so be careful!

#### D.40.3.10 void remove_child (int *child_index*)

Removes the <child_index>th child.

#### D.40.3.11 void ReadFromStream (istream & *input_stream*)

Read the data, including the index, from a stream.

Calling this will override the node's current index. The format is:

- (int32_t) id
- (int32_t) parent_id
- (int32_t) num_children
- (num_children * int32_t) child IDs

It will then call ReadData().

**See also:**

ReadData()

#### D.40.3.12 void WriteToStream (ostream & *output_stream*) const

Write the data, including the index, to a stream.

This will simply write the index to the stream, followed by WriteData().

**See also:**

WriteData().

#### D.40.3.13 virtual void Combine (const TreeNode & *other*) [pure virtual]

Add the data from <other> to self.

Implemented in PointTreeNode.

#### D.40.3.14 virtual void ReadData (istream & *input_stream*) [protected, pure virtual]

Read the data, excluding the index, from a stream.

Implemented in PointTreeNode.

#### D.40.3.15 virtual void WriteData (ostream & *output_stream*) const [protected, pure virtual]

Write the data, excluding the index, to a stream.

Implemented in PointTreeNode.

### D.40.4 Member Data Documentation

#### D.40.4.1 const int kInvalidNodeID = -1 [static]

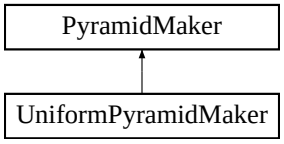
The documentation for this class was generated from the following files:

- tree/tree-node.h
- tree/tree-node.cc

## D.41 UniformPyramidMaker Class Reference

`#include <uniform-pyramid-maker.h>`

Inheritance diagram for UniformPyramidMaker::




### D.41.1 Detailed Description

Creates MultiResolutionHistograms with bins of uniform size at each level.

The `discretize_factor` parameter, passed into Preprocess(), is particularly important because it can greatly affect the shape of the resulting pyramid. This parameter multiplies all of the feature values by 10^{discretize_factor}. We typically set it in such a way that the resulting features are on the order of 10^2.

It is also important to note that uniform-bin pyramids are useful only when dealing with low-dimensional data (up to about 10-dimensional). When dealing with higher-dimensional features, consider using VGPyramidMaker instead.

### Public Member Functions

- UniformPyramidMaker ()
- void Preprocess (const PointSetList &point_sets, double finest_side_length, double side_factor, int discretize_factor, bool do_translations, bool global_translations)

  *Set parameters and preprocess the pyramid settings.*

- void Preprocess (const vector< PointRef > &points, double finest_side_length, double side_factor, int discretize_factor, bool do_translations, bool global_translations)

  *Set parameters and preprocess the pyramid settings.*

- void ReadFromStream (istream &input_stream)

  *Read parameters and preprocessed data from a stream.*

- void ReadFromFile (const char *filename)

  *Read parameters and preprocessed data from a file.*

- void WriteToStream (ostream &output_stream) const

  *Write parameters and preprocessed data to a stream.*

- void WriteToFile (const char *filename) const

  *Write parameters and preprocessed data to a file.*

- int num_levels () const

  *Get the number of levels in the pyramids that will be created.*

- virtual MultiResolutionHistogram * MakePyramid (const PointSet &ps)

  *Creates a uniform pyramid from a PointSet.*

- virtual MultiResolutionHistogram * MakePyramid (const vector< PointRef > &points)
- vector< MultiResolutionHistogram * > MakePyramids (const PointSetList &psl)

  *Turn a list of PointSets into a bunch of MultiResolutionHistograms.*

### D.41.2 Constructor & Destructor Documentation

#### D.41.2.1 UniformPyramidMaker ()

### D.41.3 Member Function Documentation

#### D.41.3.1 void Preprocess (const PointSetList & *point_sets*, double *finest_side_length*, double *side_factor*, int *discretize_factor*, bool *do_translations*, bool *global_translations*)

Set parameters and preprocess the pyramid settings.

**Parameters:**

***finest_side_length*** The length of a side of the smallest bin at the lowest level of the pyramid.

***side_factor*** The length of the side of the bin is this factor larger for bins one level above.

***discretize_factor*** Scales all of the features by a factor 10^{discretize_factor}.

***do_translations*** Determines whether to apply random shifts to the bins.

***global_translations*** Determines whether the shift should be the same across all levels or not.

Rather than preprocessing, ReadFromStream() may be called in lieu of this.

#### D.41.3.2 void Preprocess (const vector< PointRef > & *points*, double *finest_side_length*, double *side_factor*, int *discretize_factor*, bool *do_translations*, bool *global_translations*)

Set parameters and preprocess the pyramid settings.

#### D.41.3.3 void ReadFromStream (istream & *input_stream*)

Read parameters and preprocessed data from a stream.

Aborts if the stream is bad.

#### D.41.3.4 void ReadFromFile (const char * *filename*)

Read parameters and preprocessed data from a file.

**See also:**

ReadFromStream

#### D.41.3.5 void WriteToStream (ostream & *output_stream*) const

Write parameters and preprocessed data to a stream.

Aborts if the stream is bad.

#### D.41.3.6 void WriteToFile (const char * *filename*) const

Write parameters and preprocessed data to a file.

**See also:**

WriteToStream

#### D.41.3.7 int num_levels () const

Get the number of levels in the pyramids that will be created.

Must be called after Preprocess() or ReadFromStream().

#### D.41.3.8 MultiResolutionHistogram * MakePyramid (const PointSet & *ps*) [virtual]

Creates a uniform pyramid from a PointSet.

