SYNTHETIC DATA GENERATOR FOR TESTING OF CLASSIFICATION RULE ALGORITHMS

R. Seidlová[*], *J. Poživil*[†], *J. Seidl*[*], *L. Malecl*[†]

Abstract: We developed a data generating system that is able to create systematically testing datasets that accomplish user's requirements such as number of rows, number and type of attributes, number of missing values, class noise and imbalance ratio. These datasets can be used for testing of the algorithms designed for solving classification rule problem. We used them for optimizing of the parameters of the classification algorithm based on the behavior of ant colonies. But they can be advantageously used for other applications too. Program generates output files in ARFF format. Two standards and one user-define probability distributions are used in data generation: uniform distribution, normal distribution and irregular distribution for nominal attributes. To our knowledge, our system is probably the first synthetic data generation system that systematically generates datasets for examination and judgment of the classification rule algorithms.

Key words: *synthetic data, classification rule algorithm, ant colony data mining*

Received: May 14, 2014 **DOI:** 10.14311/NNW.2017.27.010
Revised and accepted: March 22, 2017

1. Introduction

One of the many algorithms based on artificial ant colonies behavior is classification rule algorithm known as AntMiner [19]. It turned out, that an ant-based classification rule's search is more flexible and robust than traditional approaches. For comparison of this algorithm with the others it is essential to quantify accuracy, time of calculation and number of rules. It is difficult to determine that one algorithm is better than the other since different algorithms usually use different testing datasets with certain constraints such as dimension, number and type of attribution, data distribution, number of class, missing values and so on. For the algorithms development and especially for optimization of the algorithm parameters, it appears necessary to use synthetic generated datasets. Although there are some databases repositories with a variety of datasets obtained from the real life

[*]Romana Seidlová – Corresponding author; Jaromír Seidl; Institute of Chemical Technology in Prague, Technická 5, CZ-166 28 Praha 6, Czech Republic, E-mail: seidlovr@vscht.cz, seidlj@vscht.cz

[†]Jaroslav Poživil; Lukáš Malecl; University of Business in Prague, Spálená 76/14, CZ-110 00 Praha 1, Czech Republic, E-mail: pozivil@vso-praha.eu, malecl@vso-praha.eu

environment, it is very difficult to obtain larger amount of very large datasets (with at least 10000 instances). Such datasets are crucial for recognition of differences between algorithms and for efficient optimization of their parameters.

Generating synthetic data allows to control the data distributions used for testing and can help us to allow a fair performance comparison among the algorithms.

This paper is organized into five sections. Section 2 outlines recent approaches to synthetic data generation. In the Section 3.1 proposed algorithm is described. Section 3.2 describes data set parameters affecting the accuracy of classification that were used at data generation. Illustrative example of data format provides Section 6. Results of computational experiments designed to evaluate the performance of AntMiner algorithm are described and analyzed in Section 4. Major conclusions are drawn in Section 5.

2. Related work on synthetic data generation

Recently, there is an increasing need for synthetic data generation, in various fields such as testing and developing enterprise application, demonstrating of them to a potential customer and testing of data mining application. It is very useful to generate data sets with known characteristic to assess whether or not data mining tools can discover those characteristics.

Commercial synthetic data generation products have recently become available [6, 8, 13]. These products do a good job producing moderate amounts of simply defined data. They have limited range of representation and there are some types of relations, functional dependencies and constraints that are not easy to describe using these commercial products [10, 11].

Data generation tools have been developed in the academic world as well [5, 12, 15]. These provide greater flexibility in the description and generation of synthetic data.

Most authors are focused on the association rule mining (ARM) problem. Cooper and Zito [7] think that a good choice for investigating of statistical properties of ARM algorithms is synthetic databases generated by the IBM QUEST program [22]. Jeske et.al. [14] describe a platform for the generation of shipping container data used for testing of data mining tools developed for port security.

Authors [18] propose a new scheme of knowledge-based classification and rule generation using a fuzzy multilayer perceptron. Paper [3] presents properties of network-based moving objects. Chosen approach is suitable for generating large data sets. Authors [1] present a data generator for generating synthetic complex-structured XML data, which can be of use to researches in XML data management.

Lately, some authors [2] deal with generating of synthetic data using a publicly available tool Benerator. Procedure is suitable for social, economic and demographic data. Authors of Lula University of Technology deal with synthetic data for hybrid prognosis [17].

Surprisingly, little work has been done on systematically generating artificial datasets that accomplish user's requirements such a number of missing values, class noise and imbalance ratio. Our work differs from previously published works in some aspects (the following ways). First, our data generation process is centered on multiclass classification rule problem, especially solved by ant colony algorithm.

Second, our tool generates more complex datasets – the user can select the datasets properties such number of missing values, class noise and imbalance ratio. To our knowledge, our system is probably the first synthetic data generation system that systematically generates datasets for examination and judgment of the classification rule algorithms.

