

TiMBL: Tilburg Memory-Based Learner

version 6.0

API Reference Guide

ILK Technical Report – ILK 07-05

Ko van der Sloot

Induction of Linguistic Knowledge
Computational Linguistics
Tilburg University

P.O. Box 90153, NL-5000 LE, Tilburg, The Netherlands
URL: <http://ilk.uvt.nl>

July 11, 2007

Contents

1	Changes	4
1.1	From version 5.1 to 6.0	4
1.2	From version 5.0 to 5.1	4
2	Quick-start	5
2.1	Setting up an experiment	5
2.2	Running an experiment	5
2.2.1	Training	6
2.2.2	Testing	6
2.2.3	special cases of Learning and Testing	6
2.3	More about settings	6
2.4	Storing and retrieving intermediate results	7
3	Getting the details	9
3.1	Classify functions: elementary	9
3.2	Classify functions: advanced	10
3.3	Classify functions: neighborSets	11
3.3.1	How to get results from a neighborSet	11
3.3.2	Usefull operations on neighborSet objects	12
4	Advanced Functions	13
4.1	Modifying the InstanceBase	13
4.2	Getting more information out of Timbl	13
5	Server Mode	15

6	Annotated example programs	16
6.0.1	example 1, api_test1.cxx	16
6.0.2	example 2, api_test2.cxx	19
6.0.3	example 3, api_test3.cxx	22
6.0.4	example 4, api_test4.cxx	25
6.0.5	example 5, api_test5.cxx	30
6.0.6	example 6, api_test6.cxx	33

Preface

This is a brief description of the TimblAPI class and its main functions. Not everything found in `TimblAPI.h` is described. Some functions are still “work in progress” and some others are artefacts to simplify the implementation of the TiMBL main program¹. To learn more about using the API, you should study programs like `classify.cxx`, `tse.cxx`, and the examples given in this manual, which can be found in the `demos` directory of this distribution.

As you can learn from these examples, all you need to get access to the TimblAPI functions, is to include `TimblAPI.h` in the program, and to include `libTimbl.a` in your linking path.

Important note: The described functions return a result (mostly a `bool`) to indicate succes or failure. To simplify the examples, we ignore these return values. This is, of course, bad practice, to be avoided in real life programming.²

Warning: Although the TiMBL internals perform some sanity checking, it is quite possible to combine API functions such that some undetermined state is reached, or even a conflict arises. The effect of the `SetOptions()` function, for instance, might be quite surprising. If you have created your own program with the API it might be wise to test against well-know data to see if the results make sense.

¹Timbl.cxx is therefore *not* a good example of how to use the API.

²as stated by commandment 6 of “The Ten Commandments for C Programmers” by Henry Spencer:

If a function be advertised to return an error code in the event of difficulties, thou shalt check for that code, yea, even though the checks triple the size of thy code and produce aches in thy typing fingers, for if thou thinkest “it cannot happen to me”, the gods shall surely punish thee for thy arrogance.

Chapter 1

Changes

1.1 From version 5.1 to 6.0

The major change in 6.0 is the introduction of the `neighborSet` class, with some special `Classify` functions. We added `Classify` functions that deliver pointers into Timbl's internal data. This is fast, but dangerous. Also, a `WriteInstanceBaseXml()` function is added, which comes in handy when you want to know more about the instance base. Two more examples demonstrating `neighborSets` and such are added in Appendix B.

1.2 From version 5.0 to 5.1

The API is quite stable at the moment. Most TiMBL changes did not affect the API. The only real API change is in the `GetWeights()` function. (see the section on Storing and retrieving intermediate results). A few options were added to Timbl, influencing the table in Appendix A. We have also changed and enhanced the examples in Appendix B.

Chapter 2

Quick-start

2.1 Setting up an experiment

There is just one way to start a TiMBL experiment, which is to call the `TimblAPI` constructor:

```
TimblAPI( const std::string& args, const std::string& name = " " );
```

`args` is used as a "command line" and is parsed for all kind of options which are used to create the right kind of experiment with the desired settings for metric, weighting etc. If something is wrong with the settings, *no* object is created.

The most important option is `-a` to set the kind of algorithm, e.g. `-a IB1` to get an IB1 experiment or `-a IGTREE` to get an IGTREE experiment. A list of possible options is give in Appendix A.

The optional name can be useful if you have multiple experiments. In case of warnings or errors, this name is appended to the message.

For example:

```
TimblAPI *My_Experiment = new TimblAPI( "-a IGTREE +vDI+DB",  
                                         "test1" );
```

`My_Experiment` is created as an IGTREE experiment with the name "test1" and the verbosity is set to DI+DB, meaning that the output will contain DIstance and DistriBution information.

The counterpart to creation is the `~TimblAPI()` destructor, which is called when you delete an experiment:

```
delete My_Experiment;
```

2.2 Running an experiment

Assuming that we have appropriate datafiles (such as `dimin.train` and `dimin.test` in the TiMBL package), we can get started right away with the functions `Learn()` and `Test()`.

2.2.1 Training

```
bool Learn( const std::string& f );
```

This function takes a file with name 'f', and gathers information such as: number of features, number and frequency of feature values and the same for class names. After that, these data are used to calculate a lot of statistical information, which will be used for testing. Finally, an InstanceBase is created, tuned to the current algorithm.

2.2.2 Testing

```
bool Test( const std::string& in,
           const std::string& out,
           const std::string& perc = "" );
```

Test a file given by 'in' and write results to 'out'. If 'perc' is not empty, then a percentage score is written to file 'perc'.

For example:

```
My_Experiment->Learn( "dimin.train" );
My_Experiment->Test( "dimin.test", "my_first_test" );
```

An InstanceBase will be created from dimin.train, then dimin.test is tested against that InstanceBase and output is written to my_first_test.

2.2.3 special cases of Learning and Testing

There are special cases where Learn () behaves differently:

- When the algorithm is IB2, Learn () will automatically take the first n lines of f (set with the -b n option) to bootstrap itself, and then the rest of f for IB2-learning. After Learning IB2, you can use Test () as usual.
- When the algorithm is CV, Learn () is not defined, and all work is done in a special version of Test (). 'f' is assumed to give the name of a file, which, on separate lines, gives the names of the files to be cross-validated.

See Appendix B for a complete CV example (program api_test3).

2.3 More about settings

After an experiment is set up with the TimblAPI constructor, many options can be changed "on the fly" with:

```
bool SetOptions( const std::string& opts );
```

'opts' is interpreted as a list of options which are set, just like in the TimblAPI constructor. When an error in the opts string is found, `SetOptions()` returns false. Whether any options are really set or changed in that case is undefined. Note that a few options can only be set *once* when creating the experiment, most notably the algorithm. Any attempt to change these options will result in a failure. See Appendix A for all valid options and information about the possibility to change them within a running experiment.

Note: `SetOptions()` is "lazy"; changes are cached until the moment they are really needed, so you can do several `SetOptions()` calls with even different values for the same option. Only the last one seen will be used for running the experiment.

To see which options are in effect, you can use the calls `ShowOptions()` and `ShowSettings()`.

```
bool ShowOptions( std::ostream& );
```

Shows all options with their possible and current values.

```
bool ShowSettings( std::ostream& );
```

Shows all options and their current values.

