JNCC2 user manual and tutorial (rev 1)

Giorgio Corani and Marco Zaffalon
IDSIA
Galleria 2, CH-6928 Manno (Lugano)
Switzerland
{giorgio,zaffalon}@idsia.ch

Technical Report No. IDSIA-09-07-rev1
May 2008

IDSIA / USI-SUPSI
Dalle Molle Institute for Artificial Intelligence
Galleria 2, 6928 Manno, Switzerland

JNCC2 user manual and tutorial (rev 1)

Giorgio Corani and Marco Zaffalon
IDSIA
Galleria 2, CH-6928 Manno (Lugano)
Switzerland
{giorgio,zaffalon}@idsia.ch

May 2008

This paper introduces JNCC2, the Java implementation of the Naive Credal Classifier 2 (NCC2). JNCC2 is open source; it is hence freely available together with manual, sources and javadoc documentation.

NCC2 is an extension of Naive Bayes Classifier (NBC) to imprecise probabilities, designed so as to return robust classifications even on small and/or incomplete data sets. A peculiar feature of NCC2 is that it returns indeterminate classifications (i.e., more than one class) when faced with doubtful instances. The empirical results of Corani and Zaffalon (2008) have shown that NCC2 returns indeterminate judgments on instances whose classification is truly doubtful; in fact, NBC achieves a much higher classification accuracy on the instances precisely classified by NCC2, than on those indeterminately classified by NCC2.

1 Introduction

Classifiers learn from data the relationship that holds between a set of attributes (also called *features*) characterizing a given object, and the class the object belongs to. For instance, e-mail filtering is a classification problem: the classifier analyzes the frequency of some keywords contained in the message, to eventually decide whether the message is an ordinary e-mail or spam. Automated reading of postal codes, handwritten characters recognition and speech recognition constitute further examples of classification problems.

This paper introduces JNCC2, i.e., the Java implementation of the Naive Credal Classifier 2 (Corani and Zaffalon, 2008); JNCC2 is released as open source software under the GNU GPL license. JNCC2 runs hence under any platform for which the Java Virtual Machine is available; this includes Unix, Windows and Mac operating systems.

The Naive Credal Classifier 2 (NCC2) is designed to overcome some well-known drawbacks of the classical Naive Bayes Classifier (NBC); in particular, by relying on weaker assumption than Naive Bayes, it is able to deliver credible classifications, even in spite of small and/or incomplete data sets. This is achieved by returning *set-valued classifications*, i.e., a set of classes instead of a single (potentially unreliable) class, when faced with doubtful instances. Set-valued (or *indeterminate*) classification yield weaker conclusions compared to determinate classifications; yet, in the case of doubtful instances, they deliver more reliable conclusions than the former one. Moreover, indeterminate classifications clearly highlight instances whose classification is doubtful.

The first version of the Naive Credal Classifier has been proposed by Zaffalon (2001); it has shown excellent accuracy in complex case studies, regarding for instance dementia diagnosis (Zaffalon, 2002) and agricultural problems (Zaffalon *et al.*, 2003). Later on, however, the classifier has been greatly reworked (Corani and Zaffalon, 2008) to include a novel methodology to treat missing data; the resulting classifier has been named NCC2.

The empirical results of Corani and Zaffalon (2008) show that the instances indeterminately classified by NCC2 are in fact very uncertain: this statement is supported by the analysis of the NBC accuracy, which turns out to be much higher on the instances determinately classified by NCC2, than on those indeterminately classified by NCC2.

The paper is organized as follows: Section 2 provides an overview of the NCC2 algorithms, showing how it extends Naive Bayes to deal robustly with small data sets and missing data; Section 3 is a tutorial which shows how to use JNCC2, carrying out practical examples; Section 4 reports some experimental results obtained on publicly available data sets and hence easily replicable by the user.

2 Naive Bayes and Naive Credal Classifier 2

Classification is the problem to allocate individual instances into classes, on the base of a set of features (or attributes); classifiers are learned on a set of previously labeled instances (training set), and then they can be used to classify novel instances (test set).

Classifiers aim at learning about a domain using data as only source of knowledge. In order to draw credible conclusions in these conditions, it is important to properly account for the ignorances that characterize the process of learning from data. There are at least two such ignorances: (a) prior ignorance about the domain, as we use data as only source of knowledge and (b) ignorance arising from missing values, as data are often incomplete; in this case, ignorance is about the process that originates the missing values: i.e., the missingness process.

In the following we review how these issues are addressed by the classical Naive Bayes Classifier and by the Naive Credal Classifier 2. The common point between NBC and NCC2 is the (naive) hypothesis of statistical independence of the features conditional on the class. The assumption is naive as instead quite often the features are related and hence mutually dependent; however, classifiers based on the naive hypothesis have been shown to be surprisingly effective in real-world applications, even if clear dependencies between the features are present (Domingos and Pazzani, 1997).

In the following, we review how the two classifiers deal with prior ignorance and missing data ignorance, and how they finally issue the classification.

2.1 Prior ignorance

NBC (and any Bayesian classifier as well) rests on the following paradigm: the classification is issued on the basis of a unique posterior distribution, computed multiplying via Bayes' rule a unique prior density (representing the investigator beliefs, *before* analyzing the data) and a unique likelihood. This way, especially on small data sets, the outcome can be sensitive to the specification of the prior distribution; if this happens, the classification reflects the beliefs of the investigator, rather than the objective knowledge acquired from the data. In particular, often a flat prior, assumed to be non-informative, is chosen; this is the case of many NBC implementations. Yet, this choice can bias the conclusions if the data generation mechanism is instead skewed, and the available data set is small. In fact, despite the literature effort devoted to design non-informative priors densities, the specification of any such prior appears to involve subjectivity.

However, using a single prior distribution is not the only possibility. In the recent years, new theories of so-called *imprecise probability* (Walley, 1991) have emerged that enable one to work with a set of densities, rather than with a unique density. From the imprecise probability viewpoint, the specification of a single prior distribution to model ignorance is too a strong assumption, which possibly biases the results; instead, models should use sets of distributions to that extent.

With reference to classification, for instance, NCC2 considers a set of priors, rather than a unique prior distribution; such set of priors is referred to as *prior credal set*. NCC2 computes *a set of posterior distributions* (derived from the set or priors applying Bayes' rule element-wise), and returns all the classes that are *non-dominated*[1] within the set. When several non-dominated classes are found, NCC2 issues a set-valued (or indeterminate) classification; for instance, it might output both 'disease A' and 'disease B'. A key point is that non-dominated classes are *incomparable*; this means that there is no information in the model that allows one to rank them. In other words, credal classifiers drop the dominated classes, as sub-optimal, and express indecision about the optimal class by yielding the remaining set of non-dominated classes. This is a major difference with respect to NBC, which returns instead the class with the highest probability in the unique posterior distribution (note that, however, the two models coincide if one defines a credal set containing a single prior for NCC2).

