Getting Started with ROSANA-ML

Roland Stuckardt
D-60433 Frankfurt am Main, Germany
e-mail: roland@stuckardt.de

10th April 2003

1 Synopsis

This document describes how to install and use the ROSANA-ML System (Version 5.1, October 2002) and the sample data (texts, parses, keys, feature vector signatures) under a Linux environment. How to install ROSANA is described in detail in the document [4]; in the document at hand, supplementary information regarding the specific aspects of the installation and usage of the machine-learning descendant ROSANA-**ML** is given.

2 Licensing conditions

ROSANA-ML and ROSANA are made available on condition that the user unconditionally accepts the "**License Agreement for ROSANA-ML and ROSANA**" as stated in appendix A of the document at hand. In using ROSANA-ML and/or ROSANA or any component that comes along with the ROSANA-ML or ROSANA distribution, the user acknowledges that he/she has read through and understood the License Agreement for ROSANA-ML and ROSANA and unconditionally accepts it.

3 Prerequisites

- Common Lisp (e.g., Xanalys Lispworks 4.2.0 for Linux, or Allegro Common Lisp of Franz, Inc.)
- a reasonably equipped PC (500 MHz, 128 MB RAM should do)
- basic knowledge of Lisp
- basic knowledge of machine-learning methods as described, e.g., in [2] (particularly, regarding decision-tree learning approaches and Quinlan's C4.5 algorithm [3])
- the C4.5 implementation [1] of the University of Regina[1]

[1] Other C4.5 implementations should work as well; however, the decision-tree input frontend of ROSANA-ML would have to be rewritten.

- in-depth knowledge of anaphor resolution and the respective formal evaluation issues (cf., e.g., [5])
- basic knowledge regarding machine-learning approaches to anaphor resolution; it will be repeatedly referred to the paper [6] which gives a detailed description of the ROSANA-ML approach and its evaluation results
- the willingness as well as the ability to cope with an experimental software system for which virtually no documentation is available

4 Installation

The basic installation steps are identical to those of the ancestor ROSANA (cf. [4]), the only difference being the name of the directory to which the gzipped distribution file unpacks to (`ROSANAML51` instead of `ROSANA51`). Hence, if the installation directory `$ROSANAPATH = ~/CommonLisp` , then the constant ROSANAHOME will have to be set to

```
(defconstant ROSANAHOME "~/CommonLisp/ROSANAML51")
```

5 Using ROSANA-ML under Xanalys Lispworks

If you are using Xanalys Lispworks 4.2.0 for Linux, a simple graphical interface is provided.

1. *Starting the graphical version of ROSANA-ML:* load the file *capi-rosanaml.lisp.*

   ```
   CL-USER 2 > (load "capi-rosanaml")
   ```

   During first-time use, a copyright and licensing message will be displayed. Please read carefully through the "**License Agreement for ROSANA-ML and ROSANA**" that comes along with this distribution (cf. appendix A). Upon acceptance of the licensing conditions, ROSANA-ML will be loaded and initialized, and a frame titled ROSANA-ML will be opened which displays the ROSANA-ML listener. The ROSANA-ML frame offers some basic means to process the sample data and to modify the processing, trace settings, and machine-learning settings of ROSANA-ML.

Whereas the above first step is virtually identical to the startup procedure of ROSANA, the subsequent steps are specific to ROSANA-ML because they reflect the underlying machine-learning methodology, which distinguishes between two basic cases: the training phase and the application phase (cf. figure 1; the reader is referred to the in-depth discussion in [6]). The graphical user interface of ROSANA-ML provides two additional menus through which the main parameters of the decision tree learning procedure may be configured according to the specific experimental requirements. In the `ML Signature` menu, different pre-defined *feature vector signatures* are offered (cf. section 11 for a more comprehensive discussion, including an outline how users may define and employ additional signatures that are tailored to their specific experimental requirements). Menu `ML Settings` offers certain options regarding the generation of training data, e.g. whether anaphor-type-specific or general

