FID 3.5: Overview and Experimentation

Cezary Z. Janikow
Department of Mathematics and Computer Science
University of Missouri – St. Louis
St. Louis, Missouri 63121
janikowc@umsl.edu

Eryn R. Cantrell
Department of Mathematics and Computer Science
University of Missouri – St. Louis
St. Louis, Missouri 63121
eryn.cantrell@gmail.com

***Abstract*— FID is the original fuzzy decision tree, first introduced almost twenty years ago, that sparked a huge variety of hybrid algorithms merging approximate reasoning, fuzzy systems, and mainstream classification algorithms. With the continued interest, this paper describes a newly released update 3.5. One important new addition is a module that can be used to study the effect of noise and missing values on the performance of any classification system - something not well explored in the literature.**

Keywords – Fuzzy Decision Trees; Approximate Reasoning; FID

I. INTRODUCTION

As computing and storage capacities continue their rapid increase, the amount of available data grows enormously. Improved data collection and computerization of all aspects of life and commerce further increase the amounts of such readily available data. Therefore, methods and systems able to process data and provide some knowledge, or decisions, are more important than ever. Supervised classification programs extract knowledge from training data with known classifications. The knowledge is often in the form of an explicit data structure and some inference method. Among such systems, decision tree systems prove very popular and practical due to their conformity, comprehensibility, accuracy, and low processing and representation complexity [1].

Decision trees implement the recursive partitioning algorithm, which is a data-driven technique for building tree-based knowledge models [1]. The algorithm starts with all the training data in the root node, and it recursively selects a test, based on the available attributes, to split the node. Measures such as entropy and information gain guide the selection of usually a single attribute for the split, while some measures of complexity and potential tree pruning algorithms stop the process, balance tree size against accuracy, and prevent over fitting [1].

In addition to low algorithmic complexity to build and represent knowledge, a tree can also be easily visualized, interpreted, and even converted to rules. A simple inference rule can also be used to classify new data [1].

Many other systems, such as neural networks, offer very good classification performance as well but they generally lack the comprehensibility often sought by users [3]. However, such systems, often based on non-symbolic quantitative rules, generally offer better or more natural means of processing noise and missing values, problems inherently present in data.

Fuzzy systems and approximate reasoning attempt to bridge the gap between incomprehensible quantitative processing and comprehensible qualitative processing [4]. Fuzzy sets provide the basis for fuzzy representation, and together with fuzzy logic allow the modeling of language-related uncertainties, noise, and inexact data, while providing a symbolic framework for knowledge comprehensibility [3, 4].

Fuzzy Decision Trees proposed to combine fuzzy representation, and its approximate reasoning, with symbolic decision trees [3]. As such, fuzzy decision trees provide robust behavior for handling language-related uncertainty, noise, and missing or faulty features, while also providing comprehensible knowledge interpretation [3].

FID, the first software implementing these ideas, was originally proposed in [3], and since then it has prompted many researchers to follow such ideas of hybridization and mixing potentially incompatible representations and algorithms, resulting in many improvements and novel ideas [5, 6]. FID is still extensively used, given its plentiful parameters allowing testing of different fuzzy norms, working with fuzzy sets or exemplars, *etc*. We have just released a new version of the software, FID 3.5, which in addition to some needed upgrades also introduces two preprocessing modules: k-fold cross validation and induction of noise and missing values into data sets. These modules can be used to prepare data for any system operating on data represented by features.

In the paper, we describe the new version, and provide the algorithms for the two modules. The modules are included with FID 3.5, which is available for download [6]. We also show some results from using the modules with FID 3.5. Since few studies have been done and published on the effect of noise and missing values on the performance of any system, we hope that the availability of these modules will spark others to use them to study the behavior of their systems under such adverse conditions in directly comparable and controllable empirical settings.

II. FID

FID has three major components: one for partitioning continuous attributes, one for building an explicit tree, and one for knowledge inference from the tree [3]. Trees are built and represented using fuzzy sets, while inference can be done using either approximate reasoning or exemplar-based reasoning [2].