The frequency of indeterminate classifications decreases on large data sets, on which the specification of the prior distribution plays a minor role indeed. In fact, on large data sets, the posterior distributions computed by NCC2 tend to collapse towards a single distribution. On the contrary, in the case of extremely scarce data, NCC2 will tend to yield weakly informative conclusions (which means a large set of returned classes), that are nevertheless robust to the scarce available knowledge. This shift of paradigm allows NCC2 to deliver robust classifications in spite of small learning sets. In fact, NCC2 issues indeterminate classifications when faced with instances that are hard to classify, due to a combination of prior ignorance and poor information about those specific instances in the learning set, and over which NBC would output prior-dependent (and hence potentially unreliable) classifications.

2.2 Missing data ignorance

We can generally think of the data generation mechanism as composed by two processes: the first one which produces the complete, yet not observable, data; such data are referred to as *latent*. Then, a second process, called *missingness process* (MP), turns them into the incomplete but observable data we have access to; observable data are referred to also as *manifest*. A manifest value is hence identical to the corresponding latent one, unless the latent value has been turned into missing by the MP. The MP can process the latent data by generating random missingness or following a selective pattern, to eventually produce the manifest dataset we observe.

Although the MP can interfere with the process of learning the classifier from data, or of empirically measuring its performance, most classifiers (including NBC) simply ignore the MP: i.e., missing data are ignored during the learning, while during the testing, if a new instance to be classified contains a missing value, the probabilities of the different classes are computed by marginalizing the missing variable out. This is also the way the commonest implementations of NBC deal with missing data.

However, a sequence of works of statistical character (Little and Rubin, 1987; Heitjan, 1997; Grünwald and Halpern, 2003) has shown that ignoring missing data in this way is appropriate only if a particular condition, known as *missing-at-random* (MAR), is satisfied. In fact, recent research (Grünwald and Halpern, 2003) has pointed out that MAR is much less a frequent condition than it is usually supposed to be. Even on intuitive grounds, it is easy to imagine situations where data are turned into missing with different probability depending on the actual value of the variables. Generally it is not possible to use the data to test the assumptions about the process responsible

[1]Class c_i dominates class c_j if the estimated probability of c_i is larger than that of (c_j) for all the posteriors of the set.

for the missingness (Manski, 2003); hence, assuming MAR should be the result of an investigation involving also domain experts. It follows that, if one is ignorant about the MP, assuming MAR cannot be regarded as an objective-minded approach.

The NCC of Zaffalon (2001) included a methodology for robust inference of the classifier from incomplete data sets, without ignoring missing data; yet, these algorithms are appropriate just for a specific setting of the missingness process, and therefore they are not of general validity.

However NCC2 includes a much more flexible methodology to manage missing data, which allows one to declare some (possibly all or none) of the features as subject to a MAR process, and the remaining ones as subject to an MP unknown to us; the set of features subject to a MAR MP can be set differently from training and test set. This treatment of missing data rests on the *Conservative Inference Rule*, which enables one to compute (imprecise) conditional expectations with incomplete data (Zaffalon, 2005).

2.2.1 Conservative Inference Rule

CIR (Zaffalon, 2005) is a conditioning rule (i.e., a rule for computing conditional expected values) that generalizes the traditional conditioning; it assumes that prior beliefs are dealt with via a credal set $\mathcal{P}(\theta)$ and accounts for data sets, in which the missingness process is MAR for some variables, and unknown for some others. Moreover, CIR is able to manage variables whose MP is MAR in learning and unknown in testing, or vice versa. The two MPs (i.e., the MAR and the Non-MAR processes) are assumed to be independent of each other and their behavior is allowed for vary with different units, i.e., they are *not* assumed to be identically distributed.

To further describe CIR, let us introduce some notation.

Instances are indexed by i; the *learning set* (or *training set*) contains instances for which $1 \leq i \leq N$, while the unit to classify (not belonging to the learning set) is indexed by M. A set of units to classify is referred to as *test set*.

The class is denoted as C; for the i-th instance, it takes value c_i in the set $\mathcal{C}$. We assume class c_i to be always observed, as usual in supervised learning problems.

The l-th MAR feature is denoted as $\hat{A}_l$ ($0 \leq l \leq r$, with r total number of MAR features); for the i-th instance, it takes generic values $\hat{a}_l$ from the set $\hat{\mathcal{A}}_l$;

The j-th feature affected by an unknown MP ($0 \leq j \leq k$, with k total number of Non-MAR features) is denoted as A_j; for the i-th instance, it takes generic values a_j from the set $\mathcal{A}_j$.

Moreover, let us introduce the vectors of manifest variables shown in Figure 1: $\boldsymbol{d}$ contains classes and Non-MAR features of the instances of the learning set; $\boldsymbol{x}_M$ the Non-MAR features of the instance to classify; $\boldsymbol{d}^-$ is the union of $\boldsymbol{d}$ and $\boldsymbol{x}_M$; $\hat{\boldsymbol{x}}$ contains the MAR features of the instances of the learning set, and $\hat{\boldsymbol{x}}^+$ the MAR features of the instances of the learning set and of the unit to classify.

Manifest variables are denoted as $\mathbf{o}_{LV}$, where LV is the name of the corresponding latent variable. The manifest variables referring to the vectors of Figure 1 are hence denoted as $\mathbf{o}_{\boldsymbol{d}}$, $\mathbf{o}_{\boldsymbol{d}^-}$, $\mathbf{o}_{\boldsymbol{x}_M}$, $\mathbf{o}_{\hat{\boldsymbol{x}}}$, $\mathbf{o}_{\hat{\boldsymbol{x}}^+}$.

If a manifest vector (for instance $\boldsymbol{o}_M$) contains missing data, several realizations of the latent vector ($\boldsymbol{x}_M$ in the example) are possible; such realizations are obtained by considering all the possible replacements for missing data. The expression $\boldsymbol{x}_M \in \boldsymbol{o}_M$ denotes hence a realization $\boldsymbol{x}_M$ of the latent vector that is possible given the manifest value $\boldsymbol{o}_M$. Clearly, $\boldsymbol{x}_M = \boldsymbol{o}_M$ if $\boldsymbol{o}_M$ does not contain missing data.

