FAMBL 2.3
Reference Guide

Antal van den Bosch
ILK / Computational Linguistics and AI
Tilburg University, The Netherlands
`http://ilk.uvt.nl`

18th November 2004

FAMBL, Family-Based Learner, version 2.3, is a freely available software package for research or educational purposes only.

FAMBL comes WITHOUT ANY WARRANTY. Author nor distributor accept responsibility to anyone for the consequences of using it or for whether it serves any particular purpose or works at all.

©1997–2004 ILK / Tilburg University.

1 FAMBL: Family-Based Learning

This document is intended as a quick reference guide of the FAMBL algorithm and its current implementation, version 2.3. FAMBL is a memory-based learning algorithm that carefully generalizes over instances. It merges nearest-neighbour instances (exemplars) of the same class into generalized exemplars. In FAMBL, these are called families. FAMBL combines earlier approaches to generalized exemplars, NGE (Salzberg, 1991) and RISE (Domingos, 1995), with some new ideas and heuristics.

FAMBL is documented in two papers (Van den Bosch, 1999a, 1999b) that come with the FAMBL software distribution (see next section). The papers describe the core element, generalizing instances into families, that makes FAMBL different from pure memory-based learning. The latter is implemented in the TiMBL software package[1]. FAMBL should be seen as a specialised variant of TiMBL. If you are interested in performing pure memory-based learning without generalized instances or families, TiMBL is the preferred package due to its faster learning speed. If you do wish to use FAMBL for experimentation, you have to install TiMBL: FAMBL makes use of some of TiMBL's efficient procedures in preprocessing data.

FAMBL is a special variant of memory-based learning that has the ability to generalise over instances. FAMBL devotes a special learning phase to converting the original instance base (the starting point for standard memory-based classification) into a set of *families*. Families are merged instances that are closely alike, and that are associated with the same classification. Merging takes the form of making disjunctions of values for those symbolic feature values on which family members mismatch. For numeric values, merging generates sides of hyperrectangles that span between the minimum and maximum values of the merged features. After families are extracted from the original instance base, the latter is discarded and all further classification (e.g. of test material) is based on nearest-neighbour matching with the families. Depending on the type of data, this may allow for considerable memory compression, and sometimes improvements in classification accuracy and speed.

The current implementation of FAMBL is equipped with various optional metrics for feature weighting (including weighting of feature values and feature combinations), symbolic value difference estimation, and exemplar weighting – more than are mentioned and tested in the original papers. This document is intended to describe all options available via command-line arguments. These descriptions tend to refer to background aspects of the FAMBL algorithm, but an attempt is made to make the descriptions generally understandable.

The structure of the document is as follows. First, in Section 2, the install procedure of the FAMBL software is outlined. Section 3 describes the changes made since previous public releases of FAMBL. In Section 4, a list of command-line options is given, with a description of each option. The data format is described in Section 5. Section 6 provides an annotated example of a command line and its effects when FAMBL is run with it. Section 7 notes some limitations and problems of the current program.

[1] The current version, TiMBL main version 5, can be downloaded from `http://ilk.uvt.nl`.

2 Installing the software

FAMBL 2.3 is written in fairly ANSI C. It should compile and run on most UNIX systems. It can be downloaded from

> http://ilk.uvt.nl/downloads/pub/software/Fambl.2.3.tar.gz

After downloading this gzipped tar file, do the following (">" is your shell promt):

1. > tar zxvf Fambl.2.3.tar.gz
2. > cd Fambl.2.3
3. > make

The unpack operation (tar zxvf) makes the directory Fambl.2.3, and installs all source code in that directory. A "make" in that directory makes a "Fambl" executable. Edit the Makefile if you want to change compiler (gcc by default) or optimisation (-O3 by default). Type 'make clean' to remove all make output.

The distribution also includes three PDF files in the `doc` directory, **Fambl-refguide.pdf** (this document), **Fambl-jetai.pdf** (long paper), and **Fambl-icml99.pdf** (short paper), the two aforementioned background papers which outline the learning and classification procedures of FAMBL.

In the `demo` directory, two files are included that serve as example data. In Section 6 we exemplify how FAMBL is run on this data.

3 Changes

Changes between 2.2 and 2.3 are the following:

- Family creation works "from the inside out"; starting with a randomly picked intance, every same-class instance is added to the family, starting from the nearest neighbors and progressing along all neighbors at larger distances, until a different-class nearest neighbor is found. Up to the current version, instances that were generalized into families were effectively taken out of the system. Now, they are only flagged as being included in a family (as it is actually described in the papers), and family creation for other instances is *still halted* by different-class nearest neighbors already captured in a family expression.
- The new `-X` parameter allows instances to be in multiple families.