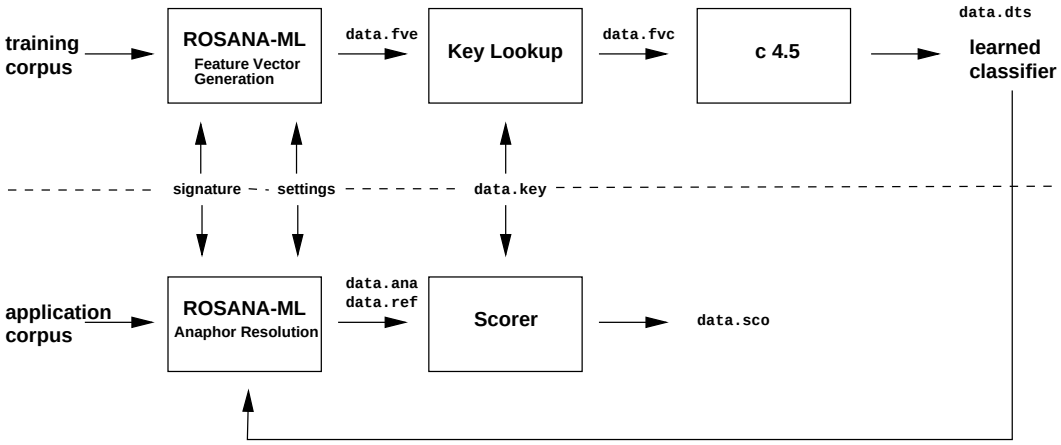

Figure 1: ROSANA-ML: training vs. application case

classifier functions should be learned (cf. [6] and the more comprehensive discussion in section 10).
For expository purposes, let's assume that a new decision tree classifier function shall be learned. Let's assume further that a *general* classifier function, which deals with third-person possessive and non-possessive pronouns, shall be learned, and that training data shall be generated with the `no cataphors` and the `no recency limit` options. In the `ML Settings` menu, whereas the former (general classifier) option is already selected in the startup configuration, the latter choice requires selecting the `no recency limit` option by hand since only the `no cataphors` box is initially selected. In addition, let's assume that signature `sig 6` shall be employed; perform the respective selection in the `ML Signature` menu.[2] We are now ready to generate the training data for decision tree (C4.5) learning.

2. *Training case - generating training data:* in menu `Train & Resolve`, select the abbreviated sample feature vector generation call `FV Gen 6sig PressReleases [ 1-31 ]`. This will paste a function call into the ROSANA-ML listener buffer. Execute this call in the ROSANA-ML listener:

   ```
   SEM 1 > (FVG-6sig-PressReleases-1-31)
   ```

   This invokes the feature vector generation process. The results are written to the respective directory of the selected signature. In the above case, the name of the directory is `$ROSANAPATH/ROSANAML51/Data/PressReleases/6sig/1-31/` . Cf. section 8 for an in-depth discussion of the employed (and, in fact, recommended) directory structure.
   Alternatively, one may use the feature vector generation interface function *dpe-anares-fv-generierung*, which makes available the full set of options (cf. section 12).

[2] Actually, when employing the predefined sample calls of the `Train & Resolve` menu, it is automatically taken provision for that the respective signatures will be selected. The global selection, as performed in the `ML Signature` menu, is not affected by these local redefinitions.

3. *Browsing the generated output files:*

   (a) Switch to the directory of the processed sample input suite:

   ```
   > cd $ROSANAPATH/ROSANAML51/Data/PressReleases/6sig/1-31/
   ```

   (b) Examine the feature vector output files which have been generated by ROSANA-ML:

   - *pr.fve*: this file contains the generated feature vectors, including the numbers of the respective anaphor and antecedent candidate occurrences over which the individual vectors have been computed;
   - *pr.fvc*: this file contains the training cases proper, viz. feature vectors without occurrence numbers which are, in addition, *classified* as belonging either to the class `COSPEC` or `NON_COSPEC`.

   Cf. section 9 for a comprehensive list of file types that are specific to ROSANA-ML.

The next step consists in the computation of the respective simplified decision tree, which is performed externally to ROSANA-ML; this process is hence not supported by the graphical interface of ROSANA and will be described in detail in section 7.
In the application phase of ROSANA-ML - decision-tree-based anaphor resolution -, it is assumed that the respective (general or type-specific) decision-tree classifiers are available. The graphical user interface provides some abbreviated sample calls:[3]

4. *Application case - applying learned decision trees for anaphor resolution:* for continuing the above example, select `"AR 6sig Genclass PressReleases [ 32-66 ]` in the `Train & Resolve` menu. Execute the pasted function call in the ROSANA-ML listener.

   ```
   SEM 2 > (AR-6sig-genclass-PressReleases-32-66)
   ```

   This invokes the anaphor resolution process. The results are written to the respective directory of the selected signature. In the above case, the name of the directory is `$ROSANAPATH/ROSANAML51/Data/PressReleases/6sig/1-31/1modus/` .
   Alternatively, machine-learning-based anaphor resolution may be started manually by employing the interface function *dpe-anares*, having available the full set of options (cf. section 12).

5. *Browsing the generated output files:*

   (a) Switch to this directory.

   (b) Examine the result files of type `.ana`, `.ref`, and `.sco`, the format of which is identical to the output files generated by ROSANA (cf. [4]).