Finally, the test of dominance based on CIR between classes c' and c'' is as follows:

$$1 < \min_{\boldsymbol{x}_M \in \boldsymbol{o}_M} \min_{\boldsymbol{d} \in \boldsymbol{o}} \inf_{p(\theta) \in \mathcal{P}(\theta)} \frac{p(c'_M | \boldsymbol{d}^-, \hat{\boldsymbol{x}}^+ \in \hat{\boldsymbol{o}}^+)}{p(c''_M | \boldsymbol{d}^-, \hat{\boldsymbol{x}}^+ \in \hat{\boldsymbol{o}}^+)}. \quad (1)$$

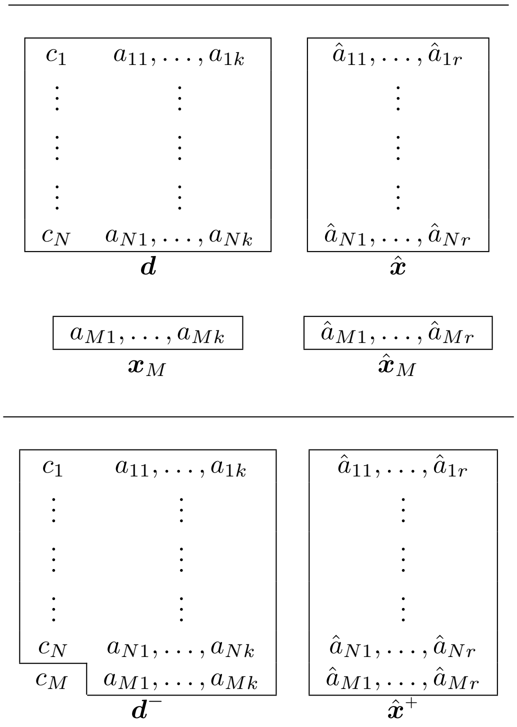

Figure 1: Graphical representation of some vectors of variables. Rows $1, \ldots, N$ constitute the training set, while the M-th unit is a new instance to be classified.

CIR can be regarded as unifying two rules: a *conservative learning rule*, and a *conservative updating rule*. The conservative updating rule prescribes to learn the classifier from an incomplete training set, by looping on the possible realizations of the Non-MAR part of the learning set; it is implemented by the middle optimization loop ($\min_{\boldsymbol{d} \in \boldsymbol{o}}$). On the other hand, the conservative updating rule prescribes to loop on the replacements for the Non-MAR missing values of the unit to classify; it is implemented by the outer minimum. The inner loop, which minimizes over the prior credal set, is common to both learning and updating rules.

The missing data, which are assumed to be MAR, are instead treated according to the standard approach followed by NBC, i.e., they are ignored; this is represented in the formulas by a notation of type $\hat{\boldsymbol{x}}^+ \in \hat{\boldsymbol{o}}^+$.

In fact, NCC2 specializes the test of Equation (1) to the case of naive classification, and such test is exploited to find out the non-dominated classes. Having defined the test of dominance, the procedure of Figure 2, based on pairwise classes comparisons, identifies the non-dominated classes.

2.3 Naive Bayes Classifier (NBC)

According to what we have seen so far, NBC is based on (a) the naive assumption, on (b) the specification of a single prior (usually a flat one) and (c) deals with missing data by assuming MAR.

As a result, the posterior probability of the generic class c' is computed as follows:

$$p(c'|\boldsymbol{d}, \hat{\boldsymbol{x}}, \boldsymbol{x}_M, \hat{\boldsymbol{x}}_M) \propto \frac{n(c')}{N} \prod_{l=1}^{r'} \frac{n(\hat{a}_{Ml}, c')}{n_l(c')}, \tag{2}$$

where:

- attributes have been re-ordered so as to index the non-missing ones in the instance to classify from 1 to $r' \leq r$. In fact, features missing in the instance to classify are marginalized out (only features indexed by $1 \leq l \leq r'$ affect the computed posterior probability);
- $n(c')$ denotes the number of instances with class c' in the learning set;
- $n(\hat{a}_l, c')$ denotes the number of joint occurrences of $(\hat{a}_l, c')$ in the learning set after dropping the units with missing values of $\hat{A}_l$;
- $n_l(c') = \sum_{\hat{a}_l \in \hat{A}_l} n(\hat{a}_l, c)$, i.e. $n_l(c')$ is the number of instances for which the value of $\hat{A}_l$ is present. Note that both counts $n(\hat{a}_l, c)$ and $n_l(c')$ *ignore* instances for which feature $\hat{A}_l$ is missing.

Note that the MAR assumption is necessary in order to justify both the marginalization of the features that are missing in the instance to classify, and the way counts $n(\hat{a}_l, c)$ and $n_l(c')$ are computed.

If either a count $n(\hat{a}_l, c')$ or $n(c')$ is 0, the probability estimated by Formula (2) for class c' would be 0. To avoid that, all counts $n(\hat{a}_l, c')$ and $n(c')$ are firstly initialized to 1, to which the frequencies empirically computed from the learning set are then added. Such an approach actually corresponds to use a flat prior density (known as Laplace prior), which is the commonest choice for NBC.

Accuracy of NBC is measured by the indicators typical of precise classifiers, such as:

- **accuracy**, i.e., the percentage of correct classifications;
- **confusion matrix**, i.e., a matrix whose generic cell (i,j) reports the number of instances of class i, which have been classified in class j; hence, it displays how misclassifications are distributed between the different true and predicted classes.

2.4 Naive Credal Classifier 2

As already outlined, NCC2 rests on (a) the naive assumption, (b) the specification of a set of priors to deal with prior ignorance and (c) on CIR for the management of missing data.

NCC2 returns the classes that are non-dominated within the set of computed posterior densities. The procedure to identify the non-dominated classes, based on pairwise comparison of the classes, is shown in Figure 2. The core of the procedure is the test of dominance, which assesses whether class c' dominates class c''. Actually, NCC2 implements the test of dominance prescribed by CIR (Equation 1), specializing it to the case of naive classification.

Evaluating NCC2 requires specific indicators, as it can return indeterminate classifications. In particular:

- **determinacy**, i.e., the percentage of classifications having as output a unique class;
- **single-accuracy**, i.e., accuracy of NCC2 when it is precise;
- **indeterminate output size**, i.e., the average number of classes returned when NCC2 is indeterminate;

CLASSIFICATION OF AN INSTANCE

1. set NonDominatedClasses := $\mathcal{C}$;
2. for class $c' \in \mathcal{C}$
   - for class $c'' \in \mathcal{C}$, $c'' \neq c'$
     - if c'' is dominated by c' (to be assessed via the below procedure), drop c'' from NonDominatedClasses;
     - exit;
   - exit
3. return NonDominatedClasses.

Figure 2: Summary of NCC2 procedures.