Requires Preprocess() or ReadFromStream() to be called first. This function allocates memory on its own; it is up to the caller to free the memory. Note that the input point set should not be on the same scale as the point sets that the UniformPyramidMaker was constructed with. If you specified a nonzero `discretize_factor` in the constructor, the PointSet <ps> will automatically be scaled according to that. Additionally, any translations (if you told the constructor to do so) will also be applied automatically– you should not have to shift or scale anything in <ps> before calling MakePyramid() on it.

Implements PyramidMaker.

#### D.41.3.9 MultiResolutionHistogram * MakePyramid (const vector< PointRef > & *points*) [virtual]

#### D.41.3.10 vector< MultiResolutionHistogram * > MakePyramids (const PointSetList & *psl*) [inherited]

Turn a list of PointSets into a bunch of MultiResolutionHistograms.

It is up to the caller to free the memory.

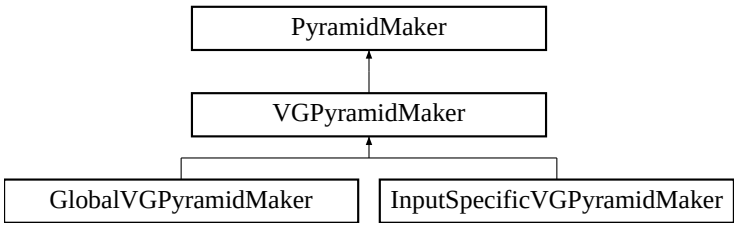
The documentation for this class was generated from the following files:

- pyramids/uniform-pyramid-maker.h
- pyramids/uniform-pyramid-maker.cc

## D.42 VGPyramidMaker Class Reference

`#include <vg-pyramid-maker.h>`

Inheritance diagram for VGPyramidMaker::




### D.42.1 Detailed Description

Abstract interface for vocabulary-guided pyramid makers.

The vocabulary-guided pyramid makers here use the output of a hierarchical clustering algorithm to generate pyramids.

#### Public Member Functions

- VGPyramidMaker (const HierarchicalClusterer &c, const DistanceComputer &distance_computer)
- virtual MultiResolutionHistogram * MakePyramid (const PointSet &point_set)=0

  *Turn a single PointSet into a MultiResolutionHistogram.*

- vector< MultiResolutionHistogram * > MakePyramids (const PointSetList &psl)

  *Turn a list of PointSets into a bunch of MultiResolutionHistograms.*

#### Protected Member Functions

- bool GetMembershipPath (const Point &f, LargeIndex *out_path, vector< double > *out_distances)

  *For a new feature, find which bin it would be placed in according to the HierarchicalClusterer.*

**Protected Attributes**

- const HierarchicalClusterer & clusterer_
- const Tree< PointTreeNode > & centers_

  *The centers extracted from clusterer_.*

- const DistanceComputer & distance_computer_

### D.42.2 Constructor & Destructor Documentation

#### D.42.2.1 VGPyramidMaker (const HierarchicalClusterer & *c*, const DistanceComputer & *distance_computer*)

Requires a HierarchicalCluster which has already been Preprocess()'d (or ReadFromStream()), and a DistanceComputer which computes the same distances that were used to generate the HierarchicalClusterer's data.

### D.42.3 Member Function Documentation

#### D.42.3.1 virtual MultiResolutionHistogram* MakePyramid (const PointSet & *point_set*) [pure virtual]

Turn a single PointSet into a MultiResolutionHistogram.

This function allocates memory on its own. It is up to the caller to free it.

Implements PyramidMaker.

Implemented in GlobalVGPyramidMaker, and InputSpecificVGPyramidMaker.

#### D.42.3.2 bool GetMembershipPath (const Point & *f*, LargeIndex * *out_path*, vector< double > * *out_distances*) [protected]

For a new feature, find which bin it would be placed in according to the HierarchicalClusterer.

The LargeIndex returned is the same size as the number of levels in the tree. Each element of the LargeIndex tells you which child index to traverse. The first element is 0 by default, since the root of the tree has no parent. For instance, if the returned LargeIndex is [0 3 9], then the path down the tree is: root R, 3rd child of R, followed by 9th child of that node.

The returned vector of doubles gives the distance to the corresponding center at each level, according to distance_computer_. Thus, the first element is the distance from the point <f> to the center of the root node, etc.

The pointers must not be NULL.

Returns true on success, which is always.

Reimplemented in GlobalVGPyramidMaker.

#### D.42.3.3 vector< MultiResolutionHistogram * > MakePyramids (const PointSetList & *psl*) [inherited]

Turn a list of PointSets into a bunch of MultiResolutionHistograms.

It is up to the caller to free the memory.

### D.42.4 Member Data Documentation

#### D.42.4.1 const HierarchicalClusterer& clusterer_ [protected]

#### D.42.4.2 const Tree<PointTreeNode>& centers_ [protected]

The centers extracted from clusterer_.