For example:

```
My_Experiment->SetOptions( "-w2 -m:M" );  
My_Experiment->SetOptions( "-w3 -v:DB" );  
My_Experiment->ShowSettings( cout );
```

See Appendix B (program `api_test1`) for the output.

2.4 Storing and retrieving intermediate results

To speed up testing, or to manipulate what is happening internally, we can store and retrieve several important parts of our experiment: The `InstanceBase`, the `FeatureWeights`, and the `ProbabilityArrays`.

Saving is done with:

```
bool WriteInstanceBase( const std::string& f );  
bool SaveWeights( const std::string& f );  
bool WriteArrays( const std::string& f );
```

Retrieve with their counterparts:

```
bool GetInstanceBase( const std::string& f );  
bool GetWeights( const std::string& f, Weighting w );  
bool GetArrays( const std::string& f );
```

All use 'f' as a filename for storing/retrieving. `GetWeights` needs information to decide *which* weighting to retrieve. `Weighting` is defined as the enumerated type:

```
enum Weighting { UNKNOWN_W, UD, NW, GR, IG, X2, SV };
```

Some notes:

1. The InstanceBase is stored in a internal format, with or without hashing, depending on the -H option. The format is described in the TiMBL manual. Remember that it is a bad idea to edit this file in any way.
2. GetWeights() can be used to override the weights that Learn() calculated. UNKNOWN_W should not be used.
3. The Probability arrays are described in the TiMBL manual. They can be manipulated to tune MVDM testing.

If you like you may dump the Instancebase in XML format. No Retrieve function is available for this format.

```
bool WriteInstanceBaseXml( const std::string& f );
```

Chapter 3

Getting the details

3.1 Classify functions: elementary

After an experiment is trained with `Learn()`, we do not have to use `Test()` to do bulk-testing on a file. We can create our own tests with the `Classify` functions:

```
bool Classify( const std::string& Line, std::string& result );
bool Classify( const std::string& Line, std::string& result,
               double& distance );
bool Classify( const std::string& Line, std::string& result,
               std::string& Distrib, double& distance );
```

Results are stored in 'result' (the assigned class). 'distance' will get the calculated distance, and 'Distrib' the distribution at 'distance' which is used to calculate 'result'. Distrib will be a string like "{ NP 2, PP 6 }". It is up to you to parse and interpret this. (In this case: There were 8 classes assigned at 'distance', 2 NP's and 6 PP's, giving a 'result' of "PP")

If you want to perform analyses on these distributions, it might be a good idea to read the next section about the other range of `Classify()` functions.

A main disadvantage compared to using `Test()` is that `Test()` is optimized. `Classify()` has to test for sanity of its input and also whether a `SetOptions()` has been performed. This slows down the process.

A good example of the use of `Classify()` is the `classify.cxx` program in the TiMBL Distribution.

Depending on the Algorithm and Verbosity setting, it may be possible to get some extra information on the details of each classification using:

```
const bool ShowBestNeighbors( std::ostream& os, bool distr ) const;
```

Provided that the option `+v n` or `+v k` is set and we use IB1 or IB2, output is produced similar to what we see in the TiMBL program. When 'distr' is true, their distributions are also displayed. Bear in mind: The `+vn` option is expensive in time and memory and does not work for IGTREE, TRIBL, and TRIBL-2.

3.2 Classify functions: advanced

A faster, but more dangerous version of `Classify` is also available. It is faster because it returns pointers into Timbl's internal datastructures. It is dangerous because it returns pointers into Timbl's internal datastructures (using 'const' pointers, so it is fortunately difficult to really damage Timbl)

```
const TargetValue *Classify( const std::string& );
const TargetValue *Classify( const std::string&,
                             const ValueDistribution * & );
const TargetValue *Classify( const std::string&, double& );
const TargetValue *Classify( const std::string&,
                             const ValueDistribution * &,
                             double& );
```

A `ValueDistribution` is a list-like object (but it is not a real list!) that contains `TargetValue` objects and weights. It is the result of combining all nearest neighbors and applying the desired weightings. Timbl chooses a best `TargetValue` from this `ValueDistribution` and the `Classify` functions return that as their main result.

Important: Because these functions return pointers into Timbl's internal representation, the results are only valid until the next `Classify` function is called. (or the experiment is deleted).

Both the `TargetValue` and `ValueDistribution` objects have output operators defined, so you can print them. `TargetValue` also has a `Name()` function, which returns a `std::string` so you can collect results. `ValueDistribution` has an iterator-like interface which makes it possible to walk through the `Distribution`.

An iterator on a `ValueDistribution *vd` is created like this:

```
ValueDistribution::dist_iterator it=vd->begin();
```

Unfortunately, the iterator cannot be printed or used directly. It walks through a map like structure with pairs of values, of which only the `second` part is of interest to you. You may print it, or extract its `Value()` (which happens to be a `TargetValue` pointer) or extract its `Weight()`, which is a double.

Like this:

```
while ( it != vd->end() ){
    cout << it->second << " has a value: ";
    cout << it->second->Value() << " an a weight of "
         << it->second->Weight() << endl;
    ++it;
}
```

Printing `it->second` is in fact nothing else than printing the `TargetValue` plus its `Weight`.

In the *demos* directory you will find a complete example in `api_test6`.

Warning: it is possible to search the Timbl code for the internal representation of the `TargetValue` and `ValueDistribution` objects, but please DON'T DO THAT. The representation might change between Timbl versions.

3.3 Classify functions: neighborSets

A more flexible way of classifying is to use one of these functions:

```
const neighborSet *classifyNS( const std::string& );
bool classifyNS( const std::string&, neighborSet& );
```

The first function will classify an instance and return a pointer to a `neighborSet` object. This object may be seen as an container which holds both distances and distributions up to a certain depth, (which is *at least* the number of neighbors (-k option) that was used for the classifying task.) It is a const object, so you cannot directly manipulate its internals, but there are some functions defined to get useful information out of the `neighborSet`.

Important: The `neighborSet` *will be overwritten* on the next call to any of the classify functions. Be sure to get all the results out before that happens.

To make life easy, a second variant can be used, which fills a `neighborSet` object that you provide (the same could be achieved by a copy of the result of the first function).

Note: NeighborSets can be large, and copying therefore expensive, so you should only do this if you really have to.

3.3.1 How to get results from a neighborSet

No metric functions (such as exponential decay and the like) are performed on the `neighborSet`. You are free to insert your own metrics, or use Timbls built-in metrics.

```
double getDistance( size_t n ) const;
double bestDistance() const;
const ValueDistribution *getDistribution( size_t n ) const;
ValueDistribution *bestDistribution( const decayStruct * ds=0,
                                   size_t n=0 ) const ;
```

`getDistance(n)` will return the distance of the neighbor(s) at n. `bestDistance()` is simply `getDistance(0)`.

`getDistribution(n)` will return the distribution of neighbor(s) at n.

`bestDistribution()` will return the Weighted distribution calculated using the first n elements in the container and a metric specified by the `decayStruct`. The default `n=0`, means: use the whole container. An empty decay struct means `zeroDecay`.

The returned `ValueDistribution` object is handed to you, and you are responsible for deleting it after using it (see the previous section for more details about `ValueDistributions`).