- **set-accuracy**, i.e., the percentage of indeterminate classifications that contain the actual class (note that if a data set has two classes, the output size is necessarily 2 and set-accuracy 100%);
- **confusion matrix**, which is computed in case of NCC2 *with reference to determinate classifications only*;
- **NBC(NCC2 D)**, as the accuracy achieved by NBC on the instances classified determinately by NCC2;
- **NBC(NCC2 I)**, as the accuracy achieved by NBC on the instances classified indeterminately by NCC2.

In fact, NBC(NCC2 D) coincides in almost any case with the single-accuracy of NCC2; in fact, when determinate, NCC2 returns the same output as NBC. Differences might arise only in case that the NBC prior is not included in the credal set, and that such a prior leads to a classification which is different from the classification to which leads any single prior of the credal set. Such an event is however very rare.

The experiments of (Corani and Zaffalon, 2008) have shown that NCC2 has high accuracy when it issues precise classifications, and that, on the other hand, it successfully recognizes instances that are hard to classify (because of prior ignorance or missing values), outputting in this case set-valued classifications; in fact, the NBC accuracy undergoes a major drop on the instances indeterminately classified by NCC2. Such a drop points out that the usual way to measure the performance of a classifier, i.e., its predictive accuracy, which is an average over all the instances of the test set, may not help uncover a possible bad performance of the classifier on a subset of the test instances. These instances are precisely those that are hard to classify and that NCC2 instead isolates by delivering set-valued classifications.

The experiments also show that if a non-identically-distributed MP is modeled as a MAR MP, the resulting empirical evaluations might be severely biased: even if the predictive accuracy on a certain instance is measured properly by cross-validation, the actual accuracy on new instances of the same type can be significantly worse. This appears to highlight the fact that making tenable assumptions is important even if data are available for empirical evaluations.

2.5 Feature discretization

NCC2 is designed to work with categorical variables. Hence, as pre-processing step, JNCC2 discretizes all the numerical features, using the supervised discretization algorithm of Fayyad and Irani (1993). This techniques is known to be effective: the empirical study of Dougherty *et al.* (1995) found a slight yet consistent improvement of the classification accuracy, for a number of different data sets and classifiers, working on data discretized via such algorithm rather than on the raw numerical data. Moreover, it has been demonstrated theoretically (Hsu *et al.*, 2003) that working with discretized features instead of modelling continuous values as generated by a normal distribution improves the performance of naive Bayes.

For each experiment, discretization intervals are estimated on the training set, and then applied unchanged on the test set. A feature turns out to be not sensitive for the classification problem if it is discretized into a unique bin; in this case, it is dropped from the experiment.

2.6 Computational complexity

A further issue in classification regards the computational complexity, in terms of both *time to learn* (especially in rapidly changing environments, it may be necessary to learn or update the classifier in real time), and *time to classify,* i.e., time required to issue a classification once the classifier has been trained.

The learning complexity is linear in the number of instances for both NBC and NCC2. Updating the parameters of the classifier, after having added novel instances to the training set, is accomplished for both NBC and NCC2 in time linear with respect to the number of the novel instances.

On the other hand, the classification complexity of NBC is linear in the number of attribute variables, while the classification complexity of NCC2 is roughly quadratic in the number of attribute variables.

Warning

For data sets characterized by several numerical features, most time is spent in discretizing features, rather than in learning or testing NBC or NCC2. In fact, on data sets with a significant number of numerical features (for instance, more than 10), some 50–90% of the overall computation time is spent discretizing features. As cross-check, we have found a similar behavior also in WEKA (Witten and Frank, 2005a), running NBC on numerical data sets. In these cases hence, computation times are largely determined by feature discretization.

3 A guided tour of JNCC2

3.1 Getting and Installing JNCC2

To run JNCC2, it is necessary to have installed JRE (Java Runtime Environment), release 6 (or above); JRE is freely downloadable from `http://java.sun.com/javase/downloads/index.jsp`.

The JNCC2 website is `http://www.idsia.ch/~giorgio/jncc2.html`, from which the binary file `jncc.jar` (which can be seen as the JNCC2 executable) and the relevant documentation (user manual and scientific papers) can be downloaded.

In order to run jncc2, it is necessary to add the location of file `jncc.jar` to the environment variable CLASSPATH, which specifies the location of user-defined Java packages to the Java Virtual Machine. This is accomplished as follows:

Unix:
```
export CLASSPATH=${CLASSPATH}:/path/to/file/jncc.jar
```

Windows:
```
set CLASSPATH=%CLASSPATH%;C:\path\to\file\jncc.jar
```

Adding permanently the JNCC2 location to the environment variable CLASSPATH can be done by using procedures specific for the operating system in use. For more information about how to set the CLASSPATH, see http://java.sun.com/javase/6/docs/technotes/tools/solaris/classpath.html for Unix and http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/classpath.html for Windows.

JNCC2 runs from the command-line.

3.2 Data format

JNCC2 loads data from ARFF files; this is a plain text format, developed for WEKA (Witten and Frank, 2005a), an open-source software for data mining. WEKA has become a popular tool for data mining, and in fact there is a large number of public data sets archived in ARFF format (see for instance the repository at http://www.cs.waikato.ac.nz/ml/weka/index_datasets.html).

The header of ARFF files carries out the variable declarations; after the header, data are written as comma separated values. It is possible to insert comments within the file, so that data sets can be accompanied by some relevant information. Appendix 6 reviews the details of the ARFF format, providing some remarks relevant for the use of ARFF files with JNCC2.

Warning

JNCC2 does not properly manage name of features or categories or classes containing white spaces, even if enclosed into quotes. Double check that the features and the classes declared in your ARFF file do not contain white spaces.

3.3 A worked example

In the following the functionalities of JNCC2 are shown, using as example the data set `labor.arff`, which regards the final settlements of labor negotiations in Canadian industry. The data sets contains 16 attributes (named, for instance, 'wage increase in first year of contract', 'number of working hours during week', etc.); the class to be predicted is 'good' or 'bad', i.e., the judgment issued by an expert about the contract. There are 57 instances; the percentage of missing data per feature ranges from 0% to 84%.

In the following, we didactically show how to use the software, rather than commenting on the classification performance.

JNCC2 can perform three kinds of experiments:

- validation of both NBC and NCC2 via 10 runs of 10-folds cross-validation;
- validation of both NBC and NCC2 via a single training/testing experiment, the classes of the testing set being known;
- training of both NBC and NCC2 on the training set and classification of instances from a testing set, whose actual classes are unknown.

The directory in which the ARFF files referring to the same case study (labor in our case) reside, is referred to as *working directory*. For instance, we create the working directory `/home/giorgio/labor` containing file `labor.arff`.