#### D.42.4.3 const DistanceComputer& distance_computer_ [protected]

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

- pyramids/vg-pyramid-maker.h
- pyramids/vg-pyramid-maker.cc

# E LIBPMK File Documentation

## E.1 clustering/clusterer.cc File Reference

```
#include <assert.h>
#include <vector>
#include <iostream>
#include <fstream>
#include "clustering/clusterer.h"
#include "point_set/point-set.h"
#include "point_set/point-ref.h"
```

### Namespaces

- namespace libpmk

- namespace std

## E.2 clustering/clusterer.h File Reference

```
#include <vector>
#include <iostream>
#include "point_set/point-set.h"
#include "point_set/point-ref.h"
```

### Namespaces

- namespace libpmk

### Classes

- class Clusterer

  *Abstract interface for a flat clusterer.*

## E.3 clustering/hierarchical-clusterer.cc File Reference

```
#include <iostream>
#include <vector>
#include <fstream>
#include "clustering/hierarchical-clusterer.h"
#include "clustering/k-means-clusterer.h"
#include "point_set/point-ref.h"
#include "util/sparse-vector.h"
#include "util/distance-computer.h"
#include "tree/point-tree-node.h"
#include "tree/tree.cc"
```

### Namespaces

- namespace libpmk

## E.4 clustering/hierarchical-clusterer.h File Reference

```
#include <vector>
#include <list>
#include <iostream>
#include "clustering/clusterer.h"
#include "point_set/point-ref.h"
#include "tree/point-tree-node.h"
#include "tree/tree.cc"
#include "util/distance-computer.h"
```

### Namespaces

- namespace libpmk

### Classes

- class HierarchicalClusterer

  *Hierarchical K-Means clusterer.*

## E.5 clustering/k-means-clusterer.cc File Reference

```
#include <vector>
#include "clustering/k-means-clusterer.h"
#include "point_set/point-ref.h"
#include "point_set/point-set.h"
#include "util/distance-computer.h"
```

### Namespaces

- namespace libpmk

## E.6 clustering/k-means-clusterer.h File Reference

```
#include <vector>
#include "clustering/clusterer.h"
#include "point_set/point-ref.h"
#include "util/distance-computer.h"
```

### Namespaces

- namespace libpmk

### Classes

- class KMeansClusterer

  *Implements K-Means clustering.*

## E.7 experiment/example-selector.cc File Reference

```
#include <algorithm>
#include <vector>
#include "experiment/example-selector.h"
```

## E.8 experiment/example-selector.h File Reference

```
#include <vector>
#include "util/labeled-index.h"
```

### Namespaces

- namespace libpmk_util

### Classes

- class ExampleSelector

  *Splits a list of labels into a training and test set.*

## E.9 experiment/experiment.cc File Reference

```
#include <assert.h>
#include <vector>
#include "experiment/experiment.h"
#include "util/labeled-index.h"
#include "kernel/kernel-matrix.h"
```

## E.10 experiment/experiment.h File Reference

```
#include <vector>
#include "kernel/matrix.h"
#include "kernel/kernel-matrix.h"
#include "util/labeled-index.h"
```

### Namespaces

- namespace libpmk_util

### Classes

- class Experiment

  *Encapsulates training/testing of any method involving a kernel.*

## E.11 experiment/random-selector.cc File Reference

```
#include <assert.h>
#include "experiment/random-selector.h"
```

## E.12 experiment/random-selector.h File Reference

```
#include "experiment/example-selector.h"
```

### Namespaces

- namespace libpmk_util

### Classes

- class RandomSelector

  *Chooses a random sample of each class to use as testing.*

## E.13 experiment/svm-experiment.cc File Reference

```
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <assert.h>
#include <string>
#include <sstream>
#include "experiment/svm-experiment.h"
#include "experiment/experiment.h"
#include "kernel/kernel-matrix.h"
#include "util/labeled-index.h"
#include "svm/svm.h"
```

### Namespaces

- namespace libpmk_util

## E.14 experiment/svm-experiment.h File Reference

```
#include "experiment/experiment.h"
#include "kernel/kernel-matrix.h"
#include "util/labeled-index.h"
#include "svm/svm.h"
```

### Namespaces

- namespace libpmk_util

### Classes

- class SVMExperiment

  *Runs an experiment using LIBSVM.*

## E.15 histograms/bin.cc File Reference

```
#include "histograms/bin.h"
#include "util/sparse-vector.h"
```

### Namespaces

- namespace libpmk

## E.16 histograms/bin.h File Reference

```
#include <iostream>
#include "util/sparse-vector.h"
#include "tree/sparse-tree-node.h"
```

### Namespaces

- namespace libpmk

### Classes

- class Bin

  *Encapsulates a histogram bin.*

## E.17 histograms/histogram.cc File Reference

```
#include <assert.h>
#include <iostream>
#include "histograms/bin.h"
#include "histograms/histogram.h"
```

### Namespaces

- namespace libpmk

## E.18 histograms/histogram.h File Reference

```
#include <string>
#include <iostream>
#include "histograms/bin.h"
```

### Namespaces

- namespace libpmk

### Classes

- class Histogram

  *A sparse representation for a flat (1-D) histogram.*

## E.19 histograms/multi-resolution-histogram.cc File Reference

```
#include <assert.h>
#include <string>
#include <iostream>
#include <fstream>
#include "histograms/multi-resolution-histogram.h"
#include "histograms/bin.h"
```

```
#include "util/sparse-vector.h"
#include "tree/sparse-tree.cc"
```

### Namespaces

- namespace libpmk

## E.20 histograms/multi-resolution-histogram.h File Reference

```
#include <iostream>
#include "histograms/bin.h"
#include "tree/sparse-tree.cc"
```

### Namespaces

- namespace libpmk

### Classes

- class MultiResolutionHistogram

  *A data structure for a pyramid of histograms, with a link structure between levels.*

## E.21 kernel/kernel-matrix.cc File Reference

```
#include <math.h>
#include <assert.h>
#include <iostream>
#include <fstream>
#include "kernel/kernel-matrix.h"
```