A `decayStruct` is one of:

```
class zeroDecay();
class invLinDecay();
class invDistDecay();
class expDecay( double alpha );
class expDecay( double alpha, double beta );
```

For example, to get a ValueDistribution from a neighborSet nb, using 3 neighbors and exponential decay with alpha=0.3, you can do:

```
decayStruct *dc = new expDecay(0.3);
ValueDistribution *vd = nb->bestDistribution( dc, 3 );
```

3.3.2 Usefull operations on neighborSet objects

You can print neighborSet objects:

```
std::ostream& operator<<( std::ostream&, const neighborSet& );
std::ostream& operator<<( std::ostream&, const neighborSet * );
```

You may create a neighborSet yourself, and assign and delete them:

```
neighborSet();
neighborSet( const neighborSet& );
neighborSet& operator=( const neighborSet& );
~neighborSet();
```

If you create an neighborSet, you might want to reserve space for it, to avoid needless reallocations. Also it can be cleared, and you can ask the size (just like with normal containers):

```
void reserve( size_t );
void clear();
size_t size() const;
```

Two neighborSets can be merged:

```
void merge( const neighborSet& );
```

A neighborSet can be truncated at a certain level. This is useful after merging neighborSets. Merging sets with depth k and n will result in a set with a depth somewhere within the range $[max(k, n), k + n]$.

```
void truncate( size_t );
```

Chapter 4

Advanced Functions

4.1 Modifying the InstanceBase

The instanceBase can be modified with the functions:

```
bool Increment( const std::string& Line );
bool Decrement( const std::string& Line );
```

These functions add an Instance (as described by Line) to the InstanceBase, or remove it. This can only be done for IB1-like experiments (IB1, IB2, CV and LOO), and enforces a lot of statistical recalculations.

More sophisticated are:

```
bool Expand( const std::string& File );
bool Remove( const std::string& File );
```

which use the contents of File to do a bulk of Increments or Decrements, and recalculate afterwards.

4.2 Getting more information out of Timbl

There are a few convenience functions to get extra information on TiMBL and its behaviour:

```
bool WriteNamesFile( const std::string& f );
```

Create a file which resembles a C4.5 namesfile.

```
Algorithm Algo()
```

Give the current algorithm as a type enum Algorithm. First, the declaration of the Algorithm type:

```
enum Algorithm { UNKNOWN_ALG, IB1, IB2, IGTREE,
                TRIBL, TRIBL2, LOO, CV };
```

This can be printed with the helper function:

```
const std::string to_string( const Algorithm )
```

```
Weighting CurrentWeighting()
```

Gives the current weighting as a type enum `Weighting`.

Declaration of `Weighting`:

```
enum Weighting { UNKNOWN_W, UD, NW, GR, IG, X2, SV };
```

This can be printed with the helper function:

```
const std::string to_string( const Weighting )
```

```
Weighting CurrentWeightings( std::vector<double>& v )
```

Returns the current weighting as a type enum `Weighting` and also a vector `v` with all the current values of this weighting.

```
std::string& ExpName()
```

Returns the value of 'name' given at the construction of the experiment

```
static std::string VersionInfo( bool full = false )
```

Returns a string containing the Version number, the Revision and the Revision string of the current API implementation. If `full` is true, also information about the date and time of compilation is included.

Chapter 5

Server Mode

```
bool StartServer( const int port, const int max_c );
```

Starts a TimblServer on 'port' with maximally 'max_c' concurrent connections to it. Starting a server makes sense only after the experiment is trained.

Chapter 6

Annotated example programs

6.0.1 example 1, api_test1.cxx

```
#include "TimblAPI.h"
int main(){
    TimblAPI My_Experiment( "-a IGTREE +vDI+DB+F", "test1" );
    My_Experiment.SetOptions( "-w2 -mM" );
    My_Experiment.SetOptions( "-w3 -vDB" );
    My_Experiment.ShowSettings( std::cout );
    My_Experiment.Learn( "dimin.train" );
    My_Experiment.Test( "dimin.test", "my_first_test.out" );
}
```

Output:

```
Current Experiment Settings :
FLENGTH           : 0
MAXBESTS          : 500
TRIBL_OFFSET      : 0
INPUTFORMAT       : Unknown
TREE_ORDER        : Unknown
ALL_WEIGHTS       : false
WEIGHTING         : x2 [Note 1]
BIN_SIZE          : 20
IB2_OFFSET        : 0
DO_SILLY          : false
DO_DIVERSIFY      : false
DECAY             : Z
SEED              : -1
DECAYPARAM_A      : 1.00000
DECAYPARAM_B      : 1.00000
NORMALISATION     : -1
NORMFACTOR        : 1.00000
EXEMPLAR_WEIGHTS : false
IGNORE_EXEMPLAR_WEIGHTS : true
NO_EXEMPLAR_WEIGHTS_TEST : true
VERBOSITY         : F+DI [Note 2]
EXACT_MATCH       : false
DO_DOT_PRODUCT    : false
DO_COSINE         : false
HASHED_TREE       : true
GLOBAL_METRIC     : M [Note 3]
METRICS           :
MVD_LIMIT         : 1
MVD_DEFAULT+METRIC : 0
```

NEIGHBORS : 1
PROGRESS : 100000
CLIP_FACTOR : 10

Examine datafile 'dimin.train' gave the following results:
Number of Features: 12
InputFormat : C4.5

-test1-Phase 1: Reading Datafile: dimin.train
-test1-Start: 0 @ Wed Jul 11 10:43:20 2007
-test1-Finished: 2999 @ Wed Jul 11 10:43:20 2007
-test1-Calculating Entropy Wed Jul 11 10:43:20 2007
Lines of data : 2999
DB Entropy : 1.6178929
Number of Classes : 5

Feats	Vals	X-square	Variance	InfoGain	GainRatio
1	3	128.41828	0.021410184	0.030971064	0.024891536
2	50	364.75812	0.030406645	0.060860038	0.027552191
3	19	212.29804	0.017697402	0.039562857	0.018676787
4	37	449.83823	0.037499019	0.052541227	0.052620750
5	3	288.87218	0.048161417	0.074523225	0.047699231
6	61	415.64113	0.034648310	0.10604433	0.024471911
7	20	501.33465	0.041791818	0.12348668	0.034953203
8	69	367.66021	0.030648567	0.097198760	0.043983864
9	2	169.36962	0.056475363	0.045752381	0.046816705
10	64	914.61906	0.076243669	0.21388759	0.042844587
11	18	2807.0418	0.23399815	0.66970458	0.18507018
12	43	7160.3682	0.59689631	1.2780762	0.32537181

Feature Permutation based on Chi-Squared :

< 12, 11, 10, 7, 4, 6, 8, 2, 5, 3, 9, 1 >

-test1-Phase 2: Building index on Datafile: dimin.train
-test1-Start: 0 @ Wed Jul 11 10:43:20 2007
-test1-Finished: 2999 @ Wed Jul 11 10:43:20 2007
-test1-

Phase 3: Learning from Datafile: dimin.train
-test1-Start: 0 @ Wed Jul 11 10:43:20 2007
-test1-Finished: 2999 @ Wed Jul 11 10:43:20 2007

Size of InstanceBase = 148 Nodes, (2960 bytes), 99.61 % compression
Warning:-test1-Metric set to Overlap for IGTREE test. [Note 1]

Examine datafile 'dimin.test' gave the following results:
Number of Features: 12
InputFormat : C4.5