The specification of the features affected (either in training, testing, or both training and testing) by a NonMAR-MP is done, for all the different kinds of experiments, by creating the file `NonMar.txt` in the working directory.

Each row of this file follows this syntax:

- `training <name of the feature>`, to indicate that the feature is affected by a Non-MAR MP in training only;
- `testing <name of the feature>`, to indicate that the feature is affected by a Non-MAR MP in testing only;
- `<name of the feature>`, to indicate that the feature is affected by a Non-MAR MP in both training and testing;
- `nonmar`: this is a one-word shortcut that sets all features as NonMAR in both training and testing.

Nothing has to be written for features that are affected by a MAR MP both in training and testing. If file `NonMar.txt` is not present[2] in the working directory, JNCC2 assumes all features to be subject to a MAR MP both in training and testing; this is notified to the user via a console message.

3.3.1 Validation via cross-validation

Dataset	Instances	Features	Classes	NbcAcc	(StdDev)
labor-neg-data'	57	16	2	88.30%	13.04%
NccDeterm	(StdDev)	NccSingleAcc	(StdDev)	NccSetAcc	(StdDev)
86.97%	15.11%	93.73%	10.60%	100.00%	0.00%
NccIndetOutSize	(StdDev)	Nbc(Ncc-P)	(StdDev)	Nbc(Ncc-I)	(StdDev)
2	0	93.73%	10.60%	54.25%	45.32%

ResultsTable-CV.csv

Figure 3: Results table produced by a cross-validation experiment; for all indicators the standard deviation, computed over the 100 training/testing experiments, is reported. Results are here displayed over three rows for reasons of space, while the actual output of the program is arranged into a single line. In fact, the csv file produced by JNCC2 can be readily opened into a spreadsheet.

In 10-folds cross-validation, the instances of the data set are divided into 10 folds; folds are stratified, i.e., classes are represented with about the same proportion in each fold. Then, 10 training/testing experiments are performed, by using as training set the union of 9 folds, and the remaining fold as test set; hence, at the end, every fold is used once as test set. To get a more

[2]On case-sensitive operating systems (for instance, Unix), JNCC2 looks case-sensitively for file `NonMar.txt`. If a file named NonMar.txt is found, but written with different case, JNCC2 exits, asking the user either to fix the case of the file name, or to rename it differently.

Dataset: 'labor-neg-data' NBC Confusion Matrix		
'bad'	'good'	←classified as
17	2	'bad'
3	33	'good'

Dataset: 'labor-neg-data' NBC Confusion Matrix (determinate instances only)		
'bad'	'good'	←classified as
17	2	'bad'
3	33	'good'

ConfMatrices.txt

Figure 4: Confusion matrices produced by a cross validation experiments; the coefficients are averaged over the performed runs.

reliable measure of the classification performance, following the recommendation of Witten and Frank (2005a), JNCC2 performs 10 runs of cross-validation.

JNCC2 validates NBC and NCC2 via 10 runs of stratified cross-validation, i.e., performing 100 training/testing experiments. The command-line syntax is as follows:

```
java jncc20.Jncc <Working directory> <Arff file> cv
```

Let us assume all variables to be affected by a MAR MP in both training and testing; hence, we do not create the file `NonMar.txt`. We start the cross-validation experiment as follows:

```
java jncc20.Jncc /home/giorgio/labor labor.arff cv
```

The working directory can be indicated either in an absolute or relative way; a convenient shortcut, if the command is typed from the working directory, is '.':

```
java jncc20.Jncc .  labor.arff cv
```

The experiment takes less than one second on an ordinary PC.
Two files are created in the working directory: `ResultsTable-CV.csv` and `ConfMatrices.txt`. Their content is shown in Figure 3 and 4.

Repeating the same experiment different times can lead to small numerical differences in the indicators, because of the randomness inherent in cross-validation. Now, let us suppose the following variables to be affected by a Non-MAR MP:

- 'wage-increase-first-year' (in training);
- 'duration' (in both training and testing);

- 'statutory-holidays' (in testing).

In this case it is necessary, before running JNCC2, to create the file `NonMar.txt` in the working directory, as shown in Figure 5.

```
training 'wage-increase-first-year'
'duration'
testing 'statutory-holidays'
```

Figure 5: Example of declarations in `NonMar.txt`. Feature names are quoted in these declarations, because they are quoted in file `labor.arff`. In fact, the features listed in `NonMar.txt` have to be string-matchable (case-insensitively) with those declared in the ARFF file; if this does not happen, JNCC2 exits pointing out the mismatch.

The cross-validation experiment is then started using the same instruction as before; however this time, because of the declaration of NonMAR features, NCC2 will be more indeterminate. In fact, NCC2 precision drops from 89% to 47%; on the other hand, NBC performance does not show any significant change, as NBC always assume features to be MAR in both training and testing.

3.3.2 Validation via testing file

	A	B	C	D	E	F	G	H	I	J	K
1	INSTANCES AND PREDICTIONS										
2	spectacle	astigm.	tear-prod	ActualClass	NbcPosteriorDistribution			NbcPrediction	Ncc2Prediction		
3					soft	hard	none				
4											
5	myope	no	reduced	none	0.13	0.04	0.83	none	none	-	-
6	myope	no	normal	soft	0.62	0.17	0.2	soft	soft	-	-
7	myope	yes	reduced	none	0.02	0.19	0.8	none	none	-	-
8	myope	yes	normal	hard	0.09	0.72	0.19	hard	hard	-	-
9	hypermetrope	no	reduced	none	0.15	0.02	0.83	none	none	-	-
10	hypermetrope	no	normal	soft	0.72	0.08	0.2	soft	soft	-	-
11	hypermetrope	yes	reduced	none	0.02	0.09	0.88	none	none	-	-
12	hypermetrope	yes	normal	hard	0.17	0.52	0.31	hard	hard	-	-
13	myope	no	reduced	none	0.07	0.02	0.91	none	none	-	-
14	myope	no	normal	none	0.51	0.14	0.35	soft	soft	none	-
15	myope	yes	reduced	none	0.01	0.1	0.89	none	none	-	-
16	myope	yes	normal	hard	0.07	0.6	0.33	hard	hard	-	-
17	hypermetrope	no	reduced	none	0.08	0.01	0.91	none	none	-	-
18	hypermetrope	no	normal	soft	0.59	0.06	0.34	soft	soft	-	-
19	hypermetrope	yes	reduced	none	0.01	0.05	0.94	none	none	-	-
20	hypermetrope	yes	normal	none	0.12	0.39	0.49	none	hard	none	-

Figure 6: The predictions file produced by JNCC2, opened into a spreadsheet. The first columns contain the features and the actual class; the following columns report the distribution probability estimated by NBC, the class predicted by NBC and the prediction issued by NCC2 (possibly, more classes). The figure actually refers to the contact-lenses data set.