In classical set theory an element either belongs to a set or it does not. Such sets can be described with binary characteristic functions. A fuzzy set is represented by a linguistic label and a membership function onto the real interval [4], with much more flexible characteristic function. Similar to the basic operations of union, intersection, and complement defined in classical set theory, such operations are defined for fuzzy sets. Functions used to interpret intersection are denoted as T-norms, and functions for used to interpret union are denoted as S-norms. Originally proposed by Zadeh, min and max operators for intersection and union, respectively, define the most optimistic and the most pessimistic norms for the two cases, with many other options in between [2, 4].

A decision tree has an attribute tested in each of its internal nodes and has classification decisions associated with the leaves. The test in a node is interpreted as a fuzzy restriction, and its satisfaction is computed based on the attribute's feature of a sample. When moving down a tree path, the degrees of satisfaction of the fuzzy restrictions are accumulated according to the fuzzy T-norms. In FID, minimum, product, bounded product, and drastic product can be used. There is also the 'best' option, which chooses the best of the above [3]. When multiple leaves are satisfied, their decisions are combined using S-norms, with similar options available. However, multiple leaves can also be combined when interpreted as data exemplars [2].

FID processes data expressed with a mixture of nominal and continuous attributes. The continuous attributes can be pre-partitioned, or one of two internal partitioning algorithms can be used before tree building [2]. We believe that a separate apriori partitioning using some newer methods can benefit FID, and possibly we will implement additional partitioning methods in the near future.

A continuous attribute therefore has both the low-level domain and the partitioned domain. Data can be described using any mix of such features, mixing fuzzy or nominal labels, numeric values, and also allowing for missing values [3]. For example, assuming attributes Income (partitioned with linguistic labels Low, Medium, High), Employment (expressed as the number of hours per week), Sex (nominal), and classification for Credit (score 0-100, partitioned with three labels No, Maybe, Yes), the following are two examples of possible training data:

John: Income=12500 Employment=24 Sex=M Credit=No

Jane: Income=Medium Employment=? Sex=F Credit=25

FID processes missing values by using some statistical decomposition of data that includes missing features, while noise is naturally handled by the nature of its fuzzy representation and flexible approximate reasoning [2].

III. FID Updates

FID 3.5 offers several updates to the previous version. In addition to some general maintenance and issues of compatibility with new compilers, we also added cross validation and the ability to induce noise and missing values into a data set to simulate real world conditions. The version is available on the website [6]. Following the original FID, the new modules are invoked using either command-line arguments or through a GUI interface.

A. Cross Validation

For better statistical validity, and for better comparison, the results of any algorithm should be reproduced and averaged. However, this process is often not well explained in scientific literature and thus makes comparing results questionable. Cross validation is a well-known technique that can be statistically reproduced given only a single parameter.

The first update to FID 3.5 is the addition of k-fold cross validation with a single parameter k. If $k = 1$, then the program runs normally. If $k > 1$, the cross validation module partitions event data accordingly and then runs the main FID module k-times. The individually produced data files can be saved for inspection or for use with another algorithm. In detail, our algorithm is as follows:

1. The user specifies the number of folds (k) desired for cross validation of event data (E).
2. E is subdivided into k subsets/groups such that $E = \{G_1, \dots, G_k\}$ where:
   a. Events $\{e_1, \dots, e_n \in E\}$ are randomly added to each of the groups in such a way as to preserve the approximate frequency of the different decision classes.
   b. Each group is of roughly equal size: $|G_i| \sim |G_{i+1}|$ for $i = 1..k-1$. If the number of events is not evenly divisible by k, the remainder is dispersed one-by-one to groups starting with G_1.
   c. Each event from the original data is a member of one and only one group: $\{e \in G_i \mid e \notin G_j \; \forall j \neq i\}$.
3. For $i = 1..k$:
   a. V_i is the set of validation data such that $V_i = G_i$
   b. T_i is the set of training data such that: $T_i = \{E - V_i\}$
   c. FID is run using T_i as training set and V_i as testing/validation set. Output is saved to file (O_i).
4. Results are averaged from $O_1, \dots, O_k$ and saved to a summary file.