3. Experimental part

3.1 Implementation and properties of synthetic data generator

Synthetic data generator is implemented in Microsoft Visual Studio 2015 which is suitable for development of graphical user interface applications with Windows Forms. The output data files (and also the input ones) files are in Attribute-Relation File Format (ARFF).

ARFF format contains two basic data types, nominal and numeric. Attributes of the date type are actually numeric. Attributes of string type are effectively nominal, although before using the strings are often converted to numerical values. Relational attributes contain independent sets of records that have elementary numeric and nominal attributes. For this reason, without loss of universality, the generator works only with these two basic data types.

In an effort to systematically generate test datasets for data analysis, we used two standard and one user-define probability distributions. For numeric attributes, we used uniform distribution and normal (Gaussian) distribution. For nominal attributes we considered uniform distribution or special user-defined distribution. This method provides the mechanism for that datasets are not only generated automatically but also controlled by the parameters from the user input.

The uniform distribution is the simplest continuous probability distribution. A random variable x has the uniform distribution if all possible values of the variable are equally probable. It is also called rectangular distribution. The uniform distribution is specified by two parameters: the end points a and b. The distribution has constant probability density on the interval (a, b) and zero probability density elsewhere.

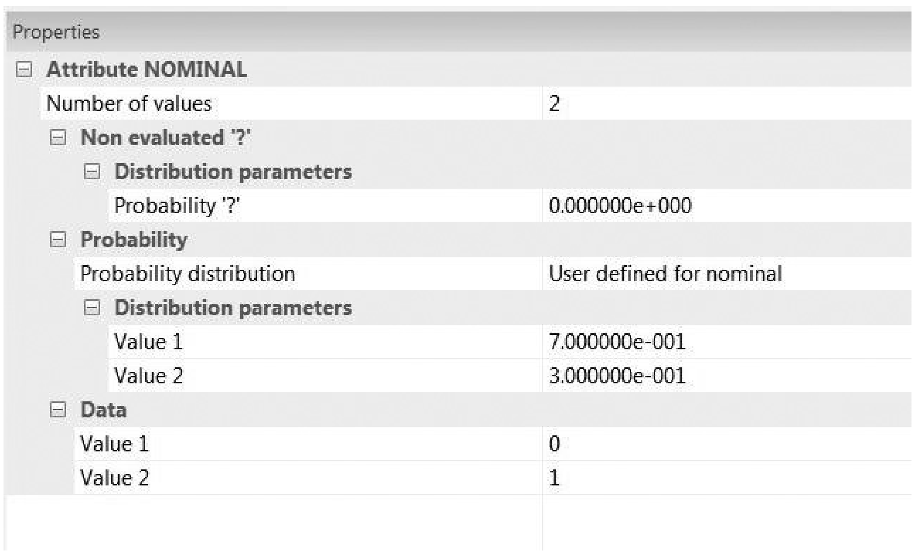
A continuous random variable x has a normal distribution or Gaussian distribution if its probability density function is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-(x-\mu)^2/2\sigma^2}, \tag{1}$$

where $x \in \mathbb{R}$, $\mu \in \mathbb{R}$ is mean and σ^2 is the variance.

Because these two distributions do not include all common cases, we further implemented irregular user-defined distribution for nominal attributes. For each attribute value, the user sets the probability of this value. For example, the proportion of smokers and nonsmokers is approximately 30:70. (See Fig. 1)

After starting the generator the user first sets attribute properties – their number, type of attributes, types of distribution for each attribute and a percentage rate of missing values. After that the user defines classification rules. The term classification refers to any context in which a decision or prediction is made based

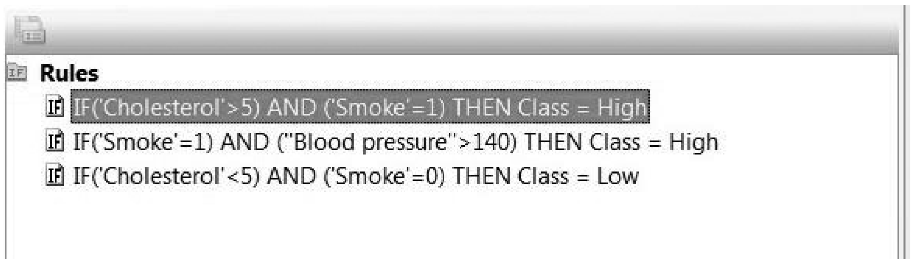


Properties	
⊟ **Attribute NOMINAL**	
Number of values	2
⊟ **Non evaluated '?'**	
⊟ **Distribution parameters**	
Probability '?'	0.000000e+000
⊟ **Probability**	
Probability distribution	User defined for nominal
⊟ **Distribution parameters**	
Value 1	7.000000e-001
Value 2	3.000000e-001
⊟ **Data**	
Value 1	0
Value 2	1

Fig. 1 *User-defined distribution input.*

on information available at any given time. A classification algorithm should be able to make such judgments in new situations. Each rule in Ant-Miner contains a condition part as the antecedent and a predicted class. The condition part is a conjunction of attribute-operator-value tuples. The classification rules have the form:

IF ⟨conditions⟩ THEN ⟨class⟩, where part of the ⟨conditions⟩ is a conjunction of several terms. Each term is a triplet ⟨attribute, operator, value⟩, for example ⟨smoker = yes⟩ or ⟨systolic blood pressure ≤ 140⟩. An example of user-defined classification rules is shown in Fig. 2).



```
Rules
  IF('Cholesterol'>5) AND ('Smoke'=1) THEN Class = High
  IF('Smoke'=1) AND ("Blood pressure">140) THEN Class = High
  IF('Cholesterol'<5) AND ('Smoke'=0) THEN Class = Low
```

Fig. 2 *Rules input.*

For generating of attributes according to the selected (chosen) probability distribution was used freely available library Boost Random Number Generator Library [16]. This library provides a framework for random number generators with well-defined properties so that the generators could be used in the demanding numeric and security domains. A uniform random number generator provides a sequence of random numbers uniformly distributed on a given range. The range can be compile-time fixed or available (only) after run-time construction of the object. Algorithm, which reads attribute types and generates values for each attribute is in Algorithm 1.

Algorithm 1 Read attribute type and generate random values of attributes.

```
INPUT Array of attributes
OUTPUT Array of attributes with values

const size_t numAttribute = m_vecAttribute.size();
CDistributionCalculationAttributePtrVector vecGenerAttribute;
vecGenerAttribute.resize(numAttribute);
for (size_t ind = 0; ind < numAttribute; ind++)
{
  CDistributionCalculationAttributePtr pNew;
  pNew = new CDistributionCalculationAttribute(m_vecAttribute[ind]);
  pNew −> GenerateValue();
  vecGenerAttribute[ind] = pNew;
 mapName.insert(tStringMap::value_type(vecAttribute[ind] −> name(),pNew));
}
```

Synthetic Data Generator is freely available at [21].

3.2 Parameters of classification rule datasets

The results of our experiments indicate that classification accuracy strongly depends on the complexity of a dataset [20]. We consider several parameters, which affect the accuracy: number of dataset instances, number of classes, number and type of attributes, percentage of missing values, class noise, imbalance ratio and information gain.

Imbalance ratio for two-class problems is a ratio between the number of majority class instances and the number of minority class instances. For a multi-class dataset can be defined as

$$I_{\mathrm{r}} = \frac{N_{\mathrm{c}} - 1}{N_{\mathrm{c}}} \sum_{i=1}^{\infty} \frac{I_{\mathrm{i}}}{I_{\mathrm{n}} - I_{\mathrm{i}}}, \qquad (2)$$

where I_{r} is in the range $(1 \leqq I_{\mathrm{r}} \leqq \infty)$ and $I_{\mathrm{r}} = 1$ is a completely balanced dataset having equal instances of all classes. N_{c} is the number of classes, I_{i} is the number of instances of class i and I_{n} is the total number of instances [23].

Noise is characterized by the proportion of incorrectly classified instances by a set of trained classifiers [4]. More complex, noise can be quantified as the sum of all off-diagonal entities (incorrectly classified instances) where each entity is the minimum of all the corresponding elements in the set of confusion matrices. The confusion matrix of the n-th classifier in a set of n classifiers can be generally represented as

$$C_n = \begin{pmatrix} i_1 1 & i_1 2 & \cdots i_1 j \\ i_2 1 & i_2 2 & \cdots i_2 j \\ \vdots & \vdots & \ddots \quad \vdots \\ i_i 1 & i_i 2 & \cdots i_i j \end{pmatrix}, \tag{3}$$

where the diagonal elements represent the correctly classified instances and off-diagonal elements are the incorrectly classified instances.

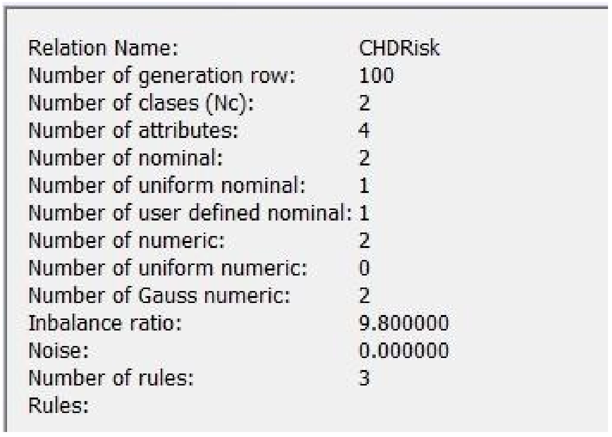
Information gain is an information-theoretical measure that evaluates the quality of attributes in a dataset [24]. It measures the reduction of uncertainty if the values of an attribute are known. For a given attribute X and a class attribute Y, the uncertainty is given by their respective entropies $H(X)$ and $H(Y)$. Then the information gain of X with respect to Y is given by $I(Y;X)$, where

$$I(Y;X) = H(Y) - H(Y|X). \tag{4}$$

After generating the records (see Algorithm 2), each record is evaluated according to the decision rules and relevant class ⟨class⟩ is determined (see Algorithm 3). After generating of a sufficient number of records, some records are removed so that the user selected imbalance ratio in the generated dataset would be preserved (see Eq. (2)).