If validation is accomplished via a testing file, the working directory should contain two ARFF files; the first to be used as training set, and the second one to be used as test set. Variable declarations should be consistent (both in the names and in the order) between training and testing ARFF files; otherwise, JNCC2 exits, notifying the inconsistency via a console message.

As we do not have a second file of labor instances, we generate files `labor-training.arff` and `labor-testing.arff`, by putting half the instances of the original `labor.arff` in each of them (this is done outside JNCC2).

In case some variables are affected by a Non-MAR MP, file `NonMar.txt` has to be created as explained in Section 3.3.1.

The experiment is then started with the following syntax:

```
java jncc20.Jncc <Working directory> <Arff training file> <Arff testing file>
```

In our case, supposing to start JNCC2 from the working directory, the command is:

```
java jncc20.Jncc .  labor-training.arff labor-testing.arff
```

Besides the two already introduced files (`ResultsTable.csv` and `ConfMatrices.txt`), a third file is produced, i.e., `Predictions-Testing-labor.csv`, which reports the issued predictions. In particular, for each instance the following values are reported: the instance itself (feature values and actual class), the probability distribution computed by NBC over the different classes, the class predicted by NBC and the prediction returned by NCC2. The values referring to the same instance are arranged in the same row. In this way, the user can easily grasp, for instance, how NBC behaves on the instances indeterminately classified by NCC2. The predictions file can be readily opened by a spreadsheet, as shown in Fig. 6. Note that the standard deviations in file `ResultsTable.csv` will be missing, since a single training/testing experiment has been performed.

3.3.3 Multiple data sets

By using some easy shell commands, it is possible to automate the execution of JNCC2 over multiple data sets. As a first example, let us assume that all the features of all the considered data sets are MAR; in this case, there is no need of declaring NonMAR features. To automate the execution of cross-validation, for instance, one puts all the data sets into the same directory, removes all the non-relevant arff files from the directory, moves into the directory and then types:

```
for file in $(ls *.arff); do java jncc20.Jncc .  $file cv; done
```

The results generated by JNCC2 on the different data sets are appended on the same files; hence, `ResultsTable.csv` will contain several rows, each referring to a different data set, while `ConfMatrices.txt` will report the sequence of confusion matrices for NBC and NCC2 computed on the different data set.

If instead, each data set requires a specific declaration of NonMAR features, the procedure to automate the execution of JNCC2 is slightly different. One creates a clean directory, and then a sub-directory for each data set; in each sub-directory, one puts the data set and, if necessary, `NonMar.txt` properly filled. Then, cross-validation over multiple data sets can be automated as follows:

```
for directory in $(ls .); do cd $directory; arffFile=$(ls *.arff);
        java jncc20.Jncc .  $arffFile cv; cd ..; done
```

In order to use the above script, it is necessary that each sub-directory contains only a single file of type ARFF and that the root directory, in which the sub-directories are created, does not contain anything else apart from the sub-directories.

The above scripts can be easily customized to meet the actual needs of the user.

3.3.4 Validation via testing file, unknown classes

In this case, NCC2 is trained using the data set loaded from an ARFF file; then NCC2 classifies some instances whose class is unknown, and which are stored in a second ARFF file. Also in this case, variable names and order should be consistent between the two ARFF files; however, the class variable is missing in the second ARFF file.

File `NonMar.txt`, if necessary, has to prepared as usual.

The syntax to start the experiment is:

```
java jncc20.Jncc <Working directory> <Arff training file> <Arff testing file>
                              unknownclasses
```

The arguments `unknownclasses` (case insensitive) makes JNCC2 aware that no class information is available in the testing file.

In our example, supposing the file containing the instances without classes to be `labor-no-classes.arff,` the experiment is started from working directory as follows:

```
java jncc20.Jncc .  labor-training.arff labor-no-classes.arff unknownclasses
```

Then JNCC2 reports to file the classifications issued by NCC2, similarly to what shown in Figure 6; in this case, however, the column reporting the (unknown) actual class will be missing.

4 Experiments

data set	features	classes	instances	time (sec).
letter	16	26	20000	260
nursery	8	5	12960	11
segment-challenge	19	7	1500	57
vote	16	2	435	1
waveform	40	3	5000	480

Table 1: Characteristics of the data sets. Data sets letter, segment-challenge and waveform have only numerical features;
data sets nursery and vote have only categorical features. Computational times refer to 10 runs of 10-folds cross-validation, performed on Pentium 4 3.00GHz machine, running Linux 2.6.

In this section, we present some experimental results, considering several publicly available ARFF data sets. To run JNCC2 on different data sets, we create a different working directory for each data set (for instance, `/home/giorgio/letter, /home/giorgio/nursery,` etc.).

All data sets are complete, i.e., they do not contain missing data.[3] The characteristics of the data sets are presented in Table 1. On each data set, we evaluate the performance of both NBC

[3]Vote contains some 3-5% of missing values for each feature. All the features of this data set are binary. However, according to the accompanying documentation of the data set, data marked as missing are not unknown; indeed, they cannot simplified as 'yes' or 'not'. Hence, we treated the symbol of missing value as a further value for all the features, rather than as actual missing values.

Dataset	NCC2			
	Determ.(%)	SingleAcc(%)	SetAcc(%)	Ind.Out.Size
letter	95.2(0.5)	76.7(0.9)	57.3(5.1)	2.5/26
nursery	99.7(0.2)	90.4(0.8)	83.5(18.8)	2.0/5
segm-challenge	91.6(2.2)	94.2(2.0)	95.8(5.1)	3.9/7
vote	99.1(1.4)	90.5(4.1)	100.0(0.0)	2.0/2
waveform	99.3(0.4)	80.1(1.4)	100.0(0.0)	2.0/3

Table 2: NCC2 results measured via 10 runs of 10 folds cross-validation; standard deviations are reported in brackets.

Dataset	NBC Accuracy (%)		
	Entire data set	Subset of instances	
		NBC(NCC2 D)	NBC(NCC2 I)
letter	74.1(0.8)	76.7(0.8)	20.5(4.0)
nursery	90.3(0.8)	90.4(0.8)	54.1(30.3)
segm-challenge	90.0(2.4)	94.2(2.0)	43.3(14.2)
vote	90.1(4.2)	90.5(4.1)	38.9(46.1)
waveform	79.9(1.4)	80.1(1.4)	52.6(29.6)

Table 3: NBC results, measured via 10 runs of 10 folds cross-validation. Standard deviations are reported in brackets.

and NCC2 via 10 runs of 10-folds cross-validation, i.e., 100 training/testing experiments. We recall that, for each training/testing experiment, JNCC2 discretizes numerical variables before inducing the classifiers.