- The default k in k-NN classification (set with the `-k` command line parameter) is 1. The `-K` parameter, specifying the maximal number of family members, has a default value of 1,000.
- The output file extensions added by FAMBL are equivalent to those used in TIMBL.

Changes between 2.2 and 2.0 included the following:

- The family extraction phase has been reverted to a simple randomized selection of initial instances. FAMBL is not dependent on TIMBL nor does it compute class prediction strength in another way.
- Along overlap and MVDM, Jeffrey Divergence has been added as a similarity function. Jeffrey Divergence is a symmetric variant of Kullback–Leibler divergence.
- Many internal code optimizations ensure that FAMBL operates considerably faster, especially with the MVDM and Jeffrey Divergence functions, as compared to previous versions.

Changes from 1.01 (the first public releese) to version 2.0 were the following:

- Most prominently, the two-stage learning procedure (consisting of a probing phase and a family extraction phase in version 1.01) has changed:
  1. First, the TIMBL software is called to learn the selected training set, and do a classification on the training set itself with $k = 2$. On the basis of the output of this experiment, the *class prediction strength* of each training instance is computed. This is done with Laplace correction: $cps_i = \frac{(N_i - Ndif_i)+1}{N_i+c}$, where N_i is the number of occurrences that instance i is a nearest neighbor of another instance or itself (the latter always happens once), and $Ndif_i$ is the number of occurrences in which i was a nearest neighbor while having a different class as compared to its neighbor, and c is the number of classes.
  2. Then, all instances are placed in reverse CPS ordering. Starting with the instance with the highest CPS, instances are picked one by one to be the starting point of a new family. Again, family limits are set to $k = 2$, so as to not overestimate the CPS scores obtained in the first stage. Essentially this is a way to keep FAMBL "careful" in the sense of the earlier version also described in the papers.
- The treatment of atomic feature values as individual, weightable features has been fully implemented and debugged, where this was only an experimental option in version 1.01. As explained in more detail in Section 4, it is now possible to declare all feature values in a multi-valued instance base as binary. Additionally, it is possible to look for combinations of feature values that have a weight higher than the sum of the individual weights of its elements, up to a user-defined cardinality.

4 Command-line options

For each option, a table displays (i) the option letter, (ii) its optional arguments, usually integers within a range, of which one is the default value, and (iii) a short description. Underneath each table, a longer description of the option and its values is given.

The **-f** option is obligatory, and has one argument (the name of the file containing labeled training examples). All other options can be set by the user, and if they are not set, the default value of that argument is selected.

`-f`	*filename*	specify the name of the file containing training examples

With **-f filename** the file containing the learning material is declared. This file must be present.

`-t`	*filename*	evaluate FAMBL on unseen cases in file *filename*

Classification performance of FAMBL after family extraction is evaluated by classifying all instances in *filename*. When a test file is not specified, the learning data (set with **-t**) is used as test material.

`-g`	[0,5] (default 2)	set feature weighting metric

-g sets the feature weighting metric w_i used in the basic k-NN distance function, which computes the distance between the instance or family$X$ and the instance Y, $\Delta(X,Y) = \sum_{i=1}^{n} w_i \; \delta(x_i, y_i)$, where n is the number of features, and x_i is the value of the ith feature of instance X. FAMBL currently hosts six options:

-g 0 All features have the same weight $w_i = 1$.

-g 1 *Information gain weighting*. The information gain of feature f is the difference in instance-base entropy (scrambledness of information) between the situations without and with knowledge of the value of that feature: $w_f = H(C) - \sum_{v \in V_f} P(v) \times H(C|v)$, where C is the set of class labels, V_f is the set of values for feature f, and $H(C) = -\sum_{c \in C} P(c) \log_2 P(c)$ is the entropy of the class labels. The probabilities are estimated from relative frequencies in the training set.

-g 2 (default) *(Information) gain ratio weighting* (Quinlan, 1993). The gain ratio of a feature f is its information gain divided by its split info, $si(f) = -\sum_{v \in V_f} P(v) \log_2 P(v)$, the entropy of its values. Dividing information gain by split info avoids a bias in favor of features with more values.

-g 3 *Chi-square weighting*, giving for each matrix between feature values and classes a global number χ^2 for the level of surprise within the matrix: the result is the sum of squares of all differences between the expected values in each cell (based on averaging over rows and columns) with the observed values. See any basic statistics book.