[3] In the sample calls, it is assumed that, in case of general classifiers, the decision tree classifier is named *pr.dts.gen*; in case of type-specific classifiers, the respective decision trees have to be named *pr.dts.pe3* (non-possessive third-person pronouns) and *pr.dts.po3* (possessive third-person pronouns). It is further assumed that these trees reside in the *1modus* subdirectory of the directory to which the feature vector files have been written (cf. below, section 8).

6 Using ROSANA-ML under other Lisp environments

ROSANA-ML should work under any Common Lisp Environment. Under environments other than Xanalys Lispworks, however, no graphical user interface will be available. Employing the non-graphical version of ROSANA-ML involves virtually the same steps as described, regarding ROSANA, in [4]:

1. *Starting the listener version of ROSANA-ML:* load the file *rosanaml.lisp*:

   ```
   CL-USER 2 > (load "rosanaml")
   ```

   During first-time use, a copyright and licensing message will be displayed. Please read carefully through the "**License Agreement for ROSANA-ML and ROSANA**" that comes along with this distribution (cf. appendix A). Upon acceptance of the licensing conditions, the binaries will be loaded, and ROSANA-ML will be initialized to some default settings.

2. *Change to Package* `SEM`*:*

   ```
   CL-USER 3 > (in-package "SEM")
   ```

3. *Processing sample data:* traning case generation as well as anaphor resolution proper may be performed by issuing one of the sample calls (as output by the help function *(help-ar)*). E.g., corresponding to the example discussed in section 5, the predefined call

   ```
   SEM 4 > (FVG-6sig-PressReleases-1-31)
   ```

   starts the feature vector generation process, and the predefined call

   ```
   SEM 5 > (AR-6sig-genclass-PressReleases-32-66)
   ```

   initiates decision-tree-based anaphor resolution.

4. *Browsing the generated feature vector output files:* as described in section 5.

5. *Experimenting with the machine-learning, output, trace, and processing settings:* these settings may be changed by typing in the function calls `(configure-ml-settings)`, `(configure-signature)`, `(configure-ar-output)`, `(configure-processing-traces)`, and `(configure-processing-settings)`. E.g.

   ```
   SEM 6 > (configure-ml-settings)
   ```

   allows the user to interactively select among the set of available machine-learning (training case generation) options.
   The settings that are specific to the machine-learning process are described in more detail in sections 10 and 11. Regarding the further settings, the reader is referred to the ROSANA documentation [4].

6. *Getting help:* the available sample calls and configuration options might be redisplayed by typing

```
SEM 7 > (help-ar)
```

7 Learning C4.5 decision trees

The computation of the decision tree classifiers is performed externally to ROSANA-ML. In the subsequent discussion, which completes the example of sections 5 and 6, it is assumed that the C4.5 implementation [1] has been properly installed and that, according to the setting of the environment variable `$PATH$`, its binaries are visible.

1. *Switch to the decision tree experiment directory:*

   ```
   > cd $ROSANAPATH/ROSANAML51/Data/PressReleases/6sig/1-31/1modus/
   ```

2. *Learning decision tree classifiers:* make shure that the links *pr.names* and *pr.data* point, respectively, to the signature specification file *6sig.names* (contained in directory *6sig*) and to the training case file *pr.fvc* (contained in directory *1-31*). It is essential that the signature definition *pr.names* is compatible with the signature of the training vectors contained in *pr.fvc*. In the shell, execute the call to C4.5:

   ```
   > c4.5 -f pr > pr.dt
   ```

   This will compute a decision tree, which is written, together with supplementary statistical information, to the file *pr.dt*.[4] In particular, *pr.dt* contains the so-called *simplified decision tree*, which results from pruning an initial decision tree that tends to overfit the training data. (For further details regarding the information-theoretical background of decision tree learning, the reader is referred to the machine-learning monography [2] of Mitchell and to the documentation that comes along with the employed C4.5 implementation [1].) The simplified decision tree ought then be manually extracted (this may require substituting several *subtrees* by hand to obtain a monolithic representation of the decision tree) and written to the file *pr.dts*.[5] These simplified decision trees are taken as the classifier functions which are applied during anaphor resolution proper of ROSANA-ML.
   One might experiment with different settings of the decision tree learning process, in particular regarding the amount of pruning that is performed during simplification of the decision tree. The default (so-called) *pruning confidence* (CF) setting of ROSANA-ML amounts to 25%. Different values may be supplied by employing the following refined format for calling C4.5:

   ```
   > c4.5 -f pr -c 50 > pr.dt
   ```

   It is suggested to store the results of experiments with different C4.5 settings in different subdirectories (*1modus*, *2modus*, etc, cf. the recommendations in section 8).

