TiMBL: Tilburg Memory-Based Learner

version 5.0

API Reference Guide

ILK Technical Report – ILK 03-12

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

November 26, 2003

1 Preface

This is 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 program[1].
To learn more about using the API, you should study programs like `classify.cxx`, `tse.cxx` and of course 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: All described functions return 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.

Warning

Although the Timbl internals perform some sanity checking, it is quite possible to combine API functions in such manner 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.

2 changes from 4.x to 5.0

In the progress of making the Timbl internals more robust, and more OO-like, we made a shift to use more STL classes like 'std::map' and 'std::string'. This mostly affects internals without touching the API, as it should be. But we decided to switch to 'std::string' at the outer level also. This change requires some moderate reprogramming of excisting TimblAPI based programs. An advantage is, that the users of the API don't have to bother with "free-ing' returned objects anymore.

3 Starting our first experiment

There is just one way to start a Timbl experiment, and that is by calling 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.

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

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.

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.

Of course there is a counterpart to creation in the form of the ˜TimblAPI() destructor, which is called when you delete an experiment with:

```
delete My_Experiment;
```

4 Running our first experiment

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

4.1 Training

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

This function takes a file with name ’f’, and gathers information like: 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.

Note: 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().

- When the algorithm is CV, Learn() is not defined, and all work is done in a special version of Test().

4.2 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' isn't NULL, then a percentage score is written to file 'perc'.

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.

For Crossvalidation, there is a special implementation for Test(f).
'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 example. (program api_test2).

5 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 for a running experiment.

Note: SetOptions() is performed "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 currect values.

Example:

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

See Appendix B (program api_test) for the output.

6 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 );
bool GetArrays( const std::string& f );
```

All use 'f' as a filename for storing/retrieving.

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. SaveWeights() only saves the current active weights, so you loose information on the whereabouts, such as whether it were IG weights or GR weights. GetWeights() can be used to override the weights that Learn() calculated.

3. The Probability arrays are described in the Timbl manual. They can be manipulated to tune MVDM testing.

7 Do it ourselves

After an experiment is trained with Learn(), we don't have to use Test() to do bulk-testing on a file. We can wrap 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 );
```

All versions will classify Line against the InstanceBase.
Results are stored in 'result' (the assigned class). 'distance' will contain 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")

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 of course.

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 choice that is made using:

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

Provided that the option "+v n" or "+v k" is set and we use an IB1 like algorithm, 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 doesn't work for IG tree and the like.

Other functions that can be used to create your own special effects:

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

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

8 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.

Declaration of Algorithm:

```
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 )
```

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

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

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

return 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.

9 Using Timbl as a Server

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

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

10 Appendix A

Here we give a summmary of the options that can be set using the "Setoptions" API call. For detailed information about these options you should refer to the Timbl Manual.

Options that only can be set **once**		
option	value	description
-a		algorithm
	0 or IB1	IB1 (default)
	1 or IG	IGTree
	2 or TRIBL	TRIBL
	3 or IB2	IB2
	4 or TRIBL2	TRIBL2
-F	format	Assume the specified inputformat.
		(Compact, C4.5, ARFF, Columns, Binary, Sparse)
-l	integer	length of Features (Compact format only).
-M	integer	size of MaxBests Array
-N	integer	Number of features
-q	integer	TRIBL treshold at level n.
-T		internal order of the Tree :
	DO	none.
	GRO	using GainRatio
	IGO	using InformationGain
	1/V	using 1/# of Vals
	G/V	using GainRatio/# of Vals
	I/V	using InfoGain/# of Vals
	X2O	using X-square
	X/V	using X-square/# of Vals
	SVO	using Shared Variance
	S/V	using Shared Variance/# of Vals
	1/S	using 1/SplitInfo
Other options		
-b	integer	number of lines used for bootstrapping (IB2 only)
-B	integer	number of bins used for discretization of numeric feature
-d		weight neighbors as function of their distance:
	Z	all the same weight. (default)
	ID	Inverse Distance.
	IL	Inverse Linear
	ED:a	Exponential Decay with factor a. (no whitespace!)
	ED:a:b	Exponential Decay with factors a and b . (no whitespace!)
-e	n	estimate time until n patterns tested.
+H/-H		write/don't write hashed trees (default)
-k	integer	k nearest neighbors (default n = 1).