### Namespaces

- namespace libpmk_util

## E.22 kernel/kernel-matrix.h File Reference

```
#include <vector>
#include <string>
#include <iostream>
```

### Namespaces

- namespace libpmk_util

### Classes

- class KernelMatrix

  *Data structure for a square symmetric matrix.*

## E.23 kernel/matrix.cc File Reference

```
#include <math.h>
#include <assert.h>
#include <iostream>
#include <fstream>
#include "kernel/matrix.h"
```

### Namespaces

- namespace libpmk_util

## E.24 kernel/matrix.h File Reference

```
#include <vector>
#include <string>
#include <iostream>
```

### Namespaces

- namespace libpmk_util

### Classes

- class Matrix

  *Data structure for a matrix of doubles.*

## E.25 point_set/mutable-point-set-list.cc File Reference

```
#include <assert.h>
#include <iostream>
#include <fstream>
#include "point_set/mutable-point-set-list.h"
#include "point_set/point-set.h"
#include "point_set/point.h"
```

### Namespaces

- namespace libpmk

## E.26 point_set/mutable-point-set-list.h File Reference

```
#include <vector>
#include <iostream>
#include "point_set/point-set-list.h"
#include "point_set/point-set.h"
#include "point_set/point.h"
```

### Namespaces

- namespace libpmk

### Classes

- class MutablePointSetList

  *A mutable, in-memory PointSetList.*

## E.27 point_set/on-disk-point-set-list.cc File Reference

```
#include <assert.h>
#include <string>
#include <iostream>
#include <list>
#include "point_set/on-disk-point-set-list.h"
#include "point_set/point-set.h"
```

### Namespaces

- namespace libpmk

## E.28 point_set/on-disk-point-set-list.h File Reference

```
#include <vector>
#include <string>
#include <iostream>
#include <fstream>
#include <list>
#include "point_set/point-set-list.h"
#include "point_set/point-set.h"
```

### Namespaces

- namespace libpmk

### Classes

- class OnDiskPointSetList

*A read-only PointSetList which reads information from disk.*

## E.29 point_set/point-ref.cc File Reference

```
#include <assert.h>
#include "point_set/point-ref.h"
#include "point_set/point-set-list.h"
#include "point_set/point.h"
```

### Namespaces

- namespace libpmk

## E.30 point_set/point-ref.h File Reference

```
#include "point_set/point-set-list.h"
#include "point_set/point.h"
```

### Namespaces

- namespace libpmk

### Classes

- class PointRef

  *A way to reference Points in a particular PointSetList.*

## E.31 point_set/point-set-list.cc File Reference

```
#include <assert.h>
#include <string>
#include <iostream>
#include <fstream>
#include "point_set/point-set-list.h"
```

```
#include "point_set/point-set.h"
#include "point_set/point-ref.h"
#include "point_set/point.h"
```

#### Namespaces

- namespace libpmk

## E.32 point_set/point-set-list.h File Reference

```
#include <assert.h>
#include <vector>
#include <string>
#include <iostream>
#include "point_set/point-set.h"
#include "point_set/point-ref.h"
#include "point_set/point.h"
```

#### Namespaces

- namespace libpmk

#### Classes

- class PointSetList

  *Abstract interface for a list of PointSet.*

## E.33 point_set/point-set.cc File Reference

```
#include <assert.h>
#include <iostream>
#include "point_set/point-set.h"
#include "point_set/point.h"
```

### Namespaces

- namespace libpmk

## E.34 point_set/point-set.h File Reference

```
#include <vector>
#include "point_set/point.h"
```

### Namespaces

- namespace libpmk

### Classes

- class PointSet

  *A data structure representing a list of Points.*

## E.35 point_set/point.cc File Reference

```
#include <iostream>
#include "point_set/point.h"
```

### Namespaces

- namespace libpmk

## E.36 point_set/point.h File Reference

```
#include <iostream>
#include <vector>
```

### Namespaces

- namespace libpmk

### Classes

- class Point

  *A generic abstract data structure storing a weighted vector of floats.*

## E.37 pyramids/global-vg-pyramid-maker.cc File Reference

```
#include <values.h>
#include <vector>
#include <fstream>
#include <ext/hash_set>
#include "pyramids/global-vg-pyramid-maker.h"
#include "clustering/hierarchical-clusterer.h"
#include "histograms/multi-resolution-histogram.h"
#include "point_set/point-set-list.h"
#include "util/distance-computer.h"
#include "util/sparse-vector.h"
```

### Namespaces

- namespace libpmk
- namespace __gnu_cxx

## E.38 pyramids/global-vg-pyramid-maker.h File Reference

```
#include <vector>
#include <iostream>
#include <set>
#include <ext/hash_map>
#include "clustering/hierarchical-clusterer.h"
#include "histograms/multi-resolution-histogram.h"
#include "point_set/point-set-list.h"
#include "pyramids/vg-pyramid-maker.h"
```

```
#include "tree/sparse-tree-node.h"
#include "tree/sparse-tree.cc"
#include "util/distance-computer.h"
#include "util/sparse-vector.h"
```

### Namespaces

- namespace libpmk

### Classes

- class GlobalVGPyramidMaker

  *Makes pyramids with bin sizes determined by a particular set of points.*

## E.39 pyramids/input-specific-vg-pyramid-maker.cc File Reference

```
#include <vector>
#include "pyramids/input-specific-vg-pyramid-maker.h"
#include "clustering/hierarchical-clusterer.h"
#include "histograms/multi-resolution-histogram.h"
#include "point_set/point-set.h"
#include "util/distance-computer.h"
#include "util/sparse-vector.h"
```