[4] Moreover, two binary representations *pr.tree* and *pr.unpruned* of the decision tree output will be generated (cf. section 8), which are referred to by some of the tools that come along with the employed C4.5 implementation, and which are not directly relevant to ROSANA-ML.

[5] This may be easily understood by comparing the *.dt* and *.dts* files that come along with the sample data.

3. *Employing the learned decision tree classifiers for anaphor resolution by referring to the respective* .dts *simplified decision tree files:* in the abbreviated sample calls (cf. section 5), the following naming conventions are adopted: when using a single (general) classifier, it has to be named *pr.dts.gen*; in case of type-specific classifier functions, the two simplified decision trees ought be named *pr.dts.pe3* (for non-possessives) and *pr.dts.po3* (for possessives).
   Alternatively, ROSANA-ML may be called manually (cf. section 12). In this case, the user is free to adopt her own naming strategy; using conventions similar to the above described, however, is strongly recommended.

8 About the directory structure

In principle, the directory hierarchy employed to store the signatures, the feature vectors, the learned decision trees, and the anaphor resolution output may be chosen arbitrarily according to the individual researcher's preferences. However, the preconfigured directory structure of the above sample application of ROSANA-ML naturally reflects the sequential ordering of the series of experiments usually carried out on a particular training / evaluation corpus (cf. [6]):

```
PressReleases/pr.pos    parses
                .mor    morphological analyses
                .ngp    supplementary (number,gender,person) information
                .key    key (coreference classes)
              0sig/...  subdirectories that correspond to specific signatures
              1sig/...
              ...
              6sig/...
              usersig/usersig.names  signature specification (referred to by C4.5)
                     /1-31/          subdirectories that correspond to specific
                                     training document subsets
                           pr.fve       set of feature vectors
                           pr.fvc       classified set of feature vectors (training cases)
                           1modus/...      subdirectories that correspond to specific
                                           C4.5 modes
                                  pr.data     link to pr.fvc
                                  pr.names    link to usersig.names
                                  pr.dt       learned C4.5 decision tree
                                  pr.dts      simplified decision tree (classifier proper)
                                  pr.tree     decision tree binary
                                  pr.unpruned unpruned decision tree binary
                                  pr.sco      anaphor resolution: evaluation results
                                  pr.ref                          output
                                  pr.ana                          coreference classes
```

At the highest level, experiments with different feature vector signatures are carried out. One level below, further experiments may be performed with different partitions of the document set into training data and evaluation data; in the above shown sample hierarchy, a subdirectory `1-31` of the `usersig` directory stores the data of a series of experiments with signature *usersig* in which *documents 1 to 31* of the *Press Releases* corpus have been used to generate the training cases, which are stored in files *pr.fve* and *pr.fvc*. Under the `1-31` directory, further experiments with different settings of the C4.5 learning algorithm proper (e.g., regarding the pruning confidence factor) may be distinguished; the results of these experiments (decision trees and - eventually - the results of decision-tree-based anaphor

resolution) are stored in respective subdirectories (e.g., in the above example hierarchy, in the directory `1modus`).

9 About the ML-specific file types

In correspondence with the practice employed regarding the corpus and evaluation data (cf. the ROSANA documentation [4]), the different kinds of *machine-learning-related* knowledge are distinguished through filename suffixes:

- *filename.fve:* the generated feature vectors, including the numbers of the respective anaphor and antecedent candidate occurrences over which the particular vectors have been computed (occurrence numbers are required to relate the vectors, during classification, to the respective key data);
- *filename.fvc:* the training cases proper, viz. the feature vectors without occurrence numbers which are, in addition, *classified* as belonging either to the class `COSPEC` or `NON_COSPEC`;
- *filename.data:* as *filename.fvc*, training cases (needed because the employed C4.5 implementation requires the specific file suffix *filename.data:*
- *filename.names:* the signature specification as required by the employed C4.5 implementation;
- *filename.dt:* the decision tree output as delivered by the employed C4.5 implementation;
- *filename.dts:* the *simplified* decision tree (as contained in *filename.dt*), which is employed as the classification function proper during anaphor resolution;
- *filename.tree:* a binary representation of the learned decision tree (referred to by some of the tools that come along with the employed C4.5 implementation);
- *filename.unpruned:* a binary representation of the learned decision tree *before pruning* (referred to by some of the tools that come along with the employed C4.5 implementation).

Regarding the futher input and output file types of anaphor resolution and evaluation / scoring, the reader is referred to [4].