After that records are artificially affected by "noise" (see Algorithm 4). Part of the records is modified so that their class is changed to another one (a wrong one). Percentage share of thus modified entries is selected by a user ("noise"). Finally, as many values for each attribute are randomly removed as there has been entered by the user. In the ARFF file these values are replaced by a question mark (see Algorithm 5).

In the Fig. 3 a generator box with dataset parameters is shown.



Relation Name:	CHDRisk
Number of generation row:	100
Number of clases (Nc):	2
Number of attributes:	4
Number of nominal:	2
Number of uniform nominal:	1
Number of user defined nominal:	1
Number of numeric:	2
Number of uniform numeric:	0
Number of Gauss numeric:	2
Inbalance ratio:	9.800000
Noise:	0.000000
Number of rules:	3
Rules:	

Fig. 3 *Listing of dataset parameters.*

Algorithm 2 Own generating of records.

```
  Generating of record in dataset using generated attributes from Algorithm1


  INPUT Vector of attributes with values from Algorithm 1
  OUTPUT Record of dataset

  bool bIsElseCondition = (!bIsCorrectCondition) && (pElse != nullptr) &&
                           (!pElse −> GetConsequentName().IsEmpty());
 if (bIsCorrectCondition | | bIsElseCondition)
 {
    CDistributionConsequentPtr pDistributionConsequent =
                                    new CDistributionConsequent;
    pDistributionConsequent -> m_VecCalc = vecGenerAttribute;

    if (bIsElseCondition)
    {
      pDistributionConsequent −> m_Consequent =
                                  pElse −> GetConsequentName();
    }
    else
    {
      pDistributionConsequent −> m_Consequent = strConsequent;
    }
    m_vecGeneratedData.push_back(pDistributionConsequent);
  }
```

4. Results and discussion

Ant-Miner algorithm uses some parameters in order to satisfy some constraints in iteration of the algorithm:

- Number of ants ($Numberofants$) indicates the maximum number of the ants involved in rule discovery process.

- Minimum number of cases per rule ($Minnumberofcases$): each rule must cover at least $Minnumberofcases$ to ensure a minimum generality.

- Maximum number of uncovered cases in the training set ($Maxnumberofuncoveredcases$): it is used as an ending constraint to terminate the main loop.

- Number of rules used to test convergence of the ants ($Numberofrules$): if the current ant has constructed a rule that is exactly the same as the rules constructed by the $Numberofrules$ -1 previous ants, then convergence has occurred. Therefore the current iteration of the main loop of the algorithm is stopped and the next iteration is started.

Algorithm 3 Detects records satisfied the classification rules and adjusts class.

```
Part of Algorithm 1


INPUT Attribute rules
OUTPUT IsCorrectCondition (BOOL)


bool bIsCorrectCondition = false;
for (auto pAttributeRule: attributeRules)
{
   CRuleItemPtrVector attributeItemRuleVec = pAttributeRule −> GetVec();
   for (auto pRule: attributeItemRuleVec)
   {
     bIsCorrectCondition = false;
     tStringMap::iterator itName =
                                mapName.find(pRule −> GetAttributeName());
     if(itName == mapName.end())
     {
       ASSERT(FALSE);
       break;
     }
     CDistributionCalculationAttributePtr pFind = itName −> second;
     ASSERT(pFind != nullptr);

     COleVariant oleGeneratedValue = pFind −> GetGeneratedValue();
     COleVariant oleRuleValue = pRule −> GetValue();

    bIsCorrectCondition = pFind −>GetARFFAttribute()−>GetARFFAttribute
                Record()
       −> CompareVal(oleGeneratedValue,pRule−>GetTypeCompare(),oleRule
                Value);
     if(!bIsCorrectCondition)
     {
         break;
     }
   }
   if (bIsCorrectCondition)
   {
     break;
   }
}
```

Algorithm 4 Generating of noise.

```
INPUT Noise, Records of dataset
OUTPUT Records of dataset modified with noise

ULONG numNoise = pInfo −> GetNoise() * numGenConswquent;
tVec vecNoiseIndex = GenerateRandom(numNoise, numGenConswquent);
for (size_t ind = 0; ind < vecNoiseIndex.size();ind++)
{
  tVec vecIndex = GenerateRandom(1, pClass −> GetNum());
  CString strConsequentNoise = pClass −>GetAtIndex(vecIndex[0]);
  ASSERT(!strConsequentNoise.IsEmpty());
  m_vecGeneratedData[vecNoiseIndex[ind]] −> m_ConsequentNoise =
                                            strConsequentNoise;
}
```