-g 4 *Shared variance weighting*, derived from the chi-square value χ of the matrix between feature values and classes. Shared variance of a feature expresses in a percentage how much that feature explains the classification of all data. Shared variance is $\frac{\chi}{N(L-1)}$ where N is the number of instances, and L is the lesser of the number of feature values or the number of classes.

-g 5 *Log-likelihood weighting* expresses, not unlike chi-square and shared variance, the deviation of the cooccurrence of two events between the virtual situation in which they are completely independent, and the actual situation in which they may be strongly dependent. Unlike chi-square and shared variance, log-likelihood goes for two observations only (in this case, one value and one class). Therefore log-likelihood is used cumulatively here; i.e., the result per feature is an accumulation of log-likelihoods between all values and classes. The maximum value is normalised to 1.0. For a detailed treatment of log-likelihood, including an example computation, see Dunning (1993).

`-m`	[0-2] (default 0)	set value difference metric

-m sets the value-difference function $\delta(x_i, y_i)$, the distance between the values of feature i in instances X and Y, in the $\Delta(X, Y)$ function that sums these distances over all features (cf. **-g**). Three options are available:

-m 0 (default value) *Overlap metric* (also known as Hamming distance, Manhattan metric, or L1 metric). For nominal data, $\delta(x_i, y_i)$ simply returns a distance of 1 with a mismatch, and a 0 with a match. With numeric values, the difference is scaled by the minimal and maximal values found for that feature. In other words,

$$\delta(x_i, y_i) = \begin{cases} \frac{x_i - y_i}{max_i - min_i} & \text{if numeric, else} \\ 0 & \text{if } x_i = y_i \\ 1 & \text{if } x_i \neq y_i \end{cases}$$

-m 1 *Modified value difference metric* (Stanfill and Waltz, 1986; Cost and Salzberg, 1993). This metric, usually referred to as MVDM, estimates a numeric difference between two feature values from the difference of their cooccurrences with the classes of the training data. The distance between two values V_1, V_2 of a feature is computed by taking the difference of the conditional distributions of the classes C_i for these values: $\delta(V_1, V_2) = \sum_{i=1}^{n} |P(C_i|V_1) - P(C_i|V_2)|$. With many values, keeping all MVDM matrices in memory becomes computationally disadvantageous. FAMBL determines by itself whether it will prestore MVDM matrices, or whether it will compute MVDM values on the fly.

-m 2 *Jeffrey Divergence* is a statistical dissimilarity metric that can be used to compute the distance between class distributions of two values of the same feature. Functionally it is quite similar to MVDM. It is best known for its application as a distance function in unsupervised vector space models, e.g. in image retrieval, where it is applied to histogram vectors. While MVDM computes a straightforward geometrical distance between two class distribution vectors, Jeffrey divergence introduces a logarithm term: $\delta(v_1, v_2) = \sum_{i=1}^{n}(P(C_i|v_1) log \frac{P(C_i|v_1)}{m} + P(C_i|v_2) log \frac{P(C_i|v_2)}{m})$ Jeffrey divergence is a symmetric variant of Kullback–Leibler distance; the m term is used for this purpose: $m = \frac{P(C_i|v_1)+P(Ci|v_2)}{2}$

`-d`	[0-3] (default 0)	set distance weighting metric

The **-d** parameter sets the distance weighting metric used in the class voting stage of the k-NN classifier. By default, it is switched off; the three available distance weighting functions all assign a relatively smaller weight (or vote) to more distant nearest neighbors in the nearest neighbor set.

-d 0 (default value) Assign equal weight (1) to all nearest neighbors in the nearest neighbor set, when determining the majority class in the set.

-d 1 *Linear-inversed distance weighting*. Nearest neighbors with smaller distances are weighted more heavily than ones at larger distances: the nearest neighbor gets a weight of 1, the furthest neighbor a weight of 0 and the other weights are scaled linearly to the interval in between, according to $w_j = \begin{cases} \frac{d_k - d_j}{d_k - d_1} & \text{if } d_k \neq d_1 \\ 1 & \text{if } d_k = d_1 \end{cases}$ where d_j is the distance to the j'th nearest neighbor, d_1 the distance of the nearest neighbor, and d_k of the furthest (k'th) neighbor.

-d 2 *Inverse distance weighting*. Each nearest neighbor is assigned a weight that is the inverse of its distance: $w_j = \frac{1}{d_j}$. A small constant is added to the denominator to avoid division by zero.