Selected indicators of NCC2 and NBC performance are reported respectively in Tables 2 and 3. Since the data sets are complete, indeterminate classifications are due to prior uncertainty only; however, as the data sets are quite large, prior uncertainty affects a small number of instances; in fact, NCC2 determinacy is higher than 90% on every data set. As a side-effect, the indicators referring to the instances indeterminately classified by NCC2 have larger standard deviations than those referring to the determinate classifications.

Table 3 reports the NBC accuracy measured on the whole test set, and then measured separately on the subsets of instances classified determinately or indeterminately by NCC2. There is however a clear drop of NBC accuracy (from about 83% to 43% on average) between the NBC(NCC2 D) and NBC(NCC2 I); this shows that NCC2 becomes indeterminate on instances that are truly hard to classify. When indeterminate, NCC2 delivers on average a set-accuracy of 83%, by returning about the 40% of the classes (the data set vote is excluded from this average, as it has set-accuracy 100% by definition); hence it remains reliable, even on doubtful instances, thanks to indeterminate classifications.

4.1 Feature selection

Redundant or related features, that hence violate the naive hypothesis, might bias the learning process of both NBC and NCC2. Hence, feature selection can sometimes improve the performance of both NBC and NCC2. Although JNCC2 automatically removes numerical features discretized into a single bin, it does not actually implement methods for feature selection; however, this can be accomplished for instance using WEKA (Witten and Frank, 2005b). Table 4 reports the difference of performance (for the sake of brevity, only three indicators are considered) before and after feature selection. In general, feature selection largely reduces the number of features, leading sometimes to significant improvements. In no case a worsening of the performance of NBC or NCC2 has been observed. No feature has been pruned from the nursery data set.

Data set	removed features	NBC ΔAcc	NCC2 ΔPrec	NCC2 ΔSingleAcc
letter	6/16	0.0	0.8	0.3
nursery	0/8	-	-	-
segm-challenge	13/19	3.2	4.0	1.2
vote	13/16	5.9	0.5	5.6
waveform	27/40	1.1	0.1	0.9

Table 4: Effects of feature selection. The variations are expressed in percentage points; for instance, NCC2 precision on letter is 96.0 on the pruned data set and 95.2 on the complete data set, hence Δ=0.8.

4.1.1 An example with missing data

	MAR MP on vote		
	NBC	NCC2	
	Accuracy (%)	Precision (%)	SingleAcc (%)
Cross-validation	95.0(3.3)	99.7(0.7)	95.2(2.9)
Testing file	96.3	99.1	96.7

Table 5: NCC2 results measured on vote, after having generated 10% random missingness. As the data set has two classes, we do not report set-accuracy and output size, which are respectively 100% and 2. For cross-validation, the standard deviation is reported in brackets.

We focus now on the vote data set, to show some results with missing data. We work on the data set after having performed feature selection (Section 4.1); hence, the data set has 3 features, 2 classes and 435 instances.

As first experiments, we generate 10% random (i.e., MAR) missingness on each feature, thus eventually building a second ARFF file (these operations are accomplished outside JNCC2). Then, we run a cross-validation experiment.

Afterward, we create the files `vote-training.arff` and `vote-testing.arff`, by dividing into two stratified halves (i.e., classes are represented with about the same proportion in the two subsets) the instances of the original data set. Then, we run validation via testing file. In both cases (cross-validation and testing file) we do not create the file `NonMar.txt`.

Results, reported in Table 5, show that the two validation methods lead to consistent conclusions, apart from minor differences. The performance of both NBC and NCC2 shows only a small worsening on the data set with missing data, compared to the complete data set. In general MAR missing data (if limited to a reasonable amount) do not heavily spoil the classifiers performance, nor they bias the empirical evaluation of the classifiers.

On the other hand, however, treating as MAR the data generated by a Non-MAR MP can lead to severe misclassifications, and also to erroneous empirical assessment of the classifiers accuracy. For instance, with reference to the vote data set, let us name as 'type A' the instances with values (*n, y, n, class1*) and as 'type B' the instances with values (*y, n, n, class0*). We turn type A instances of the training file into (**, *, n, class1*), and type B instances of the testing file into (**, *, n, class0*). Hence, data are turned into missing by a Non-MAR MP which takes into consideration the joint values of the features, and that is not identically distributed between training and testing.

First, we run a cross-validation experiment, using the instances of `vote-training.arff` only, over which hence the MP is identically distributed. We repeat this experiment twice: once without creating file `NonMar.txt,` and once declaring as Non-MAR all features in both training and testing. Results are shown in the first row of Table 6.

Then, we run validation via testing file `(vote-testing.arff)`; also in this case, we repeat the experiment with and without creating file `NonMar.txt` (second row of Table 6).

	NonMAR MP on vote				
	NBC	NCC2		NCC2 (NonMar.txt)	
	Acc.(%)	Prec(%)	SingleAcc(%)	Prec.(%)	SingleAcc(%)
Cross-val.	88.2(3.1)	89.9(0.8)	93.9(2.0)	49.5(0.9)	100(2.5)
Testing file	71.4	100	71.4	55.3	99.2

Table 6: Results on the vote data set, having generated NON-MAR missingness.

The results of Table 6 show that assuming MAR when the MP is Non-MAR can lead to severe misclassifications. Of course, this is an 'extreme' example, which heavily relies on the fact that the MP is not identically distributed. Yet, note that if one is ignorant about the MP, such a behavior should be consider as a possibility, which is just what one can do with JNCC2, by declaring the MP as Non-MAR.

In real case studies, it is however recommended that the investigator declares as MAR or NonMAR each feature after having discussed with domain experts the reasons which might turn the data into missing.

5 Conclusions

The paper has introduced JNCC2, the Java implementation of the NCC2. It is released under the term of the GPL license, and it is freely downloadable (together with manual, sources and javadoc documentation) from the website `http://www.idsia.ch/~giorgio/jncc2.html`.

NCC2, being based on imprecise probabilities, returns indeterminate classifications (i.e., several classes) when faced with doubtful instances, over which the NBC accuracy has been shown to sharply drop.

The paper covers all the software functionalities and presents several worked examples.

Acknowledgments

The Authors gratefully acknowledge partial support by the Swiss NSF grant 200021-113820/1 and by the Hasler Foundation (Hasler Stiftung) 2233 grant.