option	value	description
-m		metrics
	D	Dot product.
	O	weighted overlap. (default)
	M	modified value difference.
	N	numeric values.
	I	Ignore named values.
	J	Jeffrey Divergence.
-p	integer	show progress every n lines. (default p = 100,000)
-R	integer	solve ties at random with seed n.
-s		expect exemplar weights in all input files, and use for training
	0	expect but ignore exemplar weights from all files
	1	ignore exemplar weights from the test file
-t	leave_one_out	perform a leave one out test on the inputfile
-t	cross_validate	perform crossvalidation on the files specified in the inputfile
+v / -v		set or unset verbosity level, where level is
	s	work silently.
	o	show all options set.
	f	show Calculated Feature Weights. (default)
	p	show MVD matrices.
	e	show exact matches.
	cm	show Confusion Matrix.
	di	add distance to output file.
	db	add distribution of best matched to output file
	k	add summary of nearest neigbors to output file (sets -x)
	n	add all nearest neigbors to output file (sets -x)
		You may combine levels using '+' e.g. +v p+db or -v o+di
-V		Show VERSION.
-w		weighting
	0 or nw	No Weighting.
	1 or gr	Weight using GainRatio. (default)
	2 or ig	Weight using InfoGain
	3 or x2	Weight using Chi-square
	4 or sv	Weight using Shared Variance
	name	use Weights from file 'name'
+x/-x		use/don't use the exact match shortcut. (IB only) (default:-x)

11 Appendix B: Annotated example programs

11.1 example 1, api_test.cxx

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

int main(){
  TimblAPI *My_Experiment = new TimblAPI( "-a IGTREE +vDI+DB" ,
                                          "test1" );
  My_Experiment->SetOptions( "-w2 -mM" );
  My_Experiment->SetOptions( "-w3 -vDB" );
  My_Experiment->ShowSettings( cout );
  My_Experiment->Learn( "dimin.train" );
  My_Experiment->Test( "dimin.test", "my_first_test.out" );
  exit(0);
}
```

Output:

```
Current Experiment Settings :
FLENGTH              : 0
MAXBESTS             : 500
TRIBL_OFFSET         : 0
INPUTFORMAT          : Unknown
TREE_ORDER           : Unknown
DECAY                : Z
SEED                 : -1
DECAYPARAM           : 1.000000
EXEMPLAR_WEIGHTS     : -
IGNORE_EXEMPLAR_WEIGHT : +
NO_EXEMPLAR_WEIGHTS_TEST : +
PROBALISTIC          : -
VERBOSITY            : F+DI                        [Note 1]
EXACT_MATCH          : -
DO_DOT_PRODUCT       : -
HASHED_TREES         : +
GLOBAL_METRIC        : M                           [Note 2]
METRICS              :
NEIGHBORS            : 1
PROGRESS             : 100000
IB2_OFFSET           : 0
BIN_SIZE             : 20
WEIGHTING            : x2                          [Note 3]

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

-test1-Phase 1: Reading Datafile: dimin.train
-test1-Start:          0 @ Tue Jul  8 16:56:43 2003
```

```
-test1-Finished:    2999 @ Tue Jul  8 16:56:43 2003
-test1-Calculating Entropy         Tue Jul  8 16:56:43 2003
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: Learning from Datafile: dimin.train
-test1-Start:          0 @ Tue Jul  8 16:56:43 2003
-test1-Finished:    2999 @ Tue Jul  8 16:56:44 2003

Size of InstanceBase = 29481 Nodes, (589620 bytes), 23.00 % compression

-test1-Start Pruning:         Tue Jul  8 16:56:44 2003
-test1-Finished Pruning:         Tue Jul  8 16:56:44 2003

Size of InstanceBase = 148 Nodes, (2960 bytes), 99.61 % compression

Warning:-test1-Metric set to Overlap for IGTree test.           [Note 2]
Examine datafile gave the following results:
Number of Features: 12
InputFormat       : C4.5


Starting to test, Testfile: dimin.test
Writing output in:          my_first_test
Algorithm     : IGTree