-d 3 *Exponential-decay distance weighting*. An exponential decay function determines the slope of the decrease in voting weight: $w_j = e^{-\alpha d_j^{\beta}}$. In the current implementation, both α and β are set to 1.

`-a`		regard values of multi-valued features as atomic

The **-a** switch "unpacks" multi-valued features into arrays of atomic, or binary, features, where each binary feature denotes the presence (value 1) or absence (value 0) of a particular value in an instance. The weighting metric set by the **-g** switch also calculates weights for each of these individual features. Hypothetically, assigning weights to atomic features can be better than assigning weights to groups of values in the default situation, since some feature values may be more important than others. On the other hand, the number of observations on which a weight is based can become dramatically smaller, and the weight can be accordingly bad when used in classifying new material. See Van den Bosch and Zavrel (2000) for a survey of the effects of unpacking feature values.

`-C`	[1,32] (default 1)	set cardinality threshold of atomic feature combination search

When the **a** switch is selected, it is assumed by default that atomic feature values should be discerned and weighted individually. Setting **-C** to higher values will make FAMBL search for *combinations* of atomic feature values, of cardinality **C**, that have a weight (set by the **-g** switch) that is *higher than the sum of the weights of the composing parts*. Combinations are only made between atomic features from different original multi-valued features.

At cardinality 2, all possible combinations are tested. At higher cardinalities c, only those compound features are tested that include a strong compound at the previous cardinality $c - 1$.

This option works very differently with different weighting metrics (set with **-g**). Good results (no dramatic explosion of numbers of compund features, retaining generalisation accuracy) are generally obtained with shared variance and chi square weighting.

Needless to say, this option slows down FAMBL both in family extraction and classification. Even more needless: setting **-C** has effect only when **-a** is selected.

`-w`	[0,2] (default 0)	set family weighting metric

Families can be weighted individually with a weighting factor e, an extra factor in the function that computes the distance between a family X and an instance to be classified Y: $\Delta(X,Y) = e_X \sum_{i=1}^{n} \; w_i \; \delta(x_i, y_i)$. This makes sense when it is clear from the learning material alone that certain families are bad predictors of their own class, or very good ones. The bad ones should be suppressed by low e in the distance function; the good ones should be boosted. FAMBL allows the following variations on e:

-w 0 (default value) All families have the same weight: for each family i, $e_i = 1.0$.

-w 1 *Class prediction strength weighting* (Salzberg, 1991). The class prediction strength of a family i is the number of times the family is a nearest neighbor of a training instance regardless of its class (N), minus the number of these nearest neighbours that are of a different class ($Ndif_i$), divided by N to express a portion between 0.0 and 1.0: $e_i = \frac{N_i - Ndif_i}{N_i}$. A family with class-prediction strength $e = 1.0$ is a perfect predictor of its own class; an e near or at 0.0 indicates that the family is a bad predictor.

-w 2 *Class prediction strength weighting with Laplace correction* (Nibblet, 1987; Clark and Nibblet, 1989; Domingos, 1996). Raw class prediction strength has a bias towards low-frequent families and instances that is sometimes unwanted. The Laplace correction introduces the number of classes, c, into the equation: $e_i = \frac{(N_i - Ndif_i)+1}{N_i + c}$. All other things being equal, a larger absolute $N_i - Ndif_i$ leads to higher e.

`-b`		perform basic MBL (IB1), i.e., create only single-instance families

-b offers a sort of "backward" compatibility with pure memory-based learning. It shuts down the family extraction stage, and simply takes each individual instance (or instance type, if identical instance tokens occur) to be its own family. It should be noted that this option is generally slower than the implementation of memory-based learning in TiMBL. There are small differences in Fambl's implementation and TiMBL's, particularly in resolving class vote ties, so that performances of FAMBL with **-b** might well deviate from TiMBL's on the same data with the same algorithmic parameter settings.

-k	[1> (default 1)	k in k-NN classification

Although families in FAMBL represent a kind of locally optimised "precompiled" k, which implies that k in k-NN classification is best set to 1, the **-k** option offers the possibility to allow matching to be based on more than one family (or rather more than one "distance bin", as each k is seen as a bin in which all families at a similar distance are grouped).

-K	[1> (default 1000)	k in family extraction

By default, FAMBL extracts families without any bounds. That is. after having selected a starting instance for a family randomly, FAMBL builds the family expression by searching for all same-class nearest neighbors until it finds a different-class neighbor, or no more instances are available. Setting -K to lower values implies that the family extraction procedure generates more instance-specific families; abstraction becomes more "careful". Setting -K to 0 actually means emulating normal IB1, without any family extraction (see the -b switch).