10 Training data generation settings

The actual extension of *(anaphor, candidate)* $((\alpha, \gamma))$ pairs over which feature vectors are constructed during a particular training case generation run is determined by a set of options that is made available by the menu `ML Settings` or its non-graphical counterpart, the function call `(configure-ml-settings)`. In the lower part of the menu, the *type of third-person pronouns* may be selected that are considered as anaphors α for which feature

vectors are to be constructed. These (mutually exclusive) configuration options determine whether training data for learning one general (possessive, non-possessive) classifier or two type-specific classifiers (requiring, thus, two seperate runs) is generated, i.e.:

- `general (not type-specific) classification`: anaphors α are taken into account that are third-person possessive or non-possessive pronouns;
- `pers3 classifier`: only anaphors α are taken into account that are non-possessive pronouns;
- `poss3 classifier`: only anaphors α are taken into account that are possessive pronouns.

The experiments described in [4] gave evidence that, under certain conditions, it may be useful to employ type-specific classifiers.
In the upper part of the menu, certain properties of the antecendent candidates γ to be taken into account may be determined. The following (non-exclusive) options (toggles) are availabe:

- `no cataphors`: if selected, only candidates γ are considered that, at text surface, precede the respective anaphor α;
- `require full congruence (including gender)`: if selected, only candidates γ are considered that are fully congruent with the respective anaphor α (i.e. gender agreement is made mandatory);
- `no recency limits`: if selected, no surface recency limit is imposed on the candidates γ;
- `no restrictions`: if selected, no candidate filtering is applied, i.e. even pairs (α, γ) for which there is morphological or syntactic-configurational evidence of non-coreference are mapped to training data instances.

The `no cataphors` option was determined to be useful because ROSANA-ML proved to be unable to learn the (knowingly useful) preference of non-cataphoric resumption, which seems to be induced by the inherent symmetricality of the coreference information over which the training data is classified. The `require full congruence (including gender)` option proved to be useful because cases of partial agreement are extremely rare in practice, and the training data is by far too sparse to allow ROSANA-ML to learn the exceptional cases; instead, it seemed to spend much effort in reconstructing the full agreement rule. The latter two options (`no recency limits`, `no restrictions`) have been used to experiment with means for artificially enlarging the training data at the expense of a certain amount of adulterating it. It was found that, in certain cases, the former option might be of use; however, for the considered training / evaluation corpus, best results were obtained by not selecting these options. Cf. [4] for further details.

11 Selecting and defining signatures

Various predefined feature vector signatures are made available by the menu `ML Signature` or its non-graphical counterpart, the function call `(configure-signature)`. It is distinguished between *individual features*, which are determined by looking at a single occurrence (anaphor or candidate), and *relational features*, which are determined by relating the occurrences of anaphor and candidate.

- `sig 0`: seven features (*individual:* occurrence types, syntactic functions; *relational:* sentence and word distance, syntactic parallelism),
- `sig 1`: eighteen features (*individual:* occurrence types, syntactic functions, binarily encoded morphological number and gender, lexem (anaphor only); *relational:* sentence distance, direction of resumption, syntactic parallelism),
- `sig 2`: twelve features (*individual:* occurrence types, syntactic functions, morphological number and gender of anaphor and candidate, lexem of anaphor; *relational:* sentence distance, direction of resumption, syntactic parallelism),
- `sig 3`: eight features (*individual:* occurrence types, syntactic functions, of anaphor and candidate, lexem (anaphor only); *relational:* sentence distance, direction of resumption, syntactic parallelism),
- `sig 4`: twenty-three features (*individual:* occurrence types, syntactic functions, morphological number and gender, level of syntactic embedding of containing clause, syntactic categories and functions of immediate left and right neighbours of anaphor and candidate, lexem (anaphor only); *relational:* sentence distance, direction of resumption, syntactic parallelism, relative syntactic dominance),
- `sig 4a`: fifteen features (*individual:* occurrence types, syntactic functions, morphological number and gender, level of syntactic embedding of containing clause of anaphor and candidate, lexem (anaphor only); *relational:* sentence distance, direction of resumption, syntactic parallelism, relative syntactic dominance),
- `sig 5`: thirty-nine features (*individual:* occurrence types, syntactic functions, morphological number and gender, level of syntactic embedding of containing clause, syntactic categories and functions of three left and right neighbours of anaphor and candidate, lexem (anaphor only); *relational:* sentence distance, direction of resumption, syntactic parallelism, relative syntactic dominance),
- `sig 6`: thirty-six features (*individual:* occurrence types, syntactic functions, morphological number and gender, syntactic categories and functions of three left and right neighbours of anaphor and candidate, lexem (anaphor only); *relational:* sentence distance, direction of resumption, syntactic parallelism).