Algorithm 5 Generating of missing values.

```
INPUT Percentage of missing values, Records of dataset
OUTPUT Records of dataset with missing values

tVec vecMissNum;
vecMissNum.resize(numAttribute);
for (size_t ind = 0; ind < numAttribute; ind++)
{
  CDistributionAttributePtr pNon =
      m_vecGeneratedData[0] −> m_VecCalc[ind] −> GetARFFAttribute() −>
     GetARFFAttributeRecord() −> GetNonevaluatedAttribute();
  ASSERT(pNon != nullptr);
  vecMissNum[ind] = pNon −> m_Vec[0] * numGenConswquent;
}
typedef std::vector<tVec> tVecIndex;
tVecIndex vecMissNumIdex;
vecMissNumIdex.resize(numAttribute);
for (size_t ind = 0; ind < numAttribute; ind++)
{
  vecMissNumIdex[ind] =
          GenerateRandom(vecMissNum[ind], (numGenConswquent));
}
for (size_t ind = 0; ind < numAttribute; ind++)
{
  for (auto indexMiss: vecMissNumIdex[ind])
  {
    ASSERT(indexMiss < m_vecGeneratedData.size());
    ASSERT(ind < m_vecGeneratedData[indexMiss] −> m_VecCalc.size());
    m_vecGeneratedData[indexMiss] −> m_VecCalc[ind] −> SetMissed();
  }
}
```

- Evaporation factor ($Evaporationfactor$, $\rho \in$ (0,1)) represents the speed of evaporation of pheromones and avoids unlimited accumulation of pheromones on edges.

The greatest influences on the accuracy and calculation time have $Numberofants$ and $Evaporationfactor$ ρ. It was discovered that increasing evaporation factor will result in a slower convergence process and no significant increase in accuracy. The calculation time increases with the number of ants and with the evaporation factor. The two-dimensional plot Fig. 4 and Fig. 5 shows the influence these two parameters on execution time and accuracy and it is obviously that there is flat-maximum effect.

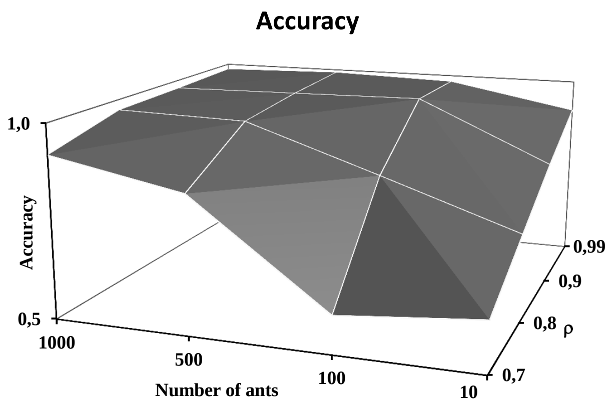


Fig. 4 *Influence of the number of ants and evaporation factor ρ on accuracy.*

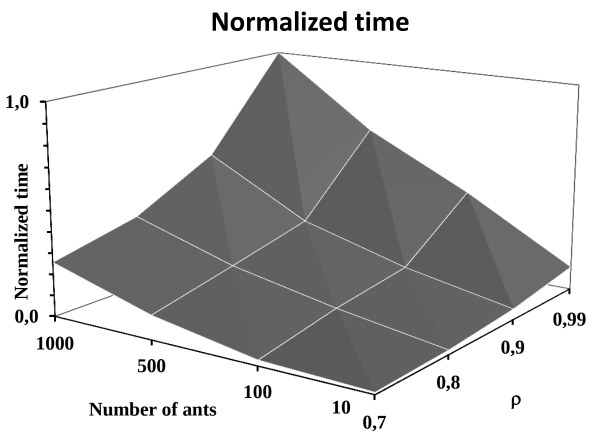


Fig. 5 *Influence of the number of ants and evaporation factor ρ on execution time.*

Synthetic data generator allows creating a model dataset for determining and comparing of the classification algorithms accuracies. In a real situation, generally there are not available appropriate amount of datasets with specified parameters.

Fig. 6 and Fig. 7 show an example of the influence of noise and missing values ratio on AntMiner algorithm accuracy in comparison with conventional algorithms.

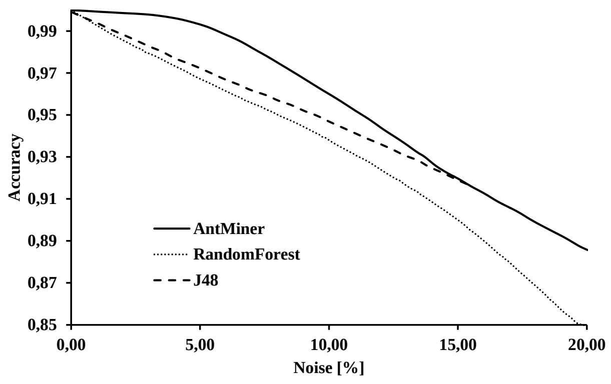


Fig. 6 *Influence of Noise value on AntMiner accuracy compared with "classical" algorithms.*