-L	[1> (default 1)	threshold to back off from MVDM to Overlap

MVDM is known to produce bad similarity estimates for pairs of values of which one or both are low-frequent. If one of both of the values in an MVDM compution occur less than the threshold set by **-L**, a switch is made to the Overlap function. Setting **-L** higher than 1, e.g. at 2, activates the back-off. Values of 2 to 10 would be suggested by statistical theory.

-u	*typestring*	sets feature types according to typestring

Features can be either symbolic or numeric. Their type has considerable effects on the metrics than can and cannot be applied to them (e.g., MVDM is designed for symbolic features). FAMBL selects the appropriate metrics, but needs to know first if a feature is symbolic or numeric. This is encoded in the "typestring" string. This string must be a sequence of "s"s and "n"s; each nth "s" (symbolic) or "n" (numeric) denotes the type of the nth feature. For example, setting **-t nnnss** declares the first three features to be numeric, and the final two features to be symbolic.

-n		declare all feature values as numeric

-n is the shortcut for declaring all features to have numeric values. If neither **-t** or **-n** is set, FAMBL assumes all features are symbolic.

`-j`		use joker (wildcard) values instead of value disjunctions (symbolic features)

Merging instances in FAMBL means, by default, that mismatching symbolic values are joined in a disjunction in the resulting family expression. The alternative **-j** introduces a "joker value", a wildcard, that replaces any disjunction of two or more values of a merged feature (as in RISE; Domingos, 1996). In classification, the wildcard matches with any value. Usually this option leads to high memory compression rates, not seldom at the cost of generalization accuracy losses, because often individual values shouldn't be "forgotten", apparently.

`-x`	[2> (default 1,000)	collapse disjuncts to wildcards above n values

Very long disjuncts at one feature in a family expression may indicate that at that position it is actually not important what value occurs; rather than an exhaustive list of values, a wildcard (joker) value may be more appropriate for generalization. As a less absolute method than the **-j** option, with the **-x** option it is possible to preset a threshold of a number of values above which the disjunction is replaced by a wildcard.

`-X`	[2> (default 1,000)	allow instances to be in multiple families

By default, no instance can be a member of more than one family. This can be switched off by specifying **-X**. When switched on, FAMBL will be unable to attain any level of compression; many families will represent overlapping subsets of instances redundantly.

`-c`		collapse hyperrectangles to into centroids (numeric features)

Hyperrectangles, the geometrical term for families of which the features are numeric, are parts of feature space within which the distance to instances falling in them is zero (Salzberg, 1991). With **-c**, hyperrectangles are collapsed into their geometrical mean, the centroid, so that instances that are within the original hyperrectangle space but not at the centroid are regarded as having a non-zero distance to the family. This option usually compresses memory.

`-D`		print NN distances along with classification output

At the end of each line in the automatically-generated output file *filestem.test...out*, containing a test instance, its actual class, and its predicted class, the distance of the nearest neighbour is printed. If this distance is 0.0, at least one of the nearest neighours was an exact match of the test instance.

`-v`	[0,4] (default 1)	set verbosity level

-v sets the level of verbosity of FAMBL's output. At **-v0**, only a welcome message and the resulting generalisation score is sent to `stdout`. At **-v2** and **-v3**, incrementally more data is printed during both learning and classification, but also on the identity of the families being created. At **-v4** FAMBL is producing kilobytes of unintelligible debug-oriented information.

`-o`		write binary Fambl file filestem.fambl

For backup purposes, **-o** writes the file *filename*.fambl to disk, where *filename* is the name of the training file, immediately after the family extraction stage. It can be read back in a next session with **-e**. The file is in a binary format.

`-e`		read binary file filestem.fambl

-e reads the file *filename*.fambl back into memory, skipping the family extraction stage. *Filename* is the name of the original training file, which does not need to be present when reading the FAMBL file.

`-r`	[1> (default 1000)	rate at which to report on progress on processing families or test instances

-r sets the rate at which FAMBL reports on how many families are extracted, or how many test instances are processed: around every manyfold of the number set after **-r**. Progress reports on reading data files and computing class-prediction-strength values during family extraction is done at a rate 100 times this number.