These signatures, which have been considered during the comprehensive evaluation experiments described in [6], should be understood as *examples* which, in particular, illustrate the information available in the internal occurrence and context representation, and how

it may be accessed and employed for generating feature vectors. The user may, and, in fact, should experiment with his own signatures, e.g. by rewriting the definition of function *baue-usersig-feature-vektor* in module *anaphernresolution.lisp*; the function definitions of the predefined signatures (*baue-6sig-feature-vektor* etc) might be taken as templates. By selecting option `sig user defined` in menu `ML Signature`, the signature defined by function *baue-usersig-feature-vektor* will be activated. (In its initial version, this function points to the signature definition *baue-4asig-feature-vektor*, which was determined to be the empirical winner in the expereiments described in [6].)

12 Calling ROSANA-ML manually

Instead of using one of the predefined shortcuts, feature vector generation as well as machine-learning-based anaphor resolution might instead be started manually, having available a more comprehensive set of options. A typical call of the interface function *dpe-anares-fv-generierung* looks as follows:

```
SEM 8 > (dpe-anares-fv-generierung
         "PressReleases/pr.pos"                ;; (1) the (parsed) document collection
         "PressReleases/usersig/1-31/pr.fve"   ;; (2) the output .fve file (optional)
         nil                                   ;; (3) dummy input .dts file (optional)
         nil                                   ;; (4) non-default .mor file (optional)
         nil                                   ;; (5) key data to be used for scoring (optional)
         (dnum-nach-dname '(1 31)))            ;; (6) specificaton of the document subset to be
                                               ;;     used for generating training cases (optional)
```

As a general rule, if a particular optional file name parameter is not provided (i.e. set to *nil* in the above call), ROSANA-ML reconstructs the name by looking at the first, mandatory parameter (*.pos* filename), taking its name (including path) without extension as the base, and adding the respective suffix. Hence, in the above sample call, ROSANA-ML searches for a file with name *PressReleases/pr.key* and uses its content for scoring the feature vectors; the scored vectors (i.e. the training cases proper) are written to a *.fvc* file in the same directory as the *.fve* file, viz. to *PressReleases/usersig/1-31/pr.fvc*. Regarding the technique used for specifying a subset of corpus documents, which are here used for computing the training cases (parameter *(6)*), the reader is referred to the ROSANA documentation [4]. If this parameter is not provided or, equivalently, set to nil, the whole bunch of documents will be processed; this might be useful when generating a comprehensive set of feature vectors for cross-validation purposes. Finally, for mainly technical reasons, there is a dummy parameter, which should specify an arbitrary simplified decision tree compatible with the employed signature (if left *nil*, ROSANA-ML follows the optional parameter policy); the reader should pay no attention to this, i.e. she should simply supply the dummy simplified decision tree (*defaultbaum.dts*) that comes along with the corpus data and a link *pr.dts* referring to it.
Machine-learning-based anaphor resolution proper may be manually called by employing the following template:

```
SEM 9 > (dpe-anares
         "PressReleases/pr.pos"      ;; (1) the (parsed) document collection
         nil                         ;; (2) application mode (nil  => anaphor resolution)
         "PressReleases/usersig/1-31/1modus/pr.dts.pe3" ;; (3) pe3 or general classifier
         "PressReleases/usersig/1-31/1modus/pr.dts.po3" ;; (4) po3 classifier or nil
         nil                         ;; (5) non-default .mor file (optional)
         "PressReleases/pr.key"      ;; (6) key data to be used for scoring or nil
                                     ;;     (nil => results are not scored)
         (dnum-nach-dname '(32 66))) ;; (7) specificaton of the document subset to be
                                     ;;     used for generating training cases (optional)
```

Parameter *(2)*, if set to nil, specifies the application mode to be anaphor resolution proper.[6] Parameters *(1)*, *(5)*, and *(7)* are identical to the respective parameters above, as are the filename derivation conventions regarding unspecified optional parameters. Contrarily to above, however, if parameter *(6)*, which specifies the key data file, is set to nil, no scoring of the anaphor resolution results is performed, i.e. the key file name is *not* automatically reconstructed. Parameters *(3)* and *(4)* specify the simplified decision trees to be used as classifier functions. If the latter parameter is non-nil, ROSANA-ML employs type-specific classifiers, interpreting parameter *(3)* as specifying the *.dts* file containing the decision tree to be used for non-possessives, and parameter *(4)* as specifying the *.dts* file containing the decision tree to be used for possessive pronouns. If only parameter *(3)* is supplied, the simplified decision tree in the respective file is employed as the general (non type-specific) classifier; a respective call may look as follows:

```
SEM 10 > (dpe-anares
          "PressReleases/pr.pos"      ;; (1) the (parsed) document collection
          nil                         ;; (2) application mode (nil  => anaphor resolution)
          "PressReleases/usersig/1-31/1modus/pr.dts.gen" ;; (3) pe3 or general classifier
          nil                                            ;; (4) po3 classifier or nil
          nil                         ;; (5) non-default .mor file (optional)
          "PressReleases/pr.key"      ;; (6) key data to be used for scoring or nil
                                      ;;     (nil => results are not scored)
          (dnum-nach-dname '(32 66))) ;; (7) specificaton of the document subset to be
                                      ;;     used for generating training cases (optional)
```

It is recommended to adopt the naming convention employed in the above examples: type-specific simplified decision trees should be stored in *.dts.pe3* and *.dts.po3* files, and general classifiers should be made available in *.dts.gen* files.