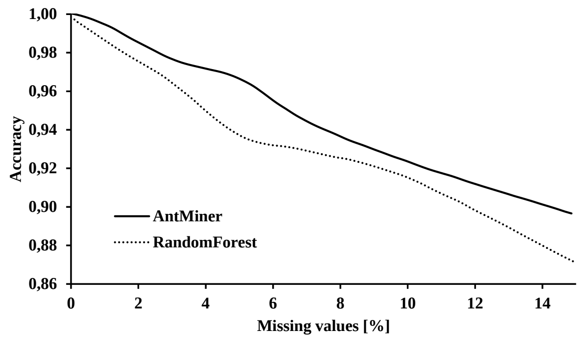


Fig. 7 *Influence of Missing value ratio on AntMiner accuracy compared with "Random Forest" algorithm.*

Comparative performance of method using ten-fold cross-validation was evaluated. N-Fold Cross validation is similar to Random Subsampling, the advantage of N-Fold Cross validation is that all the examples in the dataset are eventually used for both training and testing.

Each dataset was divided into ten partitions, and each method was run ten times, using a different partition as test set each time, with the other nine as training set. The rule list produced by a training set to predict the class of each case in the test set was used, the accuracy rate being calculated according to Eq. (5). Every rule list includes a default rule, which has no condition and takes the majority class in the set of training cases as its class, so that the default rule could be applied if none of the rules in the list covers the test case.

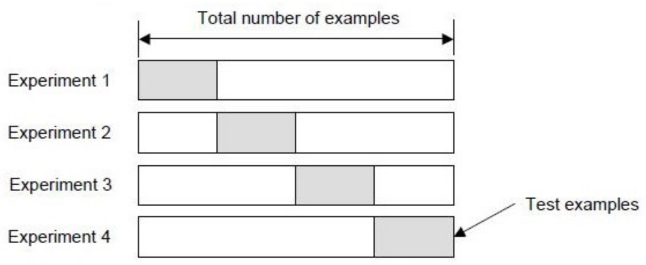


Fig. 8 *Schema of N-Fold Cross validation.*

The true error is estimated as the average error rate

$$E = \frac{1}{N}\sum_{i=1}^{N} E_i \tag{5}$$

5. Conclusions

Synthetic data generation is an interesting topic in data mining. In many research areas, a set of synthetic datasets is significant for quality assessment of a proposed algorithm. It is difficult to determine that one algorithm is better than the other since different algorithms usually use different testing datasets with certain constraints such as dimension, number and type of attribution, data distribution, number of class, missing values and so on. For the algorithms development and especially for optimization of the algorithm parameters, it appears necessary to use synthetic generated datasets. The need for the development of the generator arose from the inaccessibility sufficiently large datasets required for the optimization of algorithms. Our work concentrates on the generation of test instances for classification rules algorithms, especially algorithms based on artificial ant colonies. Developed software generates output files with specified parameters such as number of rows, number and type of attributes, number of missing values, class noise and imbalance ratio These datasets can be used for optimization of algorithm designed for solving classification rule problem or for comparison of these algorithms. To our knowledge, our system is probably the first synthetic data generation system that systematically generates datasets for examination and judgment of the classification rule algorithms.

References

[1] ABOULNAGA A., NAUGHTON J.F., ZHANG C. Generating Synthetic Complex-structured XML Data. In: WebDB'01 *Proceedings of the 4th International Workshop on the Web and Databases*, Washington DC, USA, 2001, pp. 79–84. Available from: https://pdfs.semanticscholar.org/961a/0433993ecbdc2b9551f82e11b390df7ab87b.pdf.

[2] AYALA-RIVERA V., MCDONAGH P., CERQUEUS T., MURPHY L. *Synthetic Data Generation using Benerator Tool* [online]. University College Dublin, 2013 [viewed 2016-10-01]. Technical report UCD-CSI-2013-03. Available from: https://csiweb.ucd.ie/files/UCD-CSI-2013-03.pdf.

[3] BRINKHOFF T. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*. 2002, 6(2), pp. 153–180, doi: 10.1023/A:1015231126594.

[4] BRODLEY C. E., FRIEDL M. A. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*. 1999, 11, pp. 131–167, doi: 10.1613/jair.606.

[5] BRUNO N., CHAUDHURI S. Flexible Database Generators. In: VLDB Endowment, publ. *Proceedings of the 31st VLDB*, Trondheim, Norway, 2005, pp. 1097–1107. Available from: http://dblp.uni-trier.de/db/conf/vldb/vldb2005.html.

[6] Centre, Pacific Business. DTM Database Tools [software]. [accessed 2016-10-01]. Available from: http://www.sqledit.com/.

[7] COOPER C., ZITO M. Realistic Synthetic Data for Testing Association Rule Mining Algorithms for Market Basket Databases. In: Springer Berlin Heidelberg *Knowledge Discovery in Databases: PKDD 2007: 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Warsaw, Poland, 2007, pp. 398–405, doi: 10.1007/978-3-540-74976-9_39.

[8] Global Software Applications, LLC. GSAPPS [software]. [accessed 2016-10-01]. Available from: http://www.gsapps.com/.