Starting to test, Testfile: dimin.test
Writing output in: my_first_test.out
Algorithm : IGTREE
Weighting : Chi-square
Feature 1 : 128.418283576224439
Feature 2 : 364.758115277811896
Feature 3 : 212.298037236345095
Feature 4 : 449.838231470681876
Feature 5 : 288.872176256387263
Feature 6 : 415.641126446691771
Feature 7 : 501.334653478280984
Feature 8 : 367.660212489714240
Feature 9 : 169.369615106487458
Feature 10 : 914.619058199288816
Feature 11 : 2807.041753278295346
Feature 12 : 7160.368151902808677

-test1-Tested: 1 @ Wed Jul 11 10:43:20 2007
-test1-Tested: 2 @ Wed Jul 11 10:43:20 2007
-test1-Tested: 3 @ Wed Jul 11 10:43:20 2007

```

-test1-Tested:      4 @ Wed Jul 11 10:43:20 2007
-test1-Tested:      5 @ Wed Jul 11 10:43:20 2007
-test1-Tested:      6 @ Wed Jul 11 10:43:20 2007
-test1-Tested:      7 @ Wed Jul 11 10:43:20 2007
-test1-Tested:      8 @ Wed Jul 11 10:43:20 2007
-test1-Tested:      9 @ Wed Jul 11 10:43:20 2007
-test1-Tested:     10 @ Wed Jul 11 10:43:20 2007
-test1-Tested:    100 @ Wed Jul 11 10:43:20 2007
-test1-Ready:      950 @ Wed Jul 11 10:43:20 2007
Seconds taken: 0.1331 (7135.13 p/s)
overall accuracy:      0.962105 (914/950)

```

Notes:

1. The `-w2` of the first `SetOptions()` is overruled with `-w3` from the second `SetOptions()`, resulting in a weighting of 3 or Chi-Square.
2. The first `SetOptions()` sets the verbosity with `+F+DI+DB`. The second `SetOptions()`, however, sets the verbosity with `-vDB`, and the resulting verbosity is therefore `F+DI`.
3. Due to the second `SetOptions()`, the default metric is set to `MVDM` — this is however not applicable to `IGTREE`. This raises a warning later on, when we start to test and the API informs us that `Overlap` is used instead.

Result in `my_first_test.out` (first 20 lines):

```

=,=,=,=,=,=,=,+,p,e,=, T, T           6619.8512628162
=,=,=,=,+,k,u,=,-,bl,u,m,E,P         2396.8557978603
+,m,I,=,-,d,A,G,-,d,},t,J,J          6619.8512628162
-,t,@,=,-,l,|,=-,G,@,n,T,T           6619.8512628162
-,=, I, n, -, str, y, =, +, m, E, nt, J, J 6619.8512628162
=,=,=,=,=,=,=,=,+,br,L,t,J,J         6619.8512628162
=,=,=,=,+,zw,A,=,-,m,@,r,T,T         6619.8512628162
=,=,=,=,-,f,u,=,+,dr,a,l,T,T         6619.8512628162
=,=,=,=,=,=,=,=,+,l,e,w,T,T         13780.219414719
=,=,=,=,+,tr,K,N,-,k,a,rt,J,J         6619.8512628162
=,=,=,=,+,o,=,-,p,u,=,T,T           3812.8095095379
=,=,=,=,=,=,=,=,+,l,A,m,E,E         3812.8095095379
=,=,=,=,=,=,=,=,+,l,A,p,J,J         6619.8512628162
=,=,=,=,=,=,=,=,+,sx,E,lm,P,P       6619.8512628162
+,l,a,=,-,d,@,=,-,k,A,st,J,J         6619.8512628162
-,s,i,=,-,f,E,r,-,st,O,k,J,J         6619.8512628162
=,=,=,=,=,=,=,=,+,sp,a,n,T,T       6619.8512628162
=,=,=,=,=,=,=,=,+,st,o,t,J,J       6619.8512628162
=,=,=,=,+,sp,a,r,-,b,u,k,J,J        6619.8512628162
+,h,I,N,-,k,@,l,-,bl,O,k,J,J        6619.8512628162

```

6.0.2 example 2, api_test2.cxx

This demonstrates IB2 learning. Our example program:

```
#include "TimblAPI.h"
int main(){
    TimblAPI *My_Experiment = new TimblAPI( "-a IB2 +vF+DI+DB" ,
                                             "test2" );

    My_Experiment->SetOptions( "-b100" );
    My_Experiment->ShowSettings( std::cout );
    My_Experiment->Learn( "dimin.train" );
    My_Experiment->Test( "dimin.test", "my_second_test.out" );
    delete My_Experiment;
    exit(1);
}
```

We create an experiment for the IB2 algorithm, with the `-b` option set to 100, so the first 100 lines of `dimin.train` will be used to bootstrap the learning, as we can see from the output:

```
Current Experiment Settings :
LENGTH           : 0
MAXBESTS         : 500
TRIBL_OFFSET     : 0
INPUTFORMAT      : Unknown
TREE_ORDER       : G/V
ALL_WEIGHTS      : false
WEIGHTING        : gr
BIN_SIZE         : 20
IB2_OFFSET       : 100
KEEP_DISTRIBUTIONS : false
DO_SLOPPY_LOO    : false
TARGET_POS       : 4294967295
DO_SILLY         : false
DO_DIVERSIFY     : false
DECAY            : Z
SEED             : -1
BEAM_SIZE        : 0
DECAYPARAM_A     : 1.00000
DECAYPARAM_B     : 1.00000
NORMALISATION    : -1
NORM_FACTOR      : 1.00000
EXEMPLAR_WEIGHTS : false
IGNORE_EXEMPLAR_WEIGHTS : true
NO_EXEMPLAR_WEIGHTS_TEST : true
VERBOSITY        : F+DI+DB
EXACT_MATCH      : false
DO_DOT_PRODUCT   : false
DO_COSINE        : false
HASHED_TREE      : true
GLOBAL_METRIC    : 0
METRICS          :
MVD_LIMIT        : 1
MVD_DEFAULT_METRIC : 0
NEIGHBORS        : 1
PROGRESS         : 100000
CLIP_FACTOR      : 10
```

Examine datafile 'dimin.train' gave the following results:

```
Number of Features: 12
InputFormat       : C4.5
```

```
-test2-Phase 1: Reading Datafile: dimin.train
-test2-Start:      0 @ Wed Jul 11 10:50:47 2007
-test2-Finished:  100 @ Wed Jul 11 10:50:47 2007
```


Feature 7 : 0.081493895194308
Feature 8 : 0.094083953380450
Feature 9 : 0.077856473802927
Feature 10 : 0.094322883333513
Feature 11 : 0.174949363233078
Feature 12 : 0.317423417741858

-test2-Tested: 1 @ Wed Jul 11 10:50:48 2007
-test2-Tested: 2 @ Wed Jul 11 10:50:48 2007
-test2-Tested: 3 @ Wed Jul 11 10:50:48 2007
-test2-Tested: 4 @ Wed Jul 11 10:50:48 2007
-test2-Tested: 5 @ Wed Jul 11 10:50:48 2007
-test2-Tested: 6 @ Wed Jul 11 10:50:48 2007
-test2-Tested: 7 @ Wed Jul 11 10:50:48 2007
-test2-Tested: 8 @ Wed Jul 11 10:50:48 2007
-test2-Tested: 9 @ Wed Jul 11 10:50:48 2007
-test2-Tested: 10 @ Wed Jul 11 10:50:48 2007
-test2-Tested: 100 @ Wed Jul 11 10:50:48 2007
-test2-Ready: 950 @ Wed Jul 11 10:50:48 2007
Seconds taken: 0.1129 (8416.99 p/s)
overall accuracy: 0.935789 (889/950), of which 15 exact matches
[Note 2]
There were 58 ties of which 48 (82.76%) were correctly resolved

Notes:

1. As we see here, 243 entries from the input file had a mismatch, and were therefore entered in the Instancebase.
2. We see that IB2 scores 93.58 %, compared to 96.21 % for IGTREE in our first example. For this data, IB2 is not a good algorithm. However, it saves a lot of space, and is faster than IB1. Yet, IGTREE is both faster and better. Had we used IB1, the score would have been 96.84 %.