-test1-Tested:      1 @ Tue Jul  8 16:56:44 2003
-test1-Tested:      2 @ Tue Jul  8 16:56:44 2003
-test1-Tested:      3 @ Tue Jul  8 16:56:44 2003
-test1-Tested:      4 @ Tue Jul  8 16:56:44 2003
-test1-Tested:      5 @ Tue Jul  8 16:56:44 2003
-test1-Tested:      6 @ Tue Jul  8 16:56:44 2003
-test1-Tested:      7 @ Tue Jul  8 16:56:44 2003
-test1-Tested:      8 @ Tue Jul  8 16:56:44 2003
-test1-Tested:      9 @ Tue Jul  8 16:56:44 2003
-test1-Tested:     10 @ Tue Jul  8 16:56:44 2003
-test1-Tested:    100 @ Tue Jul  8 16:56:44 2003
```

```
-test1-Ready:     950 @ Tue Jul  8 16:56:45 2003
Seconds taken: 1 (950.00 p/s)
914/950 (0.962105)
```

Notes:

1. The first SetOptions() sets the verbosity with +vDI+DB. The default for v is F so the verbosity should become F+DI+DB The second SetOptions() however sets the verbosity with -vDB, and the resulting verbosity is therefore F+DI
2. 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.
3. The -w2 of the first SetOptions() is overruled with -w3 from the second SetOptions(), resulting in a weighting of 3 or Chi-Square.

Result in my_first_test (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
```

11.2 example 2, api_test2.cxx

This demonstrates IB2 learning.

Our example program:

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

```
int main(){
  TimblAPI *My_Experiment = new TimblAPI( "-a IB2 +vDI+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" );
  exit(0);
}
```

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 :
FLENGTH              : 0
MAXBESTS             : 500
TRIBL_OFFSET         : 0
INPUTFORMAT          : Unknown
TREE_ORDER           : G/V
DECAY                : Z
SEED                 : -1
DECAYPARAM           : 1.000000
EXEMPLAR_WEIGHTS     : -
IGNORE_EXEMPLAR_WEIGHT : +
NO_EXEMPLAR_WEIGHTS_TEST : +
PROBALISTIC          : -
VERBOSITY            : F+DI+DB
EXACT_MATCH          : -
DO_DOT_PRODUCT       : -
HASHED_TREES         : +
GLOBAL_METRIC        : O
METRICS              :
NEIGHBORS            : 1
PROGRESS             : 100000
IB2_OFFSET           : 100
BIN_SIZE             : 20
WEIGHTING            : gr

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

-test2-Phase 1: Reading Datafile: dimin.train
-test2-Start:          0 @ Tue Jul  8 14:07:57 2003
-test2-Finished:     100 @ Tue Jul  8 14:07:57 2003
-test2-Calculating Entropy         Tue Jul  8 14:07:57 2003
Lines of data     : 100
DB Entropy        : 1.7148291
Number of Classes : 5

Feats   Vals    X-square    Variance     InfoGain     GainRatio
    1      3    7.6024645   0.038012322  0.059969698  0.041607175
```

```
    2     16    46.203035   0.11550759   0.31505423   0.14385015
    3     13    54.694253   0.13673563   0.28508033   0.11717175
    4     12    82.839834   0.20709958   0.29549424   0.20791549
    5      3    9.8142041   0.049071021  0.081665925  0.053303154
    6     25    93.713133   0.23428283   0.52671728   0.13537479
    7     15    56.143669   0.14035917   0.35043203   0.10312857
    8     21    49.386590   0.12346647   0.33767640   0.14930809
    9      2    3.1042762   0.031042762  0.031832973  0.033768474
   10     29    148.77401   0.37193502   0.70753849   0.15973455
   11     14    136.42479   0.34106198   0.90357639   0.25337382
   12     22    219.58669   0.54896672   1.3841139    0.34375881

Feature Permutation based on GainRatio/Values :
< 11, 5, 4, 9, 12, 1, 3, 2, 8, 7, 10, 6 >
-test2-Phase 2: Learning from Datafile: dimin.train
-test2-Start:          0 @ Tue Jul  8 14:07:57 2003
-test2-Finished:     100 @ Tue Jul  8 14:07:57 2003

Size of InstanceBase = 991 Nodes, (19820 bytes), 23.77 % compression

-test2-Phase 2: Appending from Datafile: dimin.train (starting at line 101)
-test2-Start:        101 @ Tue Jul  8 14:07:57 2003
-test2-Finished:    2999 @ Tue Jul  8 14:07:58 2003
added 242 new entries                                    [Note 1]