[9] HALL M., FRANK E., HOLMES G., PFAHRINGER B., REUTEMANN P., WITTEN I.H. The WEKA Data Mining Software [software]. [accessed 2016-10-01]. Available from: http://www.cs.waikato.ac.nz/ml/weka.

[10] HOAG J.E. *Synthetic Data Generation: Theory, Techniques and Applications*. Arkansas, 2008. PhD thesis, University of Arkansas. Available from: http://csce.uark.edu/~cwt/DOCS/STUDENTS/PHD/2007-12--PHD--Joe%20Hoag--Synthetic%20Data%20Generation%20-%20Theory,%20Techniques,%20and%20Applications--DISSERTATION.pdf.

[11] HOAG J.E., TTOMSON C.W. A Parallel General-Purpose Synthetic Data Generator. *ACM SIGMOD Record*. 2007, 36(1), pp. 19–24, doi: 10.1145/1276301.1276305.

[12] HOUKJAER K., TORP K., WIND R. Simple and Realistic Data Generation. In: DAYAL U., WHANG K. Y., LOMET D. B., eds. *Proceedings on Very Large Data Bases*, Seoul, Korea, 2006, pp. 1243-1246. Available from: https://pdfs.semanticscholar.org/99ec/2850819ab210dbad5c24247d31deaa853fa7.pdf.

[13] IRI, The CoSort Company. IRI RowGen [software]. [accessed 2016-10-01]. Available from: http://www.iri.com/products/rowgen.

[14] JESKE D.R., LIN P.J., RENDON C., XIAO R., SAMADI B. Synthetic data generation capabilties for testing data mining tools. In: IEEE Press Piscataway *MILCOM'06 Proceedings of the 2006 IEEE conference on Military communications*, New York, USA, 2006, pp. 3449-3454, doi: 10.1109/MILCOM.2006.302440.

[15] LIN P., SAMADI B., CIPOLONE A., JESKE D., COX S., RENDON C., HOLT D., XIAO R. Development of a Synthetic Data Set Generator for Building and Testing Information Discovery Systems. In: IEEE Computer Society *Proceedings of the Third International Conference on Information Technology: New Generations*, Las Vegas, USA, 2006, pp. 707–712, doi: 10.1109/ITNG.2006.51.

[16] MAURER J., WATANABE S. Boost C++ libraries [software]. [accessed 2016-10-01]. Available from: http://www.boost.org/doc/libs/1$_$61$_$0/doc/html/boost$_$random/reference.html.

[17] MISHRA M., LETURIONDO U., GALAR D. Synthetic data for hybrid prognosis. In: *Proceedings of the European Conference of the Prognostics and Health Management Society*, Nantes, France, 2014, pp. 796-801. Available from: https://www.phmsociety.org/sites/phmsociety.org/files/phm_submission/2013/phmce_14_086.pdf.

[18] MITRA S., DE R.K., PAL S.K. Knowledge-based fuzzy MLP for classification and rule generation. *IEEE Transactions on Neural Networks*. 1997, 8(6), pp. 1338–1350, doi: 10.1109/72.641457.

[19] PARPINELLI R.S., LOPES H.S., FREITAS A.A. An Ant Colony Algorithm for Classification Rule Discovery. In: R.A. SARKER, C.S. Newton ed. *Data Mining: Heuristic Approach. London: Idea Group Publishing*, 2002, pp. 191–208. Available from: http://lnfm1.sai.msu.ru/~rastor/Books/Abbass_&_Sarker_&_Newton-Data_Mining_Heuristic_Approach.pdf.

[20] SEIDLOVA R., POZIVIL J., HANTA V. Classification Rule Extracting with Ant Colony Algorithms. In: Slovak Society of Chemical Engineering *Proceedings of 39th International Conference of Slovak Society of Chemical Engineering*, Bratislava, Slovakia, 2012, pp. 664–671.

[21] SEIDLOVA R., SEIDL J. Synthetic Data Generator for Classification Rule Algorithms [software]. [accessed 2016-10-01]. Available from: `http://www.alefweb.cz/sdg.html`.

[22] Softland. IBM Quest Synthetic Data Generator [software]. [accessed 2016-10-01]. Available from: `http://ibm-quest-synthetic-data-generator.soft112.com/`.