6.0.3 example 3, api_test3.cxx

This demonstrates Cross Validation. Let's try the following program:

```
#include "TimblAPI.h"
using Timbl::TimblAPI;

int main(){
    TimblAPI *My_Experiment = new TimblAPI( "-t cross_validate" );
    My_Experiment->Test( "cross_val.test" );
    delete My_Experiment;
    exit(0);
}
```

This program creates an experiment, which defaults to IB1 and because of the special option “-t cross_validate” will start a CrossValidation experiment. Learn() is not possible now. We must use a special form of Test().

“cross_val.test” is a file with the following content:

```
small_1.train
small_2.train
small_3.train
small_4.train
small_5.train
```

All these files contain an equal part of a bigger dataset, and My_Experiment will run a CrossValidation test between these files. Note that output filenames are generated and that you cannot influence that.

The output of this program is:

```
Starting Cross validation test on files:
small_1.train
small_2.train
small_3.train
small_4.train
small_5.train
Examine datafile 'small_1.train' gave the following results:
Number of Features: 8
InputFormat      : C4.5
```

```
Starting to test, Testfile: small_1.train
Writing output in:      small_1.train.cv
Algorithm           : CV
Global metric       : Overlap
Deviant Feature Metrics:(none)
Weighting           : GainRatio
```

```
Tested:      1 @ Wed Jul 11 10:56:03 2007
Tested:      2 @ Wed Jul 11 10:56:03 2007
Tested:      3 @ Wed Jul 11 10:56:03 2007
Tested:      4 @ Wed Jul 11 10:56:03 2007
Tested:      5 @ Wed Jul 11 10:56:03 2007
Tested:      6 @ Wed Jul 11 10:56:03 2007
Tested:      7 @ Wed Jul 11 10:56:03 2007
Tested:      8 @ Wed Jul 11 10:56:03 2007
Tested:      9 @ Wed Jul 11 10:56:03 2007
Tested:     10 @ Wed Jul 11 10:56:03 2007
Ready:      10 @ Wed Jul 11 10:56:03 2007
```

Seconds taken: 0.0022 (4508.57 p/s)
overall accuracy: 0.800000 (8/10)
Examine datafile 'small_2.train' gave the following results:
Number of Features: 8
InputFormat : C4.5

Starting to test, Testfile: small_2.train
Writing output in: small_2.train.cv
Algorithm : CV
Global metric : Overlap
Deviant Feature Metrics:(none)
Weighting : GainRatio

Tested: 1 @ Wed Jul 11 10:56:03 2007
Tested: 2 @ Wed Jul 11 10:56:03 2007
Tested: 3 @ Wed Jul 11 10:56:03 2007
Tested: 4 @ Wed Jul 11 10:56:03 2007
Tested: 5 @ Wed Jul 11 10:56:03 2007
Tested: 6 @ Wed Jul 11 10:56:03 2007
Tested: 7 @ Wed Jul 11 10:56:03 2007
Tested: 8 @ Wed Jul 11 10:56:03 2007
Tested: 9 @ Wed Jul 11 10:56:03 2007
Tested: 10 @ Wed Jul 11 10:56:03 2007
Ready: 10 @ Wed Jul 11 10:56:03 2007

Seconds taken: 0.0021 (4777.83 p/s)
overall accuracy: 0.800000 (8/10)
Examine datafile 'small_3.train' gave the following results:
Number of Features: 8
InputFormat : C4.5

Starting to test, Testfile: small_3.train
Writing output in: small_3.train.cv
Algorithm : CV
Global metric : Overlap
Deviant Feature Metrics:(none)
Weighting : GainRatio

Tested: 1 @ Wed Jul 11 10:56:03 2007
Tested: 2 @ Wed Jul 11 10:56:03 2007
Tested: 3 @ Wed Jul 11 10:56:03 2007
Tested: 4 @ Wed Jul 11 10:56:03 2007
Tested: 5 @ Wed Jul 11 10:56:03 2007
Tested: 6 @ Wed Jul 11 10:56:03 2007
Tested: 7 @ Wed Jul 11 10:56:03 2007
Tested: 8 @ Wed Jul 11 10:56:03 2007
Tested: 9 @ Wed Jul 11 10:56:03 2007
Tested: 10 @ Wed Jul 11 10:56:03 2007
Ready: 10 @ Wed Jul 11 10:56:03 2007

Seconds taken: 0.0021 (4863.81 p/s)
overall accuracy: 0.900000 (9/10)
Examine datafile 'small_4.train' gave the following results:
Number of Features: 8
InputFormat : C4.5

Starting to test, Testfile: small_4.train
Writing output in: small_4.train.cv
Algorithm : CV
Global metric : Overlap
Deviant Feature Metrics:(none)
Weighting : GainRatio

Tested: 1 @ Wed Jul 11 10:56:03 2007
Tested: 2 @ Wed Jul 11 10:56:03 2007
Tested: 3 @ Wed Jul 11 10:56:03 2007

```
Tested:      4 @ Wed Jul 11 10:56:03 2007
Tested:      5 @ Wed Jul 11 10:56:03 2007
Tested:      6 @ Wed Jul 11 10:56:03 2007
Tested:      7 @ Wed Jul 11 10:56:03 2007
Tested:      8 @ Wed Jul 11 10:56:03 2007
Tested:      9 @ Wed Jul 11 10:56:03 2007
Tested:     10 @ Wed Jul 11 10:56:03 2007
Ready:      10 @ Wed Jul 11 10:56:03 2007
Seconds taken: 0.0021 (4837.93 p/s)
overall accuracy:      0.800000 (8/10)
Examine datafile 'small_5.train' gave the following results:
Number of Features: 8
InputFormat      : C4.5
```

```
Starting to test, Testfile: small_5.train
Writing output in:      small_5.train.cv
Algorithm      : CV
Global metric  : Overlap
Deviant Feature Metrics:(none)
Weighting      : GainRatio
```

```
Tested:      1 @ Wed Jul 11 10:56:03 2007
Tested:      2 @ Wed Jul 11 10:56:03 2007
Tested:      3 @ Wed Jul 11 10:56:03 2007
Tested:      4 @ Wed Jul 11 10:56:03 2007
Tested:      5 @ Wed Jul 11 10:56:03 2007
Tested:      6 @ Wed Jul 11 10:56:03 2007
Tested:      7 @ Wed Jul 11 10:56:03 2007
Tested:      8 @ Wed Jul 11 10:56:03 2007
Ready:      8 @ Wed Jul 11 10:56:03 2007
Seconds taken: 0.0018 (4415.01 p/s)
overall accuracy:      1.000000 (8/8)
```

What has happened here?