Size of InstanceBase = 2807 Nodes, (56140 bytes), 32.52 % compression

DB Entropy        : 2.06260672
Number of Classes : 5

Feats   Vals   X-square      Variance     InfoGain     GainRatio
    1      3   19.21211054   0.02808788   0.04663668   0.03472093
    2     26   120.99000006  0.08844298   0.21552161   0.09862660
    3     15   95.37436002   0.06971810   0.17130436   0.07700194
    4     16   87.51574055   0.06397349   0.13415866   0.11868667
    5      3   31.00890345   0.04533465   0.07709044   0.04882750
    6     32   142.15589806  0.10391513   0.30555295   0.07911265
    7     18   139.20481395  0.10175790   0.26973542   0.08142467
    8     30   97.89853223   0.07156325   0.18433394   0.09439585
    9      2   26.57144438   0.07769428   0.07776679   0.07804426
   10     42   234.33268308  0.17129582   0.45128191   0.09418234
   11     16   286.34676422  0.20931781   0.63347440   0.17490482
   12     40   751.89378418  0.54962996   1.35537142   0.31983281

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


Starting to test, Testfile: dimin.test
Writing output in:          my_second_test.out
Algorithm     : IB2
Global metric : Overlap
Deviant Feature Metrics:(none)
```

```
Weighting     : GainRatio
Feature 1        : 0.034720933022277
Feature 2        : 0.098626600047048
Feature 3        : 0.077001941190107
Feature 4        : 0.118686673888863
Feature 5        : 0.048827501102016
Feature 6        : 0.079112651941718
Feature 7        : 0.081424668550292
Feature 8        : 0.094395849717083
Feature 9        : 0.078044257394732
Feature 10       : 0.094182343171335
Feature 11       : 0.174904816660632
Feature 12       : 0.319832813396973

-test2-Tested:      1 @ Tue Jul  8 14:07:58 2003
-test2-Tested:      2 @ Tue Jul  8 14:07:58 2003
-test2-Tested:      3 @ Tue Jul  8 14:07:58 2003
-test2-Tested:      4 @ Tue Jul  8 14:07:58 2003
-test2-Tested:      5 @ Tue Jul  8 14:07:58 2003
-test2-Tested:      6 @ Tue Jul  8 14:07:58 2003
-test2-Tested:      7 @ Tue Jul  8 14:07:58 2003
-test2-Tested:      8 @ Tue Jul  8 14:07:58 2003
-test2-Tested:      9 @ Tue Jul  8 14:07:58 2003
-test2-Tested:     10 @ Tue Jul  8 14:07:58 2003
-test2-Tested:    100 @ Tue Jul  8 14:07:58 2003
-test2-Ready:     950 @ Tue Jul  8 14:07:59 2003
Seconds taken: 1 (950.00 p/s)
889/950 (0.935789), of which 15 exact matches       [Note 2]

There were 57 ties of which 48 (84.21%) were correctly resolved
```

Notes:

1. As we see here, 242 entries from the inputfile 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. However, IGtree is both faster and better. Had we used IB1, the score would have been 96.84 %.

11.3 example 3, api_test3.cxx

This demonstrates Cross Validation:

Lets try the following program:

```
#include "TimblAPI.h"
int main(){
  TimblAPI *My_Experiment = new TimblAPI( "-t cross_validate" );
  My_Experiment->Test( "cross_val.test" );
}
```

We create 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 can't 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 gave the following results:
Number of Features: 8
InputFormat       : C4.5

DB Entropy        : 1.1239577
Number of Classes : 3

Feats  Vals  X-square   Variance    InfoGain    GainRatio
  1     31   73.159596  0.96262626  1.0514606   0.21636590
  2     31   76.000000  1.0000000   1.1239577   0.23128409
  3     29   51.875758  0.68257576  0.96606292  0.20546896
  4     30   38.594949  0.50782828  0.76843726  0.16052103
  5     17   43.164545  0.56795455  0.75290857  0.19555822
  6     17   76.000000  1.0000000   1.1239577   0.29193340
  7     18   39.410606  0.51856061  0.86079976  0.21819805
  8     17   21.504545  0.28295455  0.40001621  0.10329552


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
Feature 1        : 0.216365903867902
Feature 2        : 0.231284092171772
Feature 3        : 0.205468961117581
Feature 4        : 0.160521026415060
```

```
Feature 5        : 0.195558220224054
Feature 6        : 0.291933402045204
Feature 7        : 0.218198052865363
Feature 8        : 0.103295521355847

Tested:      1 @ Tue Jul  8 14:39:13 2002
Tested:      2 @ Tue Jul  8 14:39:13 2002
Tested:      3 @ Tue Jul  8 14:39:13 2002
Tested:      4 @ Tue Jul  8 14:39:13 2002
Tested:      5 @ Tue Jul  8 14:39:13 2002
Tested:      6 @ Tue Jul  8 14:39:13 2002
Tested:      7 @ Tue Jul  8 14:39:13 2002
Tested:      8 @ Tue Jul  8 14:39:13 2002
Tested:      9 @ Tue Jul  8 14:39:13 2002
Tested:     10 @ Tue Jul  8 14:39:13 2002
Ready:      10 @ Tue Jul  8 14:39:13 2002
Seconds taken: 1 (10.00 p/s)
8/10 (0.800000)
Examine datafile gave the following results:
Number of Features: 8
InputFormat       : C4.5

DB Entropy        : 1.21081003
Number of Classes : 3

Feats  Vals  X-square     Variance    InfoGain    GainRatio
  1     30   70.81818182  0.93181818  1.08568141  0.22585381
  2     29   73.03896104  0.96103896  1.13831299  0.24042877
  3     25   67.85714286  0.89285714  1.01318437  0.23032898
  4     27   54.45021645  0.71645022  0.93041471  0.20656967
  5     15   35.69696970  0.46969697  0.75002647  0.20299617
  6     15   73.03896104  0.96103896  1.13831299  0.30537853
  7     16   39.89177489  0.52489177  0.83542417  0.22216874
  8     15   21.70253556  0.28555968  0.44603650  0.12394414


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
Feature 1        : 0.225853813146603
Feature 2        : 0.240428769594743
Feature 3        : 0.230328984331396
Feature 4        : 0.206569670528383
Feature 5        : 0.202996166718337
Feature 6        : 0.305378526354465
Feature 7        : 0.222168743865479
Feature 8        : 0.123944137808201

Tested:      1 @ Tue Jul  8 14:39:13 2002
Tested:      2 @ Tue Jul  8 14:39:13 2002
Tested:      3 @ Tue Jul  8 14:39:13 2002
```

```
Tested:      4 @ Tue Jul  8 14:39:13 2002
Tested:      5 @ Tue Jul  8 14:39:13 2002
Tested:      6 @ Tue Jul  8 14:39:13 2002
Tested:      7 @ Tue Jul  8 14:39:13 2002
Tested:      8 @ Tue Jul  8 14:39:13 2002
Tested:      9 @ Tue Jul  8 14:39:13 2002
Tested:     10 @ Tue Jul  8 14:39:13 2002
Ready:      10 @ Tue Jul  8 14:39:13 2002
Seconds taken: 1 (10.00 p/s)
8/10 (0.800000)
Examine datafile gave the following results:
Number of Features: 8
InputFormat       : C4.5

DB Entropy        : 1.10727241
Number of Classes : 3

Feats  Vals  X-square     Variance    InfoGain    GainRatio
  1     33   73.81677019  0.97127329  1.05464083  0.21157263
  2     31   73.81677019  0.97127329  1.05464083  0.21613678
  3     29   70.54192547  0.92818323  0.96924313  0.20527803
  4     29   63.77391304  0.83913043  0.80175556  0.17134265
  5     17   32.59109731  0.42883023  0.73188656  0.19159785
  6     17   73.81677019  0.97127329  1.05464083  0.27609049
  7     17   33.44387755  0.44005102  0.76788828  0.20441129