5 Data format

The training and test sets fed into FAMBL consist of descriptions of instances in terms of a fixed number of feature-values. The files should contain one instance per line. The number of instances, features, feature values and classes in the data are determined automatically. The last feature of the instance is assumed to be the target category. During testing, FAMBL writes the classifications of the test set to an output file. The format of this output file is the same as the input format, with the addition of the predicted category being appended after the correct category.

The data format demanded by FAMBL is a derivative of the format that is used by the well-known C4.5 decision tree learning program (Quinlan, 1993). The separator between the feature values can be commas or white space (spaces or tabs). The class label (viz. the last feature on the line) may be followed by a period, but this is not mandatory. Note that FAMBL does not require a so called *namesfile*.

Example data files that come with the FAMBL distribution, in the `demo` directory, are **example-grapho.data** and **example-grapho.test**. The top of **example-grapho.data**, displayed below, exemplifies the comma-separated one-instance-per-line C4.5 format.

```
_,_,_,_,a,b,e,r,r,(.
_,_,_,a,b,e,r,r,a,b.
_,_,a,b,e,r,r,a,t,^.
```

```
_,a,b,e,r,r,a,t,i,r.
a,b,e,r,r,a,t,i,o,-.
b,e,r,r,a,t,i,o,n,1.
e,r,r,a,t,i,o,n,_,-.
r,r,a,t,i,o,n,_,_,S.
r,a,t,i,o,n,_,_,_,-.
a,t,i,o,n,_,_,_,_,H.
_,_,_,_,a,b,o,v,e,^.
_,_,_,a,b,o,v,e,m,b.
_,_,a,b,o,v,e,m,e,V.
_,a,b,o,v,e,m,e,n,v.
a,b,o,v,e,m,e,n,t,-.

...
```

The examples displayed here represent parts of words, in the form of fixed windows of nine letters. The tenth field, i.e., the final comma-separated column, represents the pronunciation (in phonemic code) of the middle letter. More generally, the last field always represents the class; all other columns represent features.

6 An annotated example run

Using the example data files **example-grapho.data** and **example-grapho.test** (included in the package's `demo` directory), FAMBL is run with the following commandline:

```
> Fambl -f example-grapho.data -t example-grapho.test
```

This means that FAMBL will learn from **example-grapho.data**, and test on **example-grapho.test**. The full log resulting from this command is the following:

```
Fambl (Family-based learning), version 2.3, 17 November 2004
(c) 1997-2004 ILK Research Group, Tilburg University
http://ilk.uvt.nl / antalb@uvt.nl
current time: Thu Nov 18 15:16:34 2004
     metric scheme set to GR feature wgts, no distance wgts, MVDM
     sorting and reading data base example-grapho.data
     data base has 6124 instances
                   5971 instance types
                   9 features
                   57 classes
     computing feature gain ratio values
     feature   0 (   28 values):   0.078196
     feature   1 (   28 values):   0.085658
     feature   2 (   30 values):   0.100040
     feature   3 (   31 values):   0.190212
     feature   4 (   30 values):   0.720676
     feature   5 (   31 values):   0.224557
     feature   6 (   30 values):   0.127532
     feature   7 (   29 values):   0.095540
     feature   8 (   29 values):   0.087888
     average gain ratio: 0.190033
     presorting and rereading instance base example-grapho.data
     computing MVDM matrices
     took 4 seconds
 <*> family extraction stage started
       1000 families (covering 73.42%),    4.50 members av.
       2000 families (covering 96.21%),    2.95 members av.
     # families                        :      2222
     average # members                 :    2.7561
     average k distances               :    2.6854
     average description length (bytes):  102.7381
     compression (raw memory)          :    6.8076 %
     compression (vs instance types)   :   13.1088 %
     #type vs. #fam reduction          :   62.7868 %
     clusteredness                     :    140.24
     took 9 seconds
 <*> starting test with k=1
```

```
     writing output to example-grapho.test.Fambl.M.gr.k1.K1000.out
         533 instances out of     657 classified correctly
     Fambl score:  81.1263 % correct instances
     took 2 seconds
     (328.50 instances per second)
 <*> current time: Thu Nov 18 15:16:49 2004
     Fambl spent a total of 15 seconds running;
     13 on learning, 2 on testing.
     Fambl ready.
```