6 Changes from the previous release of JNCC2

The changes introduced in release 1.0 of JNCC2 can be summarized as follows:

- general clean up of the source code, now generally more readable and clear;
- reworked output format, more convenient if one runs the classifier on a large number of data sets;
- fixed a bug regarding writing to file the predictions when **validating via testing file with unknown classes**, which could sometimes cause to save to file classifications not consistent with the actual NCC2 output.

This user manual has been updated accordingly, to reflect the new features; moreover, we have introduced Section 3.3.3, which explains how to deal with multiple data sets. We also have update the bibliography as the main research paper describing NCC2 (former IDSIA Internal Report 08-07) has been eventually published by Journal of Machine Learning Research. We strongly encourage user of JNCC2 to read the paper published on JMLR rather than the technical report, as it has taken advantage from a thorough and careful review process.

References

Corani G, Zaffalon M (2008). "Learning Reliable Classifiers From Small or Incomplete Data Sets: The Naive Credal Classifier 2." *Journal of Machine Learning Research*, **9**, 581–621.

Domingos P, Pazzani M (1997). "On the optimality of the simple Bayesian classifier under zero-one loss." *Machine Learning*, **29**(2/3), 103–130.

Dougherty J, Kohavi R, Sahami M (1995). "Supervised and unsupervised discretization of continuous features." In A Prieditis, S Russell (eds.), "Proceedings of the 12th conference on machine learning," pp. 194–202. Morgan Kaufmann, San Francisco, CA.

Fayyad UM, Irani KB (1993). "Multi-interval discretization of continuous-valued attributes for classification learning." In "Proceedings of the 13th international joint conference on artificial intelligence," pp. 1022–1027. Morgan Kaufmann, San Francisco, CA.

Grünwald P, Halpern J (2003). "Updating probabilities." *Journal of Artificial Intelligence Research*, **19**, 243–278.

Heitjan D (1997). "Ignorability, sufficiency and ancillarity." *J. of the Royal Statistical Society, Series B*, **59**, 375–381.

Hsu C, Huang H, Wong T (2003). "Implications of the Dirichlet Assumption for Discretization of Continuous Variables in Naive Bayesian Classifiers." *Machine Learning*, **53**(3), 235–263.

Little RJA, Rubin DB (1987). *Statistical Analysis with Missing Data.* Wiley, New York.

Manski CF (2003). *Partial Identification of Probability Distributions.* Springer-Verlag, New York.

Walley P (1991). *Statistical Reasoning with Imprecise Probabilities.* Chapman and Hall, New York.

Witten IH, Frank E (2005a). *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann Publishers Inc,US.

Witten IH, Frank E (2005b). *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)*. Morgan Kaufmann.

Zaffalon M (2001). "Statistical inference of the naive credal classifier." In G de Cooman, TL Fine, T Seidenfeld (eds.), "ISIPTA '01: Proceedings of the Second International Symposium on Imprecise Probabilities and Their Applications," pp. 384–393. Shaker, The Netherlands.

Zaffalon M (2002). "Credal classification for mining environmental data." In AE Rizzoli, AJ Jakeman (eds.), "iEMSs 2002: Integrated Assessment and Decision Support (Transactions of the 1st Biennial Meeting of the International Environmental Modelling and Software Society)," pp. 72–77. iEMSs, Manno, Switzerland.

Zaffalon M (2005). "Conservative rules for predictive inference with incomplete data." In FG Cozman, R Nau, T Seidenfeld (eds.), "ISIPTA '05: Proceedings of the Fourth International Symposium on Imprecise Probabilities and Their Applications," pp. 406–415. SIPTA, Manno, Switzerland.

Zaffalon M, Wesnes K, Petrini O (2003). "Reliable diagnoses of dementia by the naive credal classifier inferred from incomplete cognitive data." *Artificial Intelligence in Medicine*, **29**(1–2), 61–79.

A: The ARFF data format

```
% 1.  Title:  Iris Plants Database
%
% 2.  Sources:
% (a) Creator:  R.A. Fisher
% (b) Donor:  Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
% (c) Date:  July, 1988
%
@RELATION iris
@ATTRIBUTE sepallength NUMERIC
@ATTRIBUTE sepalwidth NUMERIC
@ATTRIBUTE petallength NUMERIC
@ATTRIBUTE petalwidth NUMERIC
@ATTRIBUTE color {yellow,green,white}
@ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}
@DATA
5.1,3.5,1.4,0.2,white,Iris-setosa
4.9,3.0,1.4,0.2,yellow,Iris-setosa
4.7,3.2,?,0.2,yellow,Iris-setosa
4.6,3.1,1.5,?,green,Iris-setosa
5.0,3.4,1.5,0.2,green,Iris-versicolor
...
[other instances follow]
```

Figure 7: The (publicly available) `iris.arff` file. The variable "color" is not present in the original file, and has been introduced just to show an example of declaration of categorical variable.

The official documentation of the ARFF format can be found on the WEKA website.[4] In the following we explain however the ARFF format and provide some remarks specific for its use with JNCC2. An example of ARFF file is shown in Figure 7.

Comments can be introduced everywhere in the ARFF file, by letting a row begin with the character "%"; this makes it possible to document the data set (first rows of Figure 7).

The keyword of the header (`data`, `attribute`, `real`, `numeric`, etc; see later) are case-insensitive. The ARFF header begins with the declaration of the name of the data set: `@relation <relation name>` where <relation name> is a string (to be quoted if containing white spaces).

The subsequent lines declare the attributes of the data set (a different attribute for each line) as: `@attribute <attribute name> <data type>` where <attribute name> must start with an alphabetic character; if it contains white spaces, it has to be quoted. The <data type> field denotes whether the feature is numerical or categorical; in particular, it can be:

- "numeric" or "real" (the two strings are inter-changeable and indicate a numerical variable);
- if the variable is categorical, <data type> is constituted by the list of the categories (separated by commas), enclosed into braces; see for instance the declaration of variable "color".

After the header, there is a separating line containing the string "`@data`".

[4] http://www.cs.waikato.ac.nz/~ml/weka/arff.html

Then, the instances of the data set are written as comma separated values (an instance for each line); the order of the values should follow the order of the variable declarations. Missing values are represented by a single question mark.

When JNCC2 loads an ARFF file, it checks the consistency of the data with the header; if an inconsistency is found (for instance, a categorical variables that takes a value not declared in the header), JNCC2 exits pointing out a description of the data error.

Warnings

JNCC2 assumes the class to be the last declared feature; check this is the true also for your Arff file.

Unlike Weka, JNCC2 does not manage variables of type String or Date.

JNCC2 can incorrectly handle names (for instance, of features, categories or classes) containing white spaces, even if enclosed into quotes; check your ARFF file does not contain names with white spaces.