### Namespaces

- namespace libpmk

## E.40 pyramids/input-specific-vg-pyramid-maker.h File Reference

```
#include "clustering/hierarchical-clusterer.h"
#include "histograms/multi-resolution-histogram.h"
#include "point_set/point-set.h"
#include "pyramids/vg-pyramid-maker.h"
```

```
#include "util/distance-computer.h"
```

### Namespaces

- namespace libpmk

### Classes

- class InputSpecificVGPyramidMaker

  *Makes pyramids with bin sizes that are specific to each input.*

## E.41 pyramids/normalized-uniform-pyramid-maker.cc File Reference

```
#include <assert.h>
#include <math.h>
#include <stdlib.h>
#include <values.h>
#include <iostream>
#include <fstream>
#include "pyramids/normalized-uniform-pyramid-maker.h"
#include "util/sparse-vector.h"
```

### Namespaces

- namespace libpmk

## E.42 pyramids/normalized-uniform-pyramid-maker.h File Reference

```
#include <vector>
#include <map>
#include <iostream>
#include "pyramids/pyramid-maker.h"
#include "point_set/point-set-list.h"
#include "point_set/point-set.h"
```

```
#include "histograms/multi-resolution-histogram.h"
```

### Namespaces

- namespace libpmk

### Classes

- class NormalizedUniformPyramidMaker

  *Creates MultiResolutionHistograms with bins of uniform size at each level.*

## E.43 pyramids/pyramid-maker.cc File Reference

```
#include <vector>
#include "pyramids/pyramid-maker.h"
```

### Namespaces

- namespace libpmk

## E.44 pyramids/pyramid-maker.h File Reference

```
#include <vector>
#include "point_set/point-set.h"
#include "point_set/point-set-list.h"
#include "histograms/multi-resolution-histogram.h"
```

### Namespaces

- namespace libpmk

### Classes

- class PyramidMaker

  *Abstract interface for turning PointSets into MultiResolutionHistograms.*

## E.45 pyramids/pyramid-matcher.cc File Reference

```
#include <assert.h>
#include <math.h>
#include <memory>
#include "util/bin-weight-scheme.h"
#include "pyramids/pyramid-matcher.h"
#include "histograms/multi-resolution-histogram.h"
```

### Namespaces

- namespace libpmk

## E.46 pyramids/pyramid-matcher.h File Reference

```
#include "histograms/bin.h"
#include "histograms/multi-resolution-histogram.h"
#include "util/bin-weight-scheme.h"
```

### Namespaces

- namespace libpmk

### Classes

- class PyramidMatcher

  *Matches two MultiResolutionHistograms.*

- class **PyramidMatcher::MatchNode**

## E.47 pyramids/uniform-pyramid-maker.cc File Reference

```
#include <assert.h>
#include <math.h>
#include <stdlib.h>
```

```
#include <values.h>
#include <iostream>
#include <fstream>
#include "pyramids/uniform-pyramid-maker.h"
#include "util/sparse-vector.h"
```

#### Namespaces

- namespace libpmk

## E.48 pyramids/uniform-pyramid-maker.h File Reference

```
#include <vector>
#include <map>
#include <iostream>
#include "pyramids/pyramid-maker.h"
#include "point_set/point-set-list.h"
#include "point_set/point-set.h"
#include "histograms/multi-resolution-histogram.h"
```

#### Namespaces

- namespace libpmk

#### Classes

- class UniformPyramidMaker

  *Creates MultiResolutionHistograms with bins of uniform size at each level.*

## E.49 pyramids/vg-pyramid-maker.cc File Reference

```
#include <assert.h>
#include <math.h>
#include <stdlib.h>
```

```
#include <values.h>
#include "pyramids/vg-pyramid-maker.h"
#include "histograms/multi-resolution-histogram.h"
#include "point_set/point.h"
#include "point_set/point-set.h"
#include "util/distance-computer.h"
#include "util/sparse-vector.h"
```

### Namespaces

- namespace libpmk

## E.50 pyramids/vg-pyramid-maker.h File Reference

```
#include "clustering/hierarchical-clusterer.h"
#include "histograms/multi-resolution-histogram.h"
#include "point_set/point.h"
#include "point_set/point-set.h"
#include "point_set/point-set-list.h"
#include "pyramids/pyramid-maker.h"
#include "util/distance-computer.h"
#include "util/sparse-vector.h"
```

### Namespaces

- namespace libpmk

### Classes

- class VGPyramidMaker

  *Abstract interface for vocabulary-guided pyramid makers.*

## E.51 tools/average-exponent-kernel.cc File Reference

```
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <string>
#include <fstream>
#include "kernel/kernel-matrix.h"
```

### Functions

- int main (int argc, char **argv)

### E.51.1 Function Documentation

#### E.51.1.1 int main (int *argc*, char ** *argv*)

## E.52 tools/caltech101-splitter.cc File Reference

```
#include <assert.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include "point_set/on-disk-point-set-list.h"
#include "point_set/mutable-point-set-list.h"
```

### Functions

- void usage (const char *argv)
- int main (int argc, char **argv)

### E.52.1 Function Documentation

#### E.52.1.1 void usage (const char * *argv*)

#### E.52.1.2 int main (int *argc*, char ** *argv*)

## E.53 tools/clean-psl.cc File Reference

```
#include <math.h>
#include <iostream>
#include <set>
#include <vector>
#include "point_set/point-set.h"
#include "point_set/mutable-point-set-list.h"
```

### Functions

- void usage (const char *exec)
- int main (int argc, char **argv)