This log should be read as follows:

```
Fambl (Family-based learning), version 2.3, 17 November 2004
(c) 1997-2004 ILK Research Group, Tilburg University
http://ilk.uvt.nl / antalb@uvt.nl
current time: Thu Nov 18 15:16:34 2004
     metric scheme set to GR feature wgts, no distance wgts, MVDM
     sorting and reading data base example-grapho.data
     data base has 6124 instances
                   5971 instance types
                   9 features
                   57 classes
     computing feature gain ratio values
     feature   0 (   28 values):   0.078196
     feature   1 (   28 values):   0.085658
     feature   2 (   30 values):   0.100040
     feature   3 (   31 values):   0.190212
     feature   4 (   30 values):   0.720676
     feature   5 (   31 values):   0.224557
     feature   6 (   30 values):   0.127532
     feature   7 (   29 values):   0.095540
     feature   8 (   29 values):   0.087888
     average gain ratio: 0.190033
     presorting and rereading instance base example-grapho.data
     computing MVDM matrices
     took 4 seconds
```

FAMBL starts with an echo of its name and version, and a time stamp. It reports on the selected metrics (either by default or by user override). Since the command line did not specify any metrics, FAMBL reports it will use information gain ratio (GR) feature weighting and the overlap metric. It then proceeds to read the training file **example-grapho.data**, finding out how many instances (both instance tokens and types), features, feature values, and classes are in this file. Having established that, FAMBL computes the information gain ratio feature weights of the nine features. The final initialisation step is to re–sort and re–read the data into memory. This completes all initialisation, taking 4 seconds. FAMBL then starts the family extraction stage:

```
 <*> family extraction stage started
       1000 families (covering 73.42%),    4.50 members av.
       2000 families (covering 96.21%),    2.95 members av.
     # families                        :      2222
     average # members                 :    2.7561
     average k distances               :    2.6854
     average description length (bytes):  102.7381
     compression (raw memory)          :    6.8076 %
     compression (vs instance types)   :   13.1088 %
     #type vs. #fam reduction          :   62.7868 %
     clusteredness                     :    140.24
     took 9 seconds
```

FAMBL reports at each manifold of 1,000 extracted families how much of the original instance base is being covered already by extracted families, and how many members on average are in the extracted families. Since the largest families are extracted first generally, this average tends to drop during extraction. Upon completion, FAMBL reports on the number of families (in this example, 2,222); the average number of family members (over 2.75), the average length in bytes needed to store the families in memory (103), and three compression rates:

1. *compression (raw memory)* is the compression rate in memory needed for storing all families, as compared to storing all instances in pure memory-based learning. This is the most concrete compression rate, which is not seldom negative because of the fact that storing families involves exceedingly more bookkeeping (e.g., storing disjunctions rather than singular values) than storing instances for which the number of features and values is fixed and known.

2. *compression (vs. instance types)* compares FAMBL's storage requirements against the requirements of storing instance types with frequency counts.

3. *#type vs. #fam reduction* is the percentage reduction of the number of families versus the number of instance types. This is never negative, and provides generally the most flattering compression figures, abstracting of course over the actual numbers of bytes it takes to store families versus instance types.

FAMBL then reports on *clusteredness*, which is the weighted average of the number of clusters in which classes are scattered. The higher this number, the more disjunct a data set is. This concludes the family extraction stage, which took 9 seconds. The subsequent test stage is then started:

```
<*> starting test with k=1
    writing output to example-grapho.test.Fambl.M.gr.k1.K1000.out
        533 instances out of     657 classified correctly
    Fambl score:  81.1263 % correct instances
    took 2 seconds
    (328.50 instances per second)
```

The example test file **example-grapho.test** contains 657 instances, of which 533 are classified correctly, yielding an accuracy percentage of 81.1%. Testing took two seconds. All output of FAMBL is written during testing to a file called **example-grapho.test.Fambl.O.gr.k1.K1000.out** (**O**: weighted overlap, **gr**: information gain ratio, **k1**: $k = 1$, **K1000**: $K = 1000$). The top of this file looks as follows:

```
_,_,_,_,e,c,l,a,i,1,n
_,_,_,e,c,l,a,i,r,k,k
_,_,e,c,l,a,i,r,_,l,l
_,e,c,l,a,i,r,_,_,8,1
e,c,l,a,i,r,_,_,_,-,-
c,l,a,i,r,_,_,_,_,R,R
_,_,_,_,a,e,r,o,p,8,-
_,_,_,a,e,r,o,p,l,-,-
_,_,a,e,r,o,p,l,a,r,r
_,a,e,r,o,p,l,a,n,^,Q
a,e,r,o,p,l,a,n,e,p,p
e,r,o,p,l,a,n,e,s,l,l
r,o,p,l,a,n,e,s,_,1,(
o,p,l,a,n,e,s,_,_,n,n
p,l,a,n,e,s,_,_,_,-,-

...
```

This file provides all test instances, with an additional eleventh column containing the class predicted by FAMBL.

```
<*> current time: Thu Nov 18 15:16:49 2004
    Fambl spent a total of 15 seconds running;
    13 on learning, 2 on testing.
    Fambl ready.
```

When testing is finished, FAMBL summarises how much elapsed wall clock time was spent on learning (reading data plus extracting families) and testing, and exits.