13 Tools for analyzing the training data

The quality of the learned decision tree classifiers directly depends on the quality of the employed training data, i.e. the classified feature vectors. If there are larger sets of feature-identical vectors of which, according to the key, a non-neglectible number of instances belongs to class COSPEC, and a non-neglectible rest belongs to class NON_COSPEC, then this imposes a problem because it will be impossible for the decision tree algorithm to determine a feature-based criterion to classify these cases correctly. Thus, qualitatively analyzing these cases may give valuable hints for a goal-directed refinement of the underlying signature, and may further help to understand the limits of applying decision-tree classifiers to anaphor resolution.

[6] as opposed to feature vector generation, which indirectly employs function *dpe-anares*

Along with the ROSANA-ML distribution comes the tool *sort-fv* that facilitates this qualitative analysis by providing a sorting function that groups feature-identical vectors and that allows to further focus on the important cases in which there are non-trivial numbers of heterogeniously classified vectors. The tool works on classified feature vector (*.fvc*) files. Let's assume that ROSANA-ML has been initialized as described in sections 5 and 6. Compile and load *sort-fv* by executing the following calls:

```
SEM 11 > (compile-file "Tools/vga/sort-fv")
SEM 12 > (load "Tools/vga/sort-fv")
```

It is highly recommended to work on a special version of *.fvc* file in which the feature vectors still contain the occurrence numbers of anaphors and antecedent candidates as the two rightmost attributes. This strongly facilitates the task of qualitatively analyzing the vector groups since, inside each subgroup, vectors will be sorted according to their occurrence numbers, and, in particular, it is possible to refer to the respective position in the original corpus data (occurrence number = token number). This special *.fvc* file version may be computed out of the respective *.fve* file by manually issuing a call of the function *klassifiziere-feature-vektoren* where the last parameter is set to true, e.g.:

```
SEM 13 > (klassifiziere-feature-vektoren
          "PressReleases/usersig/1-31/pr.fve"    :: input file
          "PressReleases/pr.key"                 ;; key data
          "../Data"                              ;; path prefix
          t)    ;; indicator: occurrence numbers should be included
```

The output consists of the file *pr.fvc*. To apply the tool on this file (i.e. compute the file of grouped classified vectors), the function *sort-fvc-datei* has to be called:

```
SEM 14 > (sort-fvc-datei
          "../Data/PressReleases/usersig/1-31/pr.fvc"   ;; input file
          "../Data/PressReleases/usersig/1-31/pr.fvcs") ;; output file
```

In this case, the output (*.fvcs*) file contains all vector groups, i.e. even the trivial ones that are unambiguously classified, or that consist of only one single vector. To focus on the interesting cases, two additional parameters may be supplied:

```
SEM 15 > (sort-fvc-datei
          "../Data/PressReleases/usersig/1-31/pr.fvc"   ;; input file
          "../Data/PressReleases/usersig/1-31/pr.fvcs"  ;; output file
          0.1  ;; 0 <= delta <= 0.5. The closer this value is to 0, the smaller
               ;; is the relative COSPEC and NON_COSPEC subset size difference
               ;; of vector groups written to the output
          7)   ;; minimum vector group size (``mindestgroesse'')
```

If the first additional parameter is set to a value close to zero, only vector groups with COSPEC and NON_COSPEC subsets of nearly equal size will be output (0.5 yields all groups). The second additional parameter constitues a lower bound on the minimum overall group size of vector groups written to the output. At the end of the *.fvcs* file, statistical information regarding the total number of vectors and groups as well as the vector group coverage according to the chosen settings is included.