It is important to note that the partitioning of event data into groups occurs *before* anything else happens in the program. If $k > 1$, any other command line options entered are applied to the individual runs of FID on the training and validation sets T_i and V_i respectively. Lastly, this module can be used separately from FID's main algorithm. Partitioned event files can be prepared using FID and saved into multiple sets for cross validation by any algorithm.

B. Noise & Missing Values

Real world data is inherently noisy and incomplete, from factors such as errors, faulty measures, cost of performing tests, *etc*. [1,3]. Yet algorithms are generally introduced and discussed without detailed analysis of their behavior in such adverse conditions.

Noise or missing values can occur either in the training data, or in the application (testing) data. Generally, the training data can be cleaned up and processed more extensively, but the quality of application data is often less known. We thus added a module that can take any data and induce these conditions in a controllable way. Again, this module can be used solely to generate such data for any classification algorithm as the resulting data files are saved.

To add noise, the user must specify a few parameters: the noise amount, the probability for noise to occur on any given event, and a few options for how the noise can be applied. Anytime noise is added to *testing* data, it is actually done after the tree has been built, so only attributes used by the tree are affected.

FID can add noise to linear attributes only, nominal only, or to both. Nominal or categorical values are purely linguistic and do not have conceptual distance or numerical relationship to each other. If nominal features are included in the data, the noise probability is the independent probability of change for each nominal feature in an attribute. If a given value is selected for change, it will randomly switch to another possible value for that attribute. All possible values have the same probability, with the exception of the original value which is removed from the pool of options.

Linear continuous attributes have numerical and linguistic domains. For example, a 'grade' attribute could have possible values of 95, 'B+', 72.4, and 'C'. When adding noise to a linguistic value of a linear attribute, FID retrieves the centroid of that linguistic value (as defined by the attributes file), applies the noise algorithm, and replaces the original linguistic term with the new continuous value.

We implemented the noise algorithm for linear attributes in the following manner:

1. The user specifies the noise probability Pr_N such that $0 \leq Pr_N \leq 1$ and a noise amount Am_N such that $0 \leq Am_N \leq 1$.
2. A is the set of n linear attributes (columns) in any given data set such that $A = \{a_1, \dots a_n\}$. E is the set of m events (rows) in any given data set such that $E = \{e_1, \dots, m\}$. v_{ij} represents a value in event e_j for attribute a_i.
3. Every linear attribute has a lower and upper domain boundary denoted by α and β such that for a given attribute a_i, all its event values fall within its domain $\{x \in v_{i1}, v_{i2}, \dots, v_{im} \mid \alpha_i \leq x \leq \beta_i\}$.
4. For every value v_{ij} in any given attribute a_i, if v_{ij} is selected for noise based on the independent probability of noise parameter given by the user (Pr_N), the new value for v_{ij} is: $v_{ij} \leftarrow v_{ij} + \mathrm{N}(0, \sigma)$ where $\sigma = Am_N * (\beta_i - \alpha_i)$. In other words, v_{ij} will be modified by random variable drawn from a normal distribution in which one standard deviation is equal to the noise amount times the domain size. If the new value exceeds one of the domain boundaries, it is truncated to the boundary itself.

FID 3.5 also has the ability to remove features (i.e. add missing values to data). The user simply provides the probability of removing any given event value. This is Pr_{MV} such that $0 \leq Pr_{MV} \leq 1$. If an event value is selected by the algorithm to be removed, its value is changed to -1 representing missing features. Again, so manipulated data can be generated to be used in another algorithm.

C. Order of Processing

It is important to clarify where the noise and missing value manipulations occur with reference to discretization and cross validation.

If cross validation is used, the original data is separated multiple times into random training/testing sets, and each set is used independently. For each cross validation run (if no cross validation, the below are done just once):

1. Noise is applied to the training data if applicable
2. Some features are removed from the training data if applicable
3. Any continuous attribute without a discrete domain is then discretized
4. FID builds the tree using the training data
5. Noise is applied to the testing data (only to attributes used by the tree) as applicable
6. Some features are removed from the testing data (only on attributes used by the tree) as applicable
7. FID tests the tree using the testing data.