1. TiMBL trained itself with inputfiles small_2.train through small_5.train. (in fact using the `Expand()` API call.
2. Then TiMBL tested small_1.train against the InstanceBase.
3. Next, small_2.train is removed from the database (API call `Remove()`) and small_1.train is added.
4. Then small_2.train is tested against the InstanceBase.
5. And so forth with small_3.train ...

6.0.4 example 4, api_test4.cxx

This program demonstrates adding and deleting of the InstanceBase. It also proves that weights are (re)calculated correctly each time (which also explains why this is a time-consuming thing to do). After running this program, wg.1 should be equal to wg.5 and wg.2 eq to wg.4. Important to note is also, that while we do not use a weighting of X2 or SV here, only the "simple" weights are calculated and stored.

First the program:

```
#include <iostream>
#include "TimblAPI.h"

int main(){
    TimblAPI *My_Experiment = new TimblAPI( "-a IB1 +vDI+DB +mM" ,
                                           "test4" );

    My_Experiment->ShowSettings( std::cout );
    My_Experiment->Learn( "dimin.train" );
    My_Experiment->Test( "dimin.test", "incl.out" );
    My_Experiment->SaveWeights( "wg.1" );
    My_Experiment->WriteArrays( "arr.1" );
    My_Experiment->Increment( "=",,=,=,+,k,e,=-,r,@,l,T" );
    My_Experiment->Test( "dimin.test", "inc2.out" );
    My_Experiment->SaveWeights( "wg.2" );
    My_Experiment->WriteArrays( "arr.2" );
    My_Experiment->Increment( "+,zw,A,rt,-,k,O,p,-,n,O,n,E" );
    My_Experiment->Test( "dimin.test", "inc3.out" );
    My_Experiment->SaveWeights( "wg.3" );
    My_Experiment->WriteArrays( "arr.3" );
    My_Experiment->Decrement( "+,zw,A,rt,-,k,O,p,-,n,O,n,E" );
    My_Experiment->Test( "dimin.test", "inc4.out" );
    My_Experiment->SaveWeights( "wg.4" );
    My_Experiment->WriteArrays( "arr.4" );
    My_Experiment->Decrement( "=",,=,=,+,k,e,=-,r,@,l,T" );
    My_Experiment->Test( "dimin.test", "inc5.out" );
    My_Experiment->SaveWeights( "wg.5" );
    My_Experiment->WriteArrays( "arr.5" );
    delete My_Experiment;
    exit(1);
}
```

This produces the following output:

```
Current Experiment Settings :
LENGTH           : 0
MAXBESTS        : 500
TRIBL_OFFSET     : 0
INPUTFORMAT     : Unknown
TREE_ORDER      : G/V
ALL_WEIGHTS     : false
WEIGHTING       : gr
BIN_SIZE        : 20
IB2_OFFSET      : 0
KEEP_DISTRIBUTIONS : false
DO_SLOPPY_LOO  : false
TARGET_POS      : 4294967295
DO_SILLY       : false
DO_DIVERSIFY    : false
DECAY           : Z
SEED            : -1
BEAM_SIZE       : 0
DECAYPARAM_A   : 1.00000
DECAYPARAM_B   : 1.00000
```

NORMALISATION : -1
NORM_FACTOR : 1.00000
EXEMPLAR_WEIGHTS : false
IGNORE_EXEMPLAR_WEIGHTS : true
NO_EXEMPLAR_WEIGHTS_TEST : true
VERBOSITY : DI+DB
EXACT_MATCH : false
DO_DOT_PRODUCT : false
DO_COSINE : false
HASHED_TREE : true
GLOBAL_METRIC : M
METRICS :
MVD_LIMIT : 1
MVD_DEFAULT_METRIC : 0
NEIGHBORS : 1
PROGRESS : 100000
CLIP_FACTOR : 10

Examine datafile 'dimin.train' gave the following results:

Number of Features: 12
InputFormat : C4.5

-test4-Phase 1: Reading Datafile: dimin.train
-test4-Start: 0 @ Wed Jul 11 11:00:02 2007
-test4-Finished: 2999 @ Wed Jul 11 11:00:02 2007
-test4-Calculating Entropy Wed Jul 11 11:00:02 2007
Feature Permutation based on GainRatio/Values :
< 9, 5, 11, 1, 12, 7, 4, 3, 10, 8, 2, 6 >
-test4-Phase 2: Learning from Datafile: dimin.train
-test4-Start: 0 @ Wed Jul 11 11:00:02 2007
-test4-Finished: 2999 @ Wed Jul 11 11:00:02 2007

Size of InstanceBase = 19231 Nodes, (384620 bytes), 49.77 % compression

Examine datafile 'dimin.test' gave the following results:

Number of Features: 12
InputFormat : C4.5

Starting to test, Testfile: dimin.test
Writing output in: incl.out
Algorithm : IBl
Global metric : Value Difference, Prestored matrix
Deviant Feature Metrics:(none)
Size of value-matrix[1] = 96 Bytes
Size of value-matrix[2] = 704 Bytes
Size of value-matrix[3] = 704 Bytes
Size of value-matrix[4] = 96 Bytes
Size of value-matrix[5] = 96 Bytes
Size of value-matrix[6] = 1496 Bytes
Size of value-matrix[7] = 1496 Bytes
Size of value-matrix[8] = 336 Bytes
Size of value-matrix[9] = 56 Bytes
Size of value-matrix[10] = 2376 Bytes
Size of value-matrix[11] = 1344 Bytes
Size of value-matrix[12] = 936 Bytes
Total Size of value-matrices 9736 Bytes

Weighting : GainRatio

-test4-Tested: 1 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 2 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 3 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 4 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 5 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 6 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 7 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 8 @ Wed Jul 11 11:00:02 2007

```
-test4-Tested:      9 @ Wed Jul 11 11:00:02 2007
-test4-Tested:     10 @ Wed Jul 11 11:00:02 2007
-test4-Tested:    100 @ Wed Jul 11 11:00:02 2007
-test4-Ready:     950 @ Wed Jul 11 11:00:02 2007
Seconds taken: 0.1529 (6211.75 p/s)
overall accuracy:      0.964211 (916/950), of which 62 exact matches
```

There were 6 ties of which 6 (100.00%) were correctly resolved

```
-test4-Saving Weights in wg.1.wgt
-test4-Saving Probability Arrays in arr.1.arr
Examine datafile 'dimin.test' gave the following results:
Number of Features: 12
InputFormat       : C4.5
```

```
Starting to test, Testfile: dimin.test
Writing output in:      inc2.out
Algorithm           : IB1
Global metric      : Value Difference, Prestored matrix
Deviant Feature Metrics:(none)
Size of value-matrix[1] = 96 Bytes
Size of value-matrix[2] = 704 Bytes
Size of value-matrix[3] = 704 Bytes
Size of value-matrix[4] = 96 Bytes
Size of value-matrix[5] = 96 Bytes
Size of value-matrix[6] = 1496 Bytes
Size of value-matrix[7] = 1496 Bytes
Size of value-matrix[8] = 336 Bytes
Size of value-matrix[9] = 56 Bytes
Size of value-matrix[10] = 2376 Bytes
Size of value-matrix[11] = 1344 Bytes
Size of value-matrix[12] = 936 Bytes
Total Size of value-matrices 9736 Bytes
```