7 Limitations

In FAMBL's implementation certain choices were made that limit certain aspects of the data that can be handled by FAMBL. All restrictions are set in the `#define`s in the file `Common.h`, and may be changed before compilation by the user at own risk. In short, the restrictions are the following:

- the number of features may not exceed 2048.
- the number of feature values may not exceed `INT_MAX`. This is more than usually necessary. To save some memory, provided that all of the features in your dataset have less than 32,768 values, you might change the type definition of `value` from `int` to `short int`.
- the number of classes is not allowed to exceed 8192.
- FAMBL cannot extract more than 1,500,000 families.

Finally, the following list of problems is known to occur. In general, please take care (especially of available disk space) when applying FAMBL to very big data files. Please send bug reports to `Antal.vdnBosch@uvt.nl`.

- FAMBL writes temporary files in the current working directory. When FAMBL is interrupted before family extraction, all temporary material is left undeleted. Be sure to delete any `fambl*tmp` files after interrupted runs.
- Some number overflow checks may not be waterproof yet. Extreme data may prove this. The biggest data set to date tested successfully contained 2.5 million cases with 11 symbolic features.

Acknowledgements

Thanks go out to Walter Daelemans, Jakub Zavrel, Iris Hendrickx, Ton Weijters, Eric Postma, Jorn Veenstra, Sabine Buchholz, Bertjan Busser, Ko van der Sloot, Hans van Halteren, Gert Durieux, Stephan Raaijmakers, Jaap van den Herik, David Aha, Pedro Domingos, Dietrich Wettschereck, and Fred Wan. Thanks to the Tilburg ILK group for comments, moral and technical support.

References

Aha, D. W., D. Kibler, and M. Albert (1991). Instance-based learning algorithms. *Machine Learning*, 6:37-66.

Clark, P. and T. Niblett. (1989). The CN2 rule induction algorithm. *Machine Learning*, 3:261–284.

Cost, S. and S. Salzberg (1993). A weighted nearest neighbour algorithm for learning with symbolic features. *Machine Learning*, 10:57-78.

Cover, T. M. and P. E. Hart (1967). Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 13:21-27.

Daelemans, W., A. van den Bosch, and A. Weijters (1997). IGTree: using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, 11:407-423.

Daelemans, W., A. van den Bosch, and J. Zavrel (1997). A feature-relevance heuristic for indexing and compressing large case bases. In M. van Someren and G. Widmer, *Poster Papers of the Ninth European Conference on Machine Learing*, University of Economics, Prague, Czech Republic, pages 29-38.

Daelemans, W., A. van den Bosch, and J. Zavrel (1999). Forgetting exceptions is harmful in language learning. *Machine Learning*, 11:11-43.

Domingos, P. (1996). Unifying instance-based and rule-based induction. *Machine Learning*, 24:141–168.

Dunning, T. (1993). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19:1, 61–74.

Niblett, T. (1987). Constructing decision trees in noisy domains. In *Proceedings of the Second European Working Session on Learning*, Bled, Yugoslavia, pp. 67–78.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

Salzberg, S. (1991). A nearest hyperrectangle learning method. *Machine Learning*, 6:277–309.

Stanfill, C. and D. Waltz (1986). Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228.

Van den Bosch, A (1997). *Learning to pronounce written words. A study in inductive language learning*. Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands.

Van den Bosch, A. (1999a). Instance-family abstraction in memory-based language learning. In I. Bratko and S. Dzeroski (Eds.), *Machine Learning: Proceedings of the Sixteenth International Conference*, ICML'99, Bled, Slovenia, pp. 39–48.

Van den Bosch, A. (1999b). Careful abstraction from instance families in memory-based learning. To appear in W. Daelemans (guest ed.), *Journal of Experimental and Theoretical Artificial Intelligence*, special issue on memory-based learning of language processing.

Van den Bosch, A. and Zavrel, J. (2000). Unpacking multi-valued symbolic features and classes in memory-based language learning. In P. Langley (Ed.), *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 1055-1062. San Francisco, CA: Morgan Kaufmann, 2000.