### E.53.1 Function Documentation

#### E.53.1.1 void usage (const char * *exec*)

#### E.53.1.2 int main (int *argc*, char ** *argv*)

## E.54 tools/clusters-to-pyramids.cc File Reference

```
#include <iostream>
#include <fstream>
#include "pyramids/input-specific-vg-pyramid-maker.h"
#include "pyramids/global-vg-pyramid-maker.h"
#include "clustering/hierarchical-clusterer.h"
#include "histograms/multi-resolution-histogram.h"
#include "util/distance-computer.h"
#include "util/bin-weight-scheme.h"
#include "point_set/mutable-point-set-list.h"
#include "point_set/on-disk-point-set-list.h"
```

### Functions

- void usage (const char *exec_name)
- int main (int argc, char **argv)

#### E.54.1 Function Documentation

##### E.54.1.1 void usage (const char * *exec_name*)

##### E.54.1.2 int main (int *argc*, char ** *argv*)

## E.55 tools/dbg-disk-ptsets.cc File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <iostream>
#include <sstream>
#include <fstream>
#include "histograms/multi-resolution-histogram.h"
#include "pyramids/pyramid-matcher.h"
#include "pyramids/uniform-pyramid-maker.h"
#include "kernel/kernel-matrix.h"
#include "point_set/on-disk-point-set-list.h"
#include "pyramids/global-vg-pyramid-maker.h"
#include "clustering/hierarchical-clusterer.h"
#include "util/distance-computer.h"
#include "util/bin-weight-scheme.h"
```

### Functions

- void usage (const char *exec_name)
- int main (int argc, char **argv)

### E.55.1 Function Documentation

#### E.55.1.1 void usage (const char * *exec_name*)

#### E.55.1.2 int main (int *argc*, char ** *argv*)

## E.56 tools/distance-kernel.cc File Reference

```
#include <assert.h>
#include <stdio.h>
#include <string>
#include <fstream>
#include "kernel/kernel-matrix.h"
```

### Functions

- int main (int argc, char **argv)

### E.56.1 Function Documentation

#### E.56.1.1 int main (int *argc*, char ** *argv*)

## E.57 tools/exponent-kernel.cc File Reference

```
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <string>
#include <fstream>
#include "kernel/kernel-matrix.h"
```

### Functions

- int main (int argc, char **argv)

### E.57.1 Function Documentation

#### E.57.1.1 int main (int *argc*, char ** *argv*)

## E.58 tools/exponential-pyramid-match-kernel.cc File Reference

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include "histograms/multi-resolution-histogram.h"
#include "pyramids/pyramid-matcher.h"
#include "kernel/kernel-matrix.h"
#include "util/bin-weight-scheme.h"
```

### Functions

- void usage (const char *exec_name)
- int main (int argc, char **argv)

### E.58.1 Function Documentation

#### E.58.1.1 void usage (const char * *exec_name*)

#### E.58.1.2 int main (int *argc*, char ** *argv*)

## E.59 tools/extract-classes-from-psl.cc File Reference

```
#include <math.h>
#include <iostream>
#include <set>
#include <vector>
#include "point_set/point-set.h"
#include "point_set/mutable-point-set-list.h"
```

### Functions

- void usage (const char *exec)
- vector< int > ReadLabels (string filename)
- int main (int argc, char **argv)

#### E.59.1 Function Documentation

##### E.59.1.1 void usage (const char * *exec*)

##### E.59.1.2 vector<int> ReadLabels (string *filename*)

##### E.59.1.3 int main (int *argc*, char ** *argv*)

## E.60 tools/extract-subset-from-psl.cc File Reference

```
#include <math.h>
#include <iostream>
#include <set>
#include <vector>
#include "point_set/point-set.h"
#include "point_set/mutable-point-set-list.h"
```

### Functions

- void usage (const char *exec)
- vector< int > ReadIndices (string filename)
- int main (int argc, char **argv)

#### E.60.1 Function Documentation

##### E.60.1.1 void usage (const char * *exec*)

##### E.60.1.2 vector<int> ReadIndices (string *filename*)

##### E.60.1.3 int main (int *argc*, char ** *argv*)

## E.61 tools/feature-value-discretizer.cc File Reference

```
#include <assert.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include "point_set/mutable-point-set-list.h"
```

### Functions

- void usage (const char *argv)
- int main (int argc, char **argv)

### E.61.1 Function Documentation

#### E.61.1.1 void usage (const char * *argv*)

#### E.61.1.2 int main (int *argc*, char ** *argv*)

## E.62 tools/hierarchical-cluster-point-set.cc File Reference

```
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <fstream>
#include "point_set/mutable-point-set-list.h"
#include "point_set/on-disk-point-set-list.h"
#include "point_set/point-ref.h"
#include "clustering/hierarchical-clusterer.h"
#include "util/distance-computer.h"
```

### Functions

- void usage (const char *exec_name)
- int main (int argc, char **argv)

### E.62.1 Function Documentation

#### E.62.1.1 void usage (const char * *exec_name*)

#### E.62.1.2 int main (int *argc*, char ** *argv*)

## E.63 tools/hierarchical-histogram-maker.cc File Reference

```
#include <stdio.h>
#include <fstream>
#include <map>
#include <set>
#include <string>
#include <vector>
#include "util/sparse-vector.h"
#include "point_set/point-ref.h"
#include "point_set/mutable-point-set-list.h"
#include "clustering/hierarchical-clusterer.h"
```

### Functions

- void print_vector (const vector< int > &vec)
- int main (int argc, char **argv)