When using cross validation, noise, and/or missing values, FID saves all the modified files for the user to inspect and compare to the originals as desired. It is also possible to stop processing after step 2 or 3 above. This is useful for anyone wishing to analyze the performance of any algorithm operating on feature-based data in a systematic, repeatable and controllable way. In this case, a user simply selects an event file (training or testing data) and run the files through FID for noise, missing values, and/or discretization.

IV. Data Sets and Discretization

Given the updates made in FID 3.5, we decided to run some experiments to demonstrate its behavior under the newly developed cross validation, with various controllable amounts of noise and missing values.

We used six standard data sets from the UCI Machine Learning Repository [5]. Two of the sets had some missing values, but none of the others did. See Table 1 for a summary of the data sets. (The abbreviations under Attribute Types stand for Real, Categorical, and Integer types.)

All data sets contain some continuous attributes. For non-discretized linear attributes, we used FID's top-down discretization method as described in [2], with a maximum of four fuzzy sets.

Table 1. Summary of Data Sets

Dataset	Attributes	Attribute Types	Events	Classes	Missing Values
Breast Cancer - Diag.	30*	R	569	2	No
Glass Identification	9*	R	214	7	No
Statlog (Heart)	13	C, R	270	2	No
Ecoli	7*	R	336	8	No
Hepatitis	19	C, I, R	155	2	Yes
Breast Cancer - Orig.	9*	I	699	2	Yes

*: On the UCI website, these datasets listed identification numbers or decision classes as attributes. Our total number of attributes differs from the UCI website for this reason.

V. Experimental Results

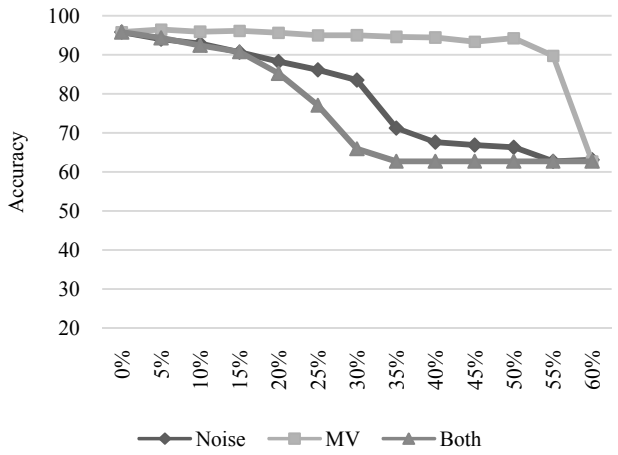
We ran many different experiments on each data set. A few notes before describing them. First, none of the data sets were pre-partitioned into training and testing sets, so we used our algorithm for cross validation as described earlier. For the experiments which artificially added random noise, the noise amount (Am_N) parameter is provided. The probability of noise (Pr_N) used was always 100% and noise was added to both linear and nominal attributes. For the experiments which added missing values, the missing value probability (Pr_{MV}) is provided. Unless otherwise mentioned, each experiment was executed in 10-fold cross validation, and averaged over 5 independent validations to account for potential skewed noise or missing value distributions. This gives a fairly confident result on the effect of noise and missing values on the performance of the algorithm.

First, we ran each data set using only 10-fold cross validation with no noise or missing values. This serves as a baseline for the performance of FID. Second, we induced progressive levels of noise (from 5% to 60% at 5% intervals) to each data set. Then we did the same with missing values. Finally, we gradually induced both noise and missing values simultaneously: noise and missing values both had the same amount at each interval. We actually performed each of these last three steps (adding noise, missing values, and both) a total of three times: first, we only modified training data (as partitioned by the cross validation process), then we only modified testing data, and finally we modified both.

Our experiments produced too many results to list each one. Most of the results were similar when introduced into training data, testing data, and to both. We have included here graphs (Fig. 1 through Fig. 6) for the results where we modified the training data only. We plan to make available all of the graphs in a technical report on our website.