14 Tools for generating training data partitions

To guarantee that the evaluation results of machine-learning-based anaphor resolution are not artefacts of the particular partition of the data into training cases and evaluation cases, cross-validation should be performed (cf. [6]). Along with the ROSANA-ML distribution comes the tool *random-split* that facilitates the randomized partition of feature vector files (for internal, classifier-oriiented cross-validation) and of sets of documents (for application-oriented cross validation, i.e. at the anaphor resolution level).
Let's assume that ROSANA-ML has been initialized as described in sections 5 and 6. Compile and load *random-split* by executing the following calls:

```
SEM 16 > (compile-file "Tools/cv/random-split")
SEM 17 > (load "Tools/cv/random-split")
```

At the level of internal, classifier-oriented cross-validation, it should be referred to the complete set of training cases, which has been computed over the whole corpus (e.g., in case of the *PressReleases* corpus, over all sixty-six documents). Let's assume that the respective classified feature vector (*.fvc*) file is available in the directory *usersig/1-66/*. Function *split-randomly-into-10* randomly partitions this file by generating ten mutually disjoint files *pr.fvc.1*, ..., *pr.fvc.10* of approximately equal size:

```
SEM 18 > (split-randomly-into-10
          "../Data/PressReleases/usersig/1-66/pr.fvc" ;; source .fvc file
          15548)                                      ;; wc -l pr.fvc
```

As the second parameter, the total number of vectors (= lines of the *.fvc* file) has to be provided, which may be determined by the Unix utility `wc`.
At the level of application-oriented (i.e. anaphor resolution) cross-validation, the atomic units of evaluation are whole documents. To generate random 6-fold partitions, simply apply function *split-numbers-randomly-into-6* as follows, where the sole parameter is the total number of documents:

```
SEM 19 > (split-numbers-randomly-into-6 66)
```

The output consists of six mutually disjoint document number lists, which are written to the standard output.

References

[1] *C4.5 implementation for Unix of the University of Regina, Release 8.* Available at `http://www.cs.uregina.ca/~dbd/cs831/notes/ml/dtrees/c4.5/tutorial.html` (1st February 2002).

[2] Tom M. Mitchell. *Machine Learning.* McGraw-Hill, New York, 1997.

[3] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, 1993.

[4] Roland Stuckardt. *Getting Started with ROSANA.* Documentation, distributed together with ROSANA and ROSANA-ML.

[5] Roland Stuckardt. *Design and Enhanced Evaluation of a Robust Anaphor Resolution Algorithm*, In: *Computational Linguistics* 27(4), 2001, 479-506.

[6] Roland Stuckardt. *Machine-Learning-Based vs. Manually Designed Approaches to Anaphor Resolution: the Best of Two Worlds*, In: *Proceedings of the 4th Discurse Anaphora and Anaphor Resolution Colloquium (DAARC 20002)*, University of Lisbon, Portugal, September 2002, 211-216. Distributed together with ROSANA-ML.

A License Agreement for ROSANA-ML and ROSANA

This agreement is made and entered into as of _____ by and between the parties:

a)
Dr. Roland Stuckardt
Im Mellsig 25
D-60433 Frankfurt am Main
Germany

referred to as: "**LICENSER**"

b)

[The user who gained access to ROSANA-ML and/or ROSANA immediately through LICENSER]

referred to as: "**USER**"

A.1 Product

ROSANA is the Common Lisp implementation of a robust syntactic salience-based anaphor resolution algorithm. ROSANA-ML is the Common Lisp implementation of a robust corpus-based anaphor resolution algorithm that employs antecedent preference strategies derived by decision tree learning.

ROSANA and ROSANA-ML come with two (copy-righted!) sample document collections, comprising ASCII versions of the texts, the respective parses (as determined by the robust parser of Timo Järvinen and Pasi Tapanainen), the respective key data (coreference annotations), and some supplementary information. An evaluation module is included which scores the output of ROSANA and ROSANA-ML according to various evaluation disciplines.

A.2 Copyright

Copyright of ROSANA-ML and ROSANA is to Dr. Roland Stuckardt.

A.3 License

LICENSER grants USER a non-exclusive license to use ROSANA-ML and/or ROSANA, versions as distributed. USER agrees to use ROSANA-ML and/or ROSANA only for non-commercial, non-profit research purposes; to report changes that USER makes to the

programs to LICENSER; and to acknowledge the use of ROSANA-ML and/or ROSANA in all publications reporting on results produced with the help of ROSANA-ML and/or ROSANA.
Use of ROSANA-ML and/or ROSANA or products derived from ROSANA-ML and/or ROSANA for any commercial purposes requires explicit written agreement of LICENSER.

A.4 Non-Disclosure

ROSANA-ML and/or ROSANA will be held in confidence by USER and will not be disclosed by USER to third parties.
USER shall and will employ all necessary precautions to ensure that no persons or institutions other than persons as are in the employ of USER or in the same research project as USER will get access to ROSANA-ML and/or ROSANA or parts thereof. Other persons or institutions desiring access to ROSANA-ML and/or ROSANA should be directed to LICENSER to obtain separate license agreements.

A.5 Fee

This license is granted by LICENSER to USER free of charge.

A.6 Disclaimer

ROSANA-ML and/or ROSANA and its documentation is provided on an "as is" basis, with no guarantee of its veracity or accuracy. No liability is accepted for any damage caused by its use.