### E.63.1 Function Documentation

#### E.63.1.1 void print_vector (const vector< int > & *vec*)

#### E.63.1.2 int main (int *argc*, char ** *argv*)

## E.64 tools/kernel-adder.cc File Reference

```
#include <stdio.h>
#include <string>
```

```
#include <fstream>
#include "kernel/kernel-matrix.h"
```

**Functions**

- int main (int argc, char **argv)

### E.64.1 Function Documentation

#### E.64.1.1 int main (int *argc*, char ** *argv*)

## E.65 tools/kernel-weighter.cc File Reference

```
#include <assert.h>
#include <stdio.h>
#include <string>
#include <fstream>
#include "kernel/kernel-matrix.h"
```

**Functions**

- int main (int argc, char **argv)

### E.65.1 Function Documentation

#### E.65.1.1 int main (int *argc*, char ** *argv*)

## E.66 tools/large-upmk-example.cc File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <iostream>
#include <sstream>
```

```
#include <fstream>
#include "histograms/multi-resolution-histogram.h"
#include "pyramids/pyramid-matcher.h"
#include "pyramids/uniform-pyramid-maker.h"
#include "kernel/kernel-matrix.h"
#include "point_set/on-disk-point-set-list.h"
```

**Functions**

- void usage (const char *exec_name)
- string GetPyramidFilename (int ii)
- int main (int argc, char **argv)

### E.66.1 Function Documentation

#### E.66.1.1 void usage (const char * *exec_name*)

#### E.66.1.2 string GetPyramidFilename (int *ii*)

#### E.66.1.3 int main (int *argc*, char ** *argv*)

## E.67 tools/normalize-kernel.cc File Reference

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include "kernel/kernel-matrix.h"
```

**Functions**

- void usage (const char *exec_name)
- int main (int argc, char **argv)

### E.67.1 Function Documentation

#### E.67.1.1 void usage (const char * *exec_name*)

#### E.67.1.2 int main (int *argc*, char ** *argv*)

## E.68 tools/partial-pyramid-match-kernel.cc File Reference

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include "histograms/multi-resolution-histogram.h"
#include "pyramids/pyramid-matcher.h"
#include "kernel/kernel-matrix.h"
#include "util/bin-weight-scheme.h"
```

### Functions

- void usage (const char *exec_name)
- int main (int argc, char **argv)

### E.68.1 Function Documentation

#### E.68.1.1 void usage (const char * *exec_name*)

#### E.68.1.2 int main (int *argc*, char ** *argv*)

## E.69 tools/pointset-merger.cc File Reference

```
#include <assert.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include "point_set/mutable-point-set-list.h"
```

### Functions

- void usage (const char *argv)
- int main (int argc, char **argv)

#### E.69.1 Function Documentation

##### E.69.1.1 void usage (const char * *argv*)

##### E.69.1.2 int main (int *argc*, char ** *argv*)

## E.70 tools/pointsets-to-uniform-pyramids.cc File Reference

```
#include <math.h>
#include <assert.h>
#include <stdlib.h>
#include <time.h>
#include <iostream>
#include <fstream>
#include "histograms/bin.h"
#include "pyramids/uniform-pyramid-maker.h"
#include "point_set/on-disk-point-set-list.h"
```

### Functions

- void usage (const char *exec_name)
- int main (int argc, char **argv)

#### E.70.1 Function Documentation

##### E.70.1.1 void usage (const char * *exec_name*)

##### E.70.1.2 int main (int *argc*, char ** *argv*)

## E.71 tools/psl-sampler.cc File Reference

```
#include <math.h>
#include <iostream>
#include <set>
#include <vector>
#include "point_set/point-set.h"
#include "point_set/mutable-point-set-list.h"
```

### Functions

- void usage (const char *exec)
- int main (int argc, char **argv)

#### E.71.1 Function Documentation

##### E.71.1.1 void usage (const char * *exec*)

##### E.71.1.2 int main (int *argc*, char ** *argv*)

## E.72 tools/pyramid-match-kernel.cc File Reference

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include "histograms/multi-resolution-histogram.h"
#include "pyramids/pyramid-matcher.h"
#include "kernel/kernel-matrix.h"
#include "util/bin-weight-scheme.h"
```

### Functions

- void usage (const char *exec_name)
- int main (int argc, char **argv)

### E.72.1 Function Documentation

#### E.72.1.1 void usage (const char * *exec_name*)

#### E.72.1.2 int main (int *argc*, char ** *argv*)

## E.73 tools/pyramid-match-self-similarity.cc File Reference

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include "histograms/multi-resolution-histogram.h"
#include "pyramids/pyramid-matcher.h"
#include "kernel/kernel-matrix.h"
#include "util/bin-weight-scheme.h"
```

### Functions

- void usage (const char *exec_name)
- int main (int argc, char **argv)

### E.73.1 Function Documentation

#### E.73.1.1 void usage (const char * *exec_name*)

#### E.73.1.2 int main (int *argc*, char ** *argv*)

## E.74 tools/pyramid-match-split.cc File Reference

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <fstream>
```

```
#include "histograms/multi-resolution-histogram.h"
#include "pyramids/pyramid-matcher.h"
#include "kernel/matrix.h"
#include "util/bin-weight-scheme.h"
```

**Functions**

- void usage (const char *exec_name)
- int main (int argc, char **argv)

### E.74.1 Function Documentation

#### E.74.1.1 void usage (const char * *exec_name*)

#### E.74.1.2 int main (int *argc*, char ** *argv*)

## E.75 tools/random-sampler.cc File Reference

```
#include <assert.h>
#include <math.h>
#include <iostream>
#include <fstream>
#include "point_set/mutable-point-set-list.h"
```