Weighting : GainRatio

```
-test4-Tested:      1 @ Wed Jul 11 11:00:02 2007
-test4-Tested:      2 @ Wed Jul 11 11:00:02 2007
-test4-Tested:      3 @ Wed Jul 11 11:00:02 2007
-test4-Tested:      4 @ Wed Jul 11 11:00:02 2007
-test4-Tested:      5 @ Wed Jul 11 11:00:02 2007
-test4-Tested:      6 @ Wed Jul 11 11:00:02 2007
-test4-Tested:      7 @ Wed Jul 11 11:00:02 2007
-test4-Tested:      8 @ Wed Jul 11 11:00:02 2007
-test4-Tested:      9 @ Wed Jul 11 11:00:02 2007
-test4-Tested:     10 @ Wed Jul 11 11:00:02 2007
-test4-Tested:    100 @ Wed Jul 11 11:00:02 2007
-test4-Ready:     950 @ Wed Jul 11 11:00:02 2007
Seconds taken: 0.2206 (4305.52 p/s)
overall accuracy:      0.964211 (916/950), of which 62 exact matches
```

There were 6 ties of which 6 (100.00%) were correctly resolved

```
-test4-Saving Weights in wg.2.wgt
-test4-Saving Probability Arrays in arr.2.arr
Examine datafile 'dimin.test' gave the following results:
Number of Features: 12
InputFormat       : C4.5
```

```
Starting to test, Testfile: dimin.test
Writing output in:      inc3.out
Algorithm           : IB1
Global metric      : Value Difference, Prestored matrix
Deviant Feature Metrics:(none)
Size of value-matrix[1] = 96 Bytes
Size of value-matrix[2] = 704 Bytes
Size of value-matrix[3] = 704 Bytes
```

Size of value-matrix[4] = 96 Bytes
Size of value-matrix[5] = 96 Bytes
Size of value-matrix[6] = 1496 Bytes
Size of value-matrix[7] = 1496 Bytes
Size of value-matrix[8] = 336 Bytes
Size of value-matrix[9] = 56 Bytes
Size of value-matrix[10] = 2376 Bytes
Size of value-matrix[11] = 1344 Bytes
Size of value-matrix[12] = 936 Bytes
Total Size of value-matrices 9736 Bytes

Weighting : GainRatio

-test4-Tested: 1 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 2 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 3 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 4 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 5 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 6 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 7 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 8 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 9 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 10 @ Wed Jul 11 11:00:02 2007
-test4-Tested: 100 @ Wed Jul 11 11:00:02 2007
-test4-Ready: 950 @ Wed Jul 11 11:00:03 2007
Seconds taken: 0.1681 (5652.84 p/s)
overall accuracy: 0.964211 (916/950), of which 62 exact matches

There were 6 ties of which 6 (100.00%) were correctly resolved

-test4-Saving Weights in wg.3.wgt
-test4-Saving Probability Arrays in arr.3.arr
Examine datafile 'dimin.test' gave the following results:
Number of Features: 12
InputFormat : C4.5

Starting to test, Testfile: dimin.test
Writing output in: inc4.out
Algorithm : IB1
Global metric : Value Difference, Prestored matrix
Deviant Feature Metrics:(none)
Size of value-matrix[1] = 96 Bytes
Size of value-matrix[2] = 704 Bytes
Size of value-matrix[3] = 704 Bytes
Size of value-matrix[4] = 96 Bytes
Size of value-matrix[5] = 96 Bytes
Size of value-matrix[6] = 1496 Bytes
Size of value-matrix[7] = 1496 Bytes
Size of value-matrix[8] = 336 Bytes
Size of value-matrix[9] = 56 Bytes
Size of value-matrix[10] = 2376 Bytes
Size of value-matrix[11] = 1344 Bytes
Size of value-matrix[12] = 936 Bytes
Total Size of value-matrices 9736 Bytes

Weighting : GainRatio

-test4-Tested: 1 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 2 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 3 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 4 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 5 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 6 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 7 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 8 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 9 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 10 @ Wed Jul 11 11:00:03 2007

-test4-Tested: 100 @ Wed Jul 11 11:00:03 2007
-test4-Ready: 950 @ Wed Jul 11 11:00:03 2007
Seconds taken: 0.1755 (5414.06 p/s)
overall accuracy: 0.964211 (916/950), of which 62 exact matches

There were 6 ties of which 6 (100.00%) were correctly resolved

-test4-Saving Weights in wg.4.wgt
-test4-Saving Probability Arrays in arr.4.arr
Examine datafile 'dimin.test' gave the following results:
Number of Features: 12
InputFormat : C4.5

Starting to test, Testfile: dimin.test
Writing output in: inc5.out
Algorithm : IB1
Global metric : Value Difference, Prestored matrix
Deviant Feature Metrics:(none)
Size of value-matrix[1] = 96 Bytes
Size of value-matrix[2] = 704 Bytes
Size of value-matrix[3] = 704 Bytes
Size of value-matrix[4] = 96 Bytes
Size of value-matrix[5] = 96 Bytes
Size of value-matrix[6] = 1496 Bytes
Size of value-matrix[7] = 1496 Bytes
Size of value-matrix[8] = 336 Bytes
Size of value-matrix[9] = 56 Bytes
Size of value-matrix[10] = 2376 Bytes
Size of value-matrix[11] = 1344 Bytes
Size of value-matrix[12] = 936 Bytes
Total Size of value-matrices 9736 Bytes

Weighting : GainRatio

-test4-Tested: 1 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 2 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 3 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 4 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 5 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 6 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 7 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 8 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 9 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 10 @ Wed Jul 11 11:00:03 2007
-test4-Tested: 100 @ Wed Jul 11 11:00:03 2007
-test4-Ready: 950 @ Wed Jul 11 11:00:03 2007
Seconds taken: 0.1656 (5737.44 p/s)
overall accuracy: 0.964211 (916/950), of which 62 exact matches