As expected, FID behaves differently for different data sets, meaning that resistance to noise and missing values is as much a property of the algorithm as of the data. In some cases, the performance drops rapidly with noise and missing values, in other cases these adverse conditions hardly affect the

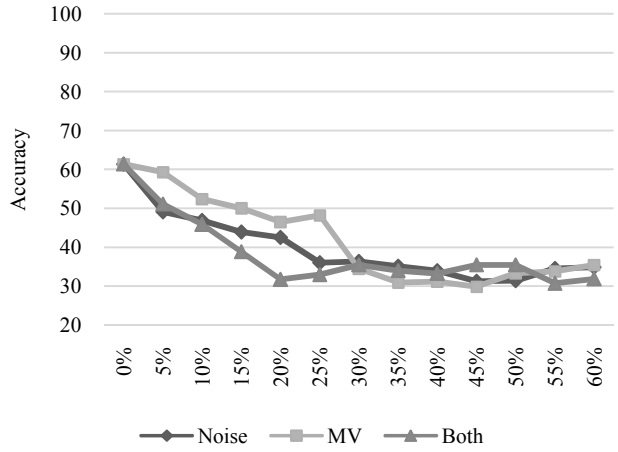
Fig. 1. Breast Cancer (D) - only training data modified



algorithm. This of course should be traced back to some properties of the data sets themselves, but we are not aware of any studies linking these together.

Also, as expected, the algorithm can handle missing values much better than noise in almost all cases. However, one unexpected observation is that in some cases adding missing

Fig. 2. Glass Identification - only training data modified



values to noise actually improves the performance over noise alone. This is a very interesting observation worth exploring further.

In almost all cases, noise and missing value levels close to 50% deteriorates the performance enough for all cases to converge to about random behavior. Some exceptions occurred, such as Hepatitis data (Fig. 5), which maintained steady performance with missing values and both noise and missing values (but not with just noise). We believe this occurred because there is enough redundant information in the data sets— given the size of the search space vs. the number of data points vs. the number of classes. We thus devised another experiment for the Breast Cancer – Diagnostic set. In this experiment, we progressively removed fractions of the data entries entirely rather than just removing feature values. The results are illustrated in Fig. 7. As speculated, the algorithm

performs the same until about 60% of the original data is missing, the same as for missing values.

VI. Conclusion

We have presented here a new version of FID, available for download from our web site. In addition to resolving some maintenance issues and improving the manual, the new version provides built-in mechanisms to study the performance of the algorithm on such adverse conditions as noise and missing values, which conditions can be induced in data in a controllable and predictable way and even saved for use in another algorithm. We hope this paper, and possibly the modules, will encourage more research on algorithm performance with dirty data, and on properties of data to handle such conditions.