**Functions**

- void usage (const char *exec_name)
- int main (int argc, char **argv)

### E.75.1 Function Documentation

#### E.75.1.1 void usage (const char * *exec_name*)

#### E.75.1.2 int main (int *argc*, char ** *argv*)

## E.76 tools/random-sampling-merger.cc File Reference

```
#include <assert.h>
#include <math.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include "point_set/mutable-point-set-list.h"
```

### Functions

- void usage (const char *argv)
- int main (int argc, char **argv)

### E.76.1 Function Documentation

#### E.76.1.1 void usage (const char * *argv*)

#### E.76.1.2 int main (int *argc*, char ** *argv*)

## E.77 tree/point-tree-node.cc File Reference

```
#include <assert.h>
#include <iostream>
#include "tree/point-tree-node.h"
#include "point_set/point.h"
```

### Namespaces

- namespace libpmk

## E.78 tree/point-tree-node.h File Reference

```
#include <iostream>
#include "tree/tree-node.h"
```

```
#include "point_set/point.h"
```

### Namespaces

- namespace libpmk

### Classes

- class PointTreeNode

  *A TreeNode containing a Point.*

## E.79 tree/sparse-tree-node.cc File Reference

```
#include <iostream>
#include "tree/sparse-tree-node.h"
```

### Namespaces

- namespace libpmk

## E.80 tree/sparse-tree-node.h File Reference

```
#include <iostream>
#include "util/sparse-vector.h"
```

### Namespaces

- namespace libpmk

### Classes

- class SparseTreeNode

  *An indexed node, used by SparseTree.*

## E.81 tree/sparse-tree.cc File Reference

```
#include <assert.h>
#include <list>
#include <stack>
#include "tree/sparse-tree.h"
#include "tree/sparse-tree-node.h"
#include "util/sparse-vector.h"
```

### Namespaces

- namespace libpmk

## E.82 tree/sparse-tree.h File Reference

```
#include <list>
#include <stack>
#include "tree/sparse-tree-node.h"
```

### Namespaces

- namespace libpmk

### Classes

- class SparseTree

  *A data structure for sparsely representing trees containing SparseTreeNode objects.*

- class SparseTree::Iterator

  *An iterator for SparseTrees.*

- class SparseTree::PreorderIterator

  *Preorder depth-first iterator for SparseTrees.*

- class SparseTree::PostorderIterator
- class SparseTree::BreadthFirstIterator

*Breadth-first iterator for SparseTrees.*

## E.83 tree/tree-node.cc File Reference

```
#include <iostream>
#include "tree/tree-node.h"
```

## E.84 tree/tree-node.h File Reference

```
#include <iostream>
#include <vector>
```

### Namespaces

- namespace libpmk

### Classes

- class TreeNode

  *An indexed node, used by Tree.*

## E.85 tree/tree.cc File Reference

```
#include <assert.h>
#include <list>
#include <stack>
#include <ext/hash_map>
#include "tree/tree.h"
#include "tree/tree-node.h"
#include "util/sparse-vector.h"
```

### Namespaces

- namespace libpmk

## E.86 tree/tree.h File Reference

```
#include <list>
#include <stack>
#include <ext/hash_map>
#include "tree/tree-node.h"
```

### Namespaces

- namespace libpmk

### Classes

- class Tree

  *A data structure for representing trees containing TreeNode objects.*

- class Tree::Iterator

  *An iterator for Trees.*

- class Tree::PreorderIterator

  *Preorder depth-first iterator for Trees.*

- class Tree::PostorderIterator
- class Tree::BreadthFirstIterator

  *Breadth-first iterator for Trees.*

## E.87 util/bin-weight-scheme.h File Reference

### Namespaces

- namespace libpmk

### Enumerations

- enum BinWeightScheme { BIN_WEIGHT_GLOBAL, BIN_WEIGHT_INPUT_SPECIFIC }

## E.88 util/distance-computer.cc File Reference

```
#include <math.h>
#include <assert.h>
#include "util/distance-computer.h"
#include "point_set/point.h"
```

### Namespaces

- namespace libpmk

## E.89 util/distance-computer.h File Reference

```
#include "point_set/point.h"
```

### Namespaces

- namespace libpmk

### Classes

- class DistanceComputer

  *An abstract interface for computing distance between two Points.*

- class L1DistanceComputer

  *Computes L1 distance between two Points.*

- class L2DistanceComputer

  *Computes the* **square** *of the L2 distance between two Points.*

## E.90 util/labeled-index.cc File Reference

```
#include "util/labeled-index.h"
```

### Namespaces

- namespace libpmk

## E.91 util/labeled-index.h File Reference

### Namespaces

- namespace libpmk

### Classes

- class LabeledIndex

  *An index-label pair.*

## E.92 util/sparse-vector.cc File Reference

```
#include "util/sparse-vector.h"
```

## E.93 util/sparse-vector.h File Reference

```
#include <vector>
```

### Namespaces

- namespace libpmk

### Typedefs

- typedef vector< int > LargeIndex
- typedef pair< LargeIndex, double > IndexValuePair
- typedef vector< IndexValuePair > SparseVector

# Index

# References


[1] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm. 7

[2] Kristen Grauman and Trevor Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 1458–1465, Washington, DC, USA, 2005. IEEE Computer Society. 3

[3] Kristen Grauman and Trevor Darrell. Approximate correspondences in high dimensions. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA, 2007. 3