There were 6 ties of which 6 (100.00%) were correctly resolved

-test4-Saving Weights in wg.5.wgt
-test4-Saving Probability Arrays in arr.5.arr

6.0.5 example 5, api_test5.cxx

This program demonstrates the use of neighborSets to classify and store results. It also demonstrates some neighborSet basics.

```
#include <iostream>
#include <string>
#include "TimblAPI.h"

using std::endl;
using std::cout;
using std::string;
using namespace Timbl;

int main(){
    TimblAPI *My_Experiment = new TimblAPI( "-a IB1 +vDI+DB+n +mM +k4 " ,
                                             "test5" );
    My_Experiment->Learn( "dimin.train" );
    {
        string line = "=,=,=,=,+k,e,=-,r,@,l,T";
        const neighborSet *neighbours1 = My_Experiment->classifyNS( line );
        if ( neighbours1 ){
            cout << "Classify OK on " << line << endl;
            cout << neighbours1;
        } else
            cout << "Classify failed on " << line << endl;
        neighborSet neighbours2;
        line = "+,zw,A,rt,-,k,O,p,-,n,O,n,E";
        if ( My_Experiment->classifyNS( line, neighbours2 ) ){
            cout << "Classify OK on " << line << endl;
            cout << neighbours2;
        } else
            cout << "Classify failed on " << line << endl;
        line = "+,z,O,n,-,d,A,xs,-,=,Ar,m,P";
        const neighborSet *neighbours3 = My_Experiment->classifyNS( line );
        if ( neighbours3 ){
            cout << "Classify OK on " << line << endl;
            cout << neighbours3;
        } else
            cout << "Classify failed on " << line << endl;
        neighborSet uit2;
        {
            neighborSet uit;
            cout << " before first merge " << endl;
            cout << uit;
            uit.merge( *neighbours1 );
            cout << " after first merge " << endl;
            cout << uit;
            uit.merge( *neighbours3 );
            cout << " after second merge " << endl;
            cout << uit;
            uit.merge( neighbours2 );
            cout << " after third merge " << endl;
            cout << uit;
            uit.truncate( 3 );
            cout << " after truncate " << endl;
            cout << uit;
            cout << " test assignment" << endl;
            uit2 = *neighbours1;
        }
        cout << "assignment result: " << endl;
        cout << uit2;
        {
            cout << " test copy construction" << endl;
            neighborSet uit(uit2);
            cout << "result: " << endl;
        }
    }
}
```

```

    cout << uit;
  }
  cout << "almost done!" << endl;
}
delete My_Experiment;
cout << "done!" << endl;
}

```

Its expected output is (without further comment):

```

Examine datafile 'dimin.train' gave the following results:
Number of Features: 12
InputFormat       : C4.5

```

```

-test5-Phase 1: Reading Datafile: dimin.train
-test5-Start:      0 @ Wed Jul 11 11:07:31 2007
-test5-Finished:  2999 @ Wed Jul 11 11:07:31 2007
-test5-Calculating Entropy      Wed Jul 11 11:07:31 2007
Feature Permutation based on GainRatio/Values :
< 9, 5, 11, 1, 12, 7, 4, 3, 10, 8, 2, 6 >
-test5-Phase 2: Learning from Datafile: dimin.train
-test5-Start:      0 @ Wed Jul 11 11:07:31 2007
-test5-Finished:  2999 @ Wed Jul 11 11:07:31 2007

```

```

Size of InstanceBase = 19231 Nodes, (384620 bytes), 49.77 % compression
Classify OK on =,=,=,=,+,k,e,=,-,r,@,l,T
# k=1 { T 1.00000 } 0.00000000000000
# k=2 { T 1.00000 } 0.0031862902473388
# k=3 { T 1.00000 } 0.0034182315118303
# k=4 { T 1.00000 } 0.0037433772844615
Classify OK on +,zw,A,rt,-,k,O,p,-,n,O,n,E
# k=1 { E 1.00000 } 0.00000000000000
# k=2 { E 1.00000 } 0.056667880327190
# k=3 { E 1.00000 } 0.062552636617742
# k=4 { E 1.00000 } 0.064423860361889
Classify OK on +,z,O,n,-,d,A,xs,-,=,A,rm,P
# k=1 { P 1.00000 } 0.059729836255170
# k=2 { P 1.00000 } 0.087740769132651
# k=3 { P 1.00000 } 0.088442788919723
# k=4 { P 1.00000 } 0.097058649951429
  before first merge
  after first merge
# k=1 { P 1.00000 } 0.059729836255170
# k=2 { P 1.00000 } 0.087740769132651
# k=3 { P 1.00000 } 0.088442788919723
# k=4 { P 1.00000 } 0.097058649951429
  after second merge
# k=1 { P 2.00000 } 0.059729836255170
# k=2 { P 2.00000 } 0.087740769132651
# k=3 { P 2.00000 } 0.088442788919723
# k=4 { P 2.00000 } 0.097058649951429
  after third merge
# k=1 { E 1.00000 } 0.00000000000000
# k=2 { E 1.00000 } 0.056667880327190
# k=3 { P 2.00000 } 0.059729836255170
# k=4 { E 1.00000 } 0.062552636617742
# k=5 { E 1.00000 } 0.064423860361889
# k=6 { P 2.00000 } 0.087740769132651
# k=7 { P 2.00000 } 0.088442788919723
# k=8 { P 2.00000 } 0.097058649951429
  after truncate
# k=1 { E 1.00000 } 0.00000000000000
# k=2 { E 1.00000 } 0.056667880327190
# k=3 { P 2.00000 } 0.059729836255170
  test assignment
assignment result:

```

```
# k=1 { P 1.00000 } 0.059729836255170
# k=2 { P 1.00000 } 0.087740769132651
# k=3 { P 1.00000 } 0.088442788919723
# k=4 { P 1.00000 } 0.097058649951429
  test copy construction
result:
# k=1 { P 1.00000 } 0.059729836255170
# k=2 { P 1.00000 } 0.087740769132651
# k=3 { P 1.00000 } 0.088442788919723
# k=4 { P 1.00000 } 0.097058649951429
almost done!
done!
```

6.0.6 example 6, api_test6.cxx

This program demonstrates the use of ValueDistributions, TargetValues and neighborSets for classification.

```
#include <iostream>
#include "TimblAPI.h"

using std::cout;
using std::endl;
using namespace Timbl;

int main(){
    TimblAPI My_Experiment( "-a IBl +vDI+DB -k3", "test6" );
    My_Experiment.Learn( "dimin.train" );
    const ValueDistribution *vd;
    const TargetValue *tv
    = My_Experiment.Classify( "-,=,0,m,+,h,K,=-,n,I,N,K", vd );
    cout << "resulting target: " << tv << endl;
    cout << "resulting Distribution: " << vd << endl;
    ValueDistribution::dist_iterator it=vd->begin();
    while ( it != vd->end() ){
        cout << it->second << " OR ";
        cout << it->second->Value() << " " << it->second->Weight() << endl;
        ++it;
    }

    cout << "the same with neighborSets" << endl;
    const neighborSet *nb = My_Experiment.classifyNS( "-,=,0,m,+,h,K,=-,n,I,N,K" );
    ValueDistribution *vd2 = nb->bestDistribution();
    cout << "default answer " << vd2 << endl;
    decayStruct *dc = new expDecay(0.3);
    delete vd2;
    vd2 = nb->bestDistribution( dc );
    delete dc;
    cout << "with exponential decay, alpha = 0.3 " << vd2 << endl;
    delete vd2;
}
```

This is the output produced:

```
Examine datafile 'dimin.train' gave the following results:
Number of Features: 12
InputFormat      : C4.5

-test6-Phase 1: Reading Datafile: dimin.train
-test6-Start:    0 @ Wed Jul 11 11:12:07 2007
-test6-Finished: 2999 @ Wed Jul 11 11:12:07 2007
-test6-Calculating Entropy      Wed Jul 11 11:12:07 2007
Feature Permutation based on GainRatio/Values :
< 9, 5, 11, 1, 12, 7, 4, 3, 10, 8, 2, 6 >
-test6-Phase 2: Learning from Datafile: dimin.train
-test6-Start:    0 @ Wed Jul 11 11:12:07 2007
-test6-Finished: 2999 @ Wed Jul 11 11:12:08 2007

Size of InstanceBase = 19231 Nodes, (384620 bytes), 49.77 % compression
resulting target: K
resulting Distribution: { E 1.00000, K 7.00000 }
E 1 OR E 1
K 7 OR K 7
the same with neighborSets
default answer { E 1.00000, K 7.00000 }
with exponential decay, alpha = 0.3 { E 0.971556, K 6.69810 }
```