Fig. 3. Statlog (Heart) - only training data modified

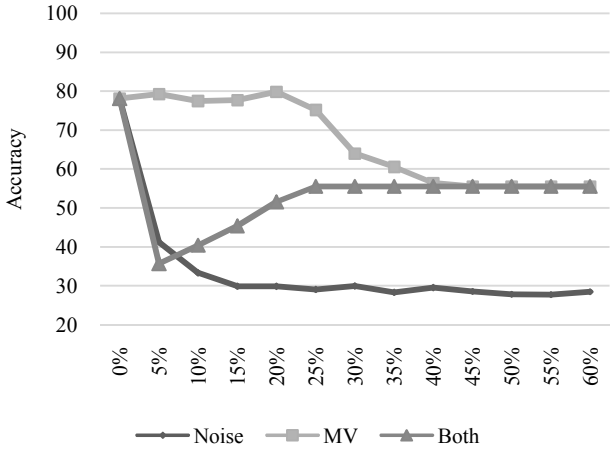


Fig. 4. Ecoli - only training data modified

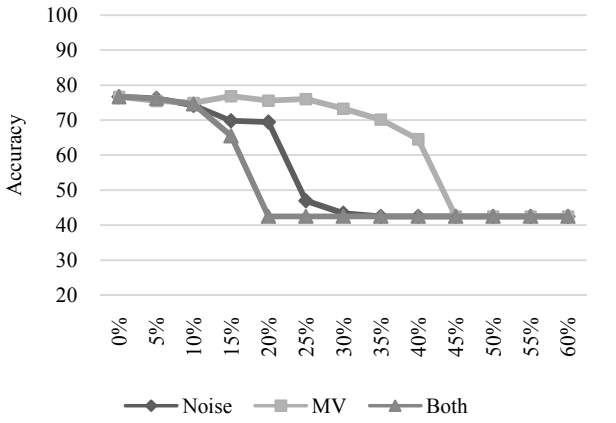


Fig. 5. Hepatitis - only training data modified

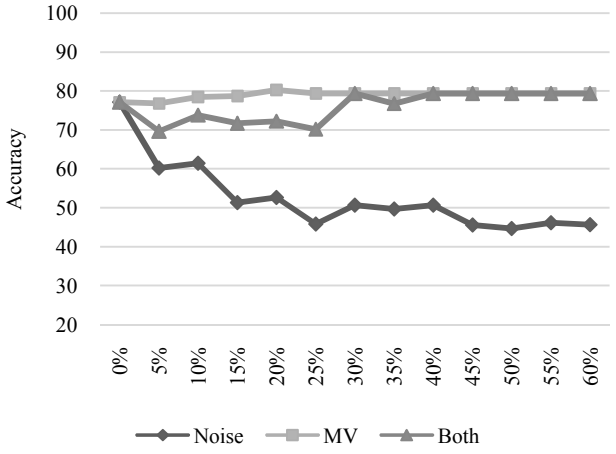


Fig. 6. Breast Cancer (O) - only training data modified

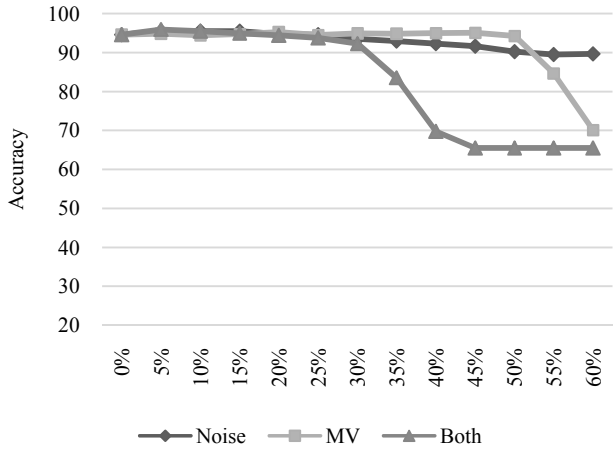
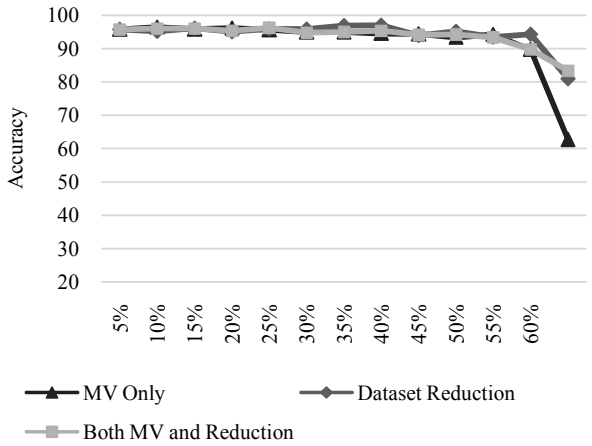


Fig. 7. Breast Cancer (D) - only training data modified



References

[1] J.R. Quinlan, *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[2] C.Z. Janikow and Krzysztof Kawa, "Fuzzy Decision Tree FID," in *Proceedings of Fuzzy Information Processing Society (NAFIPS) 2005 Annual Meeting*, Ann Arbor, MI, 2005, pp. 379-384.

[3] C.Z. Janikow, "Fuzzy Decision Trees: Issues and Methods," in *IEEE Transactions on Man, Systems, and Cybernetics*, Vol. 28, Issue 1, 1998 pp. 1-14.

[4] L.A. Zadeh, "Fuzzy logic and approximate reasoning," in *Synthese*, vol. 30, Issue 3-4, 1975, pp. 407-428.

[5] UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/.

[6] FID Website, C.Z. Janikow, http://www.cs.umsl.edu/~janikow/fid/.