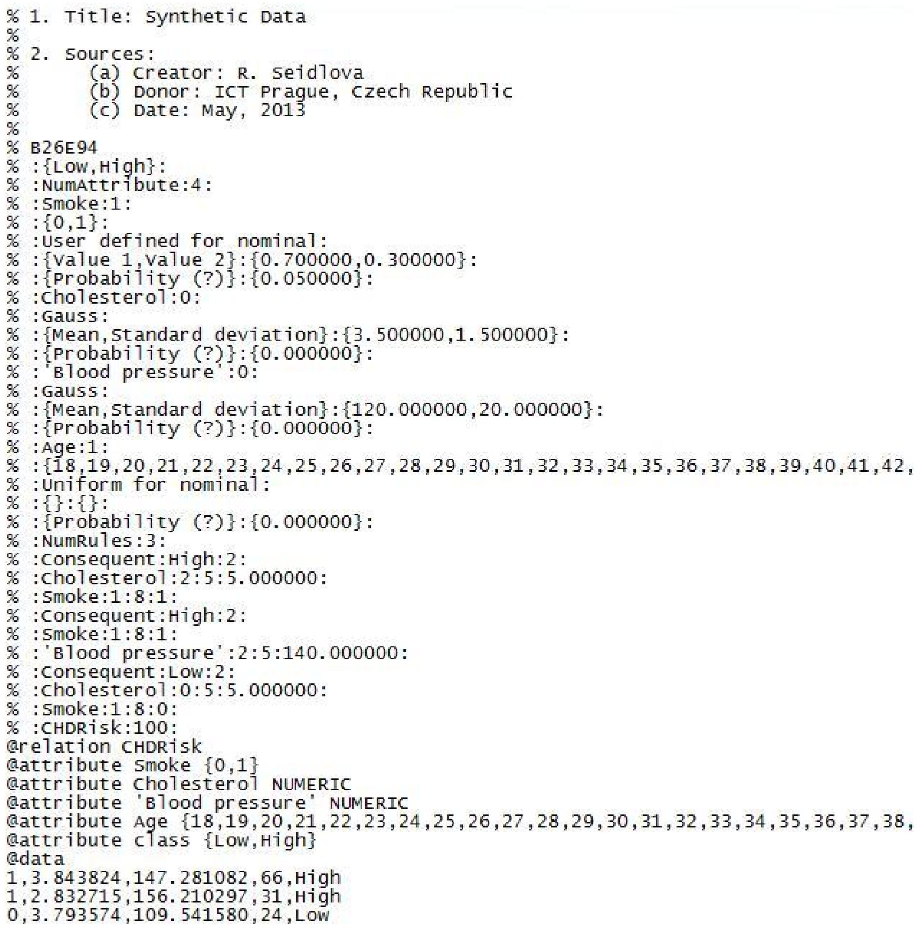
[23] TANWANI A.K., FAROOQ M. Performance evaluation of evolutionary algorithms in classification of biomedical datasets. In: *Genetic and Evolutionary Computation Conference, GECCO 2009*, Montreal, Canada, 2009, pp. 2617-2624, doi: `10.1145/1570256.1570371`.

[24] WITTEN I.H., FRANK E. *Data mining: practical machine learning tools and techniques*. San Francisco, Morgan Kaufmann, 2005. Available from: `ftp://ftp.ingv.it/pub/manuela.sbarra/Data%20Mining%20Practical%20Machine%20Learning%20Tools%20and%20Techniques%20-%20WEKA.pdf`.

6. Appendix – ARFF

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed within the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software [9]. ARFF files have two distinct sections. The first section is the Header information, which is followed by the Data information. The Header of the ARFF file contains the name of the relation (as the first line), a list of the attributes (the columns in the data), and their types. The Data section contains the data declaration line and the actual instance lines. Each instance is represented on a single line. Attribute values for each instance are separated by commas, and they are in the order in that they were declared in the header section. Missing values are represented by a "?" mark. An example of dataset is shown in the Fig. 9.

Lines that begin with a "%" mark are comments. We used these lines to store attributes information (name, type and distribution), class information and rules information (See Fig. 9)).



```
% 1. Title: Synthetic Data
%
% 2. Sources:
%      (a) Creator: R. Seidlova
%      (b) Donor: ICT Prague, Czech Republic
%      (c) Date: May, 2013
%
% B26E94
% :{Low,High}:
% :NumAttribute:4:
% :Smoke:1:
% :{0,1}:
% :User defined for nominal:
% :{Value 1,Value 2}:{0.700000,0.300000}:
% :{Probability (?)}:{0.050000}:
% :Cholesterol:0:
% :Gauss:
% :{Mean,Standard deviation}:{3.500000,1.500000}:
% :{Probability (?)}:{0.000000}:
% :'Blood pressure':0:
% :Gauss:
% :{Mean,Standard deviation}:{120.000000,20.000000}:
% :{Probability (?)}:{0.000000}:
% :Age:1:
% :{18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,
% :Uniform for nominal:
% :{}:{}:
% :{Probability (?)}:{0.000000}:
% :NumRules:3:
% :Consequent:High:2:
% :Cholesterol:2:5:5.000000:
% :Smoke:1:8:1:
% :Consequent:High:2:
% :Smoke:1:8:1:
% :'Blood pressure':2:5:140.000000:
% :Consequent:Low:2:
% :Cholesterol:0:5:5.000000:
% :Smoke:1:8:0:
% :CHDRisk:100:
@relation CHDRisk
@attribute Smoke {0,1}
@attribute Cholesterol NUMERIC
@attribute 'Blood pressure' NUMERIC
@attribute Age {18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,
@attribute class {Low,High}
@data
1,3.843824,147.281082,66,High
1,2.832715,156.210297,31,High
0,3.793574,109.541580,24,Low
```

Fig. 9 *Sample of ARFF syntax with smart using of comment lines.*