  8     16   15.45766046  0.20339027  0.30124108  0.08102333


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
Feature 1        : 0.211572632756454
Feature 2        : 0.216136783583368
Feature 3        : 0.205278026077873
Feature 4        : 0.171342647360174
Feature 5        : 0.191597849237134
Feature 6        : 0.276090486447543
Feature 7        : 0.204411289106452
Feature 8        : 0.081023332351012

Tested:      1 @ Tue Jul  8 14:39:13 2002
Tested:      2 @ Tue Jul  8 14:39:13 2002
Tested:      3 @ Tue Jul  8 14:39:13 2002
Tested:      4 @ Tue Jul  8 14:39:13 2002
Tested:      5 @ Tue Jul  8 14:39:13 2002
Tested:      6 @ Tue Jul  8 14:39:13 2002
Tested:      7 @ Tue Jul  8 14:39:13 2002
Tested:      8 @ Tue Jul  8 14:39:13 2002
Tested:      9 @ Tue Jul  8 14:39:13 2002
Tested:     10 @ Tue Jul  8 14:39:13 2002
Ready:      10 @ Tue Jul  8 14:39:13 2002
```

```
Seconds taken: 1 (10.00 p/s)
9/10 (0.900000)
Examine datafile gave the following results:
Number of Features: 8
InputFormat       : C4.5

DB Entropy        : 1.16743672
Number of Classes : 3

Feats  Vals  X-square     Variance    InfoGain    GainRatio
  1     32   73.62500000  0.96875000  1.11480514  0.22602878
  2     32   72.83333333  0.95833333  1.09493968  0.22289880
  3     30   59.37500000  0.78125000  0.96981107  0.20714134
  4     31   63.33333333  0.83333333  1.06217357  0.22200231
  5     16   43.54166667  0.57291667  0.71361797  0.19418620
  6     16   72.83333333  0.95833333  1.09493968  0.29899921
  7     17   37.09523810  0.48809524  0.76845621  0.20564998
  8     15   24.78670635  0.32614087  0.53212858  0.14797537


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
Feature 1        : 0.226028780085344
Feature 2        : 0.222898804190488
Feature 3        : 0.207141340441089
Feature 4        : 0.222002305500848
Feature 5        : 0.194186199995710
Feature 6        : 0.298999207112076
Feature 7        : 0.205649984076972
Feature 8        : 0.147975374298064

Tested:      1 @ Tue Jul  8 14:39:13 2002
Tested:      2 @ Tue Jul  8 14:39:13 2002
Tested:      3 @ Tue Jul  8 14:39:13 2002
Tested:      4 @ Tue Jul  8 14:39:13 2002
Tested:      5 @ Tue Jul  8 14:39:13 2002
Tested:      6 @ Tue Jul  8 14:39:13 2002
Tested:      7 @ Tue Jul  8 14:39:13 2002
Tested:      8 @ Tue Jul  8 14:39:13 2002
Tested:      9 @ Tue Jul  8 14:39:13 2002
Tested:     10 @ Tue Jul  8 14:39:13 2002
Ready:      10 @ Tue Jul  8 14:39:13 2002
Seconds taken: 1 (10.00 p/s)
8/10 (0.800000)
Examine datafile gave the following results:
Number of Features: 8
InputFormat       : C4.5

DB Entropy        : 1.16687507
Number of Classes : 3
```

```
Feats  Vals  X-square     Variance    InfoGain    GainRatio
  1     32   75.32307692  0.94153846  1.06687507  0.21675958
  2     31   77.66153846  0.97076923  1.11687507  0.22924703
  3     29   63.35384615  0.79192308  0.98574726  0.20875948
  4     28   55.36615385  0.69207692  0.82663406  0.17847364
  5     16   35.75641026  0.44695513  0.69800288  0.18631101
  6     16   77.66153846  0.97076923  1.11687507  0.29662222
  7     16   38.67692308  0.48346154  0.78574726  0.21148893
  8     16   19.69692308  0.24621154  0.37663406  0.10195991
```

```
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
Feature 1        : 0.216759580252811
Feature 2        : 0.229247034465546
Feature 3        : 0.208759480370678
Feature 4        : 0.178473640641812
Feature 5        : 0.186311005227154
Feature 6        : 0.296622220584837
Feature 7        : 0.211488928619078
Feature 8        : 0.101959907286607

Tested:      1 @ Tue Jul  8 14:39:13 2002
Tested:      2 @ Tue Jul  8 14:39:13 2002
Tested:      3 @ Tue Jul  8 14:39:13 2002
Tested:      4 @ Tue Jul  8 14:39:13 2002
Tested:      5 @ Tue Jul  8 14:39:13 2002
Tested:      6 @ Tue Jul  8 14:39:13 2002
Tested:      7 @ Tue Jul  8 14:39:13 2002
Tested:      8 @ Tue Jul  8 14:39:13 2002
Ready:       8 @ Tue Jul  8 14:39:13 2002
Seconds taken: 1 (8.00 p/s)
8/8 (1.000000)
```

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 ...