SandF: a DataEngine plug-in for Experiment Generation

Pilar García Díaz, Juan Ramón Velasco Pérez
DAEDALUS - Data, Decisions and Language, S.A.
Pirineos 27, 1º E-28040 Madrid Spain
Phone: +34-91-311-3386, Fax: +34-91-450-4058
email: jvelasco@daedalus.es

ABSTRACT: SandF is a DataEngine plug-in for getting samples of large Databases. When the size of working data is extremely large, data mining systems can be so slow in their processes. Sampling data is one of the possible techniques that can be used to deal with this problem. On the other hand, the selection of training and test data sets or cross validation folders is a frequent process for data analysis. For reliable results in the learning process, samples must be representative of the database; sampled data set should present the same properties (data distribution) as the original data.

KEYWORDS: Databases, data set, sampling, training and test, cross validation, randomly, stratification, DataEngine, plug-in, SandF.

INTRODUCTION

In data mining processes, it is useful and many times necessary, to deal with database samples. When a database has several million registers, working with samples of several thousands will be useful. Of course, these samples must constitute a good representation of the database, as the database must be a close approximation of the reality to be analyzed. The results of the learning process depend strongly on the quality of the samples. We must pay attention to the size of the samples and which elements we put in. The properties of samples (data distribution) must be as near as possible to the properties of the database.

To apply machine learning to a data set, we can select one of three general ways, no matter which learning algorithm is used: working with a single sample of the database, using Training and Test technique or running a Cross Validation method. Next we will comment on the three cases.

Getting a single sample is used when the data set is extremely large. To analyse the whole database will cost a great deal in terms of time and memory. In this way, you only expect to save time and memory in the process. Working with the sample will be faster and the results probably won't differ very much from the results obtained with the whole database. A disadvantage is that the results will be less precise. The conclusions will be less accurate because the algorithm learns with shorter data sets, so with less information. In any case, advantages usually weigh up more than disadvantages do.

The Training and Test technique always works with two data sets (see figure 1). The process has two sub-processes: first, the algorithm learns with the training data set. And second, the algorithm is executed on test data set for proving its accuracy for new data. Usually, training subsets are much larger than test subsets, 80% and 20% are the normal percentages for them. You have a more objective measure of error rate than with a single sample. This procedure reveals overfitting problem when the error for test data set is much larger than the error for training set.

Training with a single sample increases the probability of having biased situations and overfitting problems. In the first one, the algorithm balances its parameters according to the offered data, so it will work well for similar data. The second one means that the algorithm responds with high precision for the data set that it has trained, but very poorly for other new data sets.

The cross-validation method corrects some faults commented on before like bias and overfitting (Kohavi, 1995). This technique is much employed in machine learning. The figure 2 presents a general diagram of cross validation. The

method lets us make sure that our algorithm will operate right and its conclusions will be valid independently of data. It is based on Training and Test iterations. The main idea behind this is using all data to learn and all data to test. To achieve that we make a partition of the database by N disjoint subsets. Let us name them $\{A_1, A_2, ... A_N\}$. We will make N sub-processes in a parallel way. Each sub-process consists of a Training and Test operation with specific data sets. To the sub-process *i* corresponds the training set $\{\cup A_j \ \forall \ j \neq i\}$ and the test set $\{A_i\}$. This process is much more efficient because all data are used for training and test with different distributions.

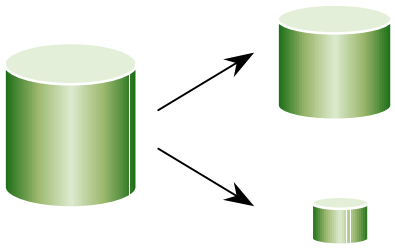

Figure 1: Training and Test technique

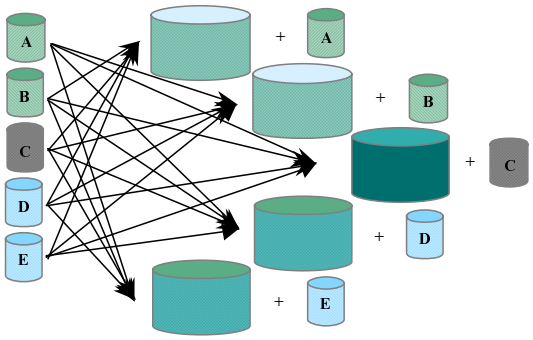

Figure 2: Cross Validation technique

Here we have revised the three general ways to sample a database in a data mining process. First, have a single sample. Second, construct training and test subsets. Third, perform a partition of database in multiple samples to applying Cross Validation.

In the next section, we will discuss first different ways to selecting data in the database to build a sample. Then we explain what SandF module is and how it works.

DATA SELECTION

The three general ways commented on before use one or more samples of data. A crucial question to attain reliable results is how to select the elements to construct the samples. There are several ways to select data and each of them has its properties. Therefore, we will choose the best for our application. Many times we need random sampling, in which the highest randomness degree is crucial. On the other hand, there are cases when it must be possible to repeat them so it is essential to use the same data samples with the same order elements in each. Comparing different learning algorithms is an example of the last situation. A statistical point of view of the sampling problem can be obtained from Domenech (1982), Neter et al. (1978) or Quesada et al (1994).

There are two ways of making a selection:

- According to the elements order in the database. If the sample keeps this order exactly, the sampling is named sequential sampling. On the opposite side, we have random sampling when the order in the sample is independent of the order in the database. There are intermediate situations, in which the two orders obey a specific function.

- According to some data properties. The properties are represented in a database by attributes. Making this kind of sampling, we obtain a sample with the same distribution as the database with regard to some attributes. The sampling is called stratified. When we don't take care of any properties data, we have no stratified sampling.

Let us consider a database with M_B elements and getting a sample with P% of the database. So our sample will have M_S elements of the original data set: $M_S = M_B*P/100$.

Sequential sampling is the easiest of all them. It consists of taking M_S consecutive elements of the database. We can pick out the first M_S elements, the last M_S elements or whatever M_S consecutive elements of the base. This method is applied when we need to know how the elements are chosen. This information can be useful to repeat the sampling on another occasion looking for the same data, for example, to compare the validity of different algorithms working with equivalent conditions.

The disadvantage is that the information is disturbed. Only when the elements in the database are a priori distributed at random, a sequential sampling has any negative effect. To think about this exception, you must be sure that the order of elements is independent of their own properties, like time or any attributes. So you can randomize the database in the preprocessing and then perform a sequential sampling without loss of precision.

In a regular sampling, the selection function of data is strongly dependent of the elements order in database. That means, one of every *x* consecutive elements in database is selected for the sample, and so on until the sample is completed. Sequential sampling is a particular case of regular sampling in the sense of in the simplest case x =1.

Regular sampling presents the same advantages, disadvantages and applications as sequential sampling. Figure 3A shows a partition scheme accomplished with regular sampling. You can observe that each sample collects one of five elements of database, always in the same order. So given an element in a sample, the next one is deductible.

Random sampling doesn't accept any dynamic to select data. Whether an element is contained in a sample or not is completely randomly. The main characteristic of this technique is that the information is not disturbed because the data don't have any treatment during the selection process.

Sometimes data in database are collected in a specific mode, so we need some preprocessing before analyzing them. For example, if we measure a property of a material according to temperature, probably we will heat it progressively and write the measurement from a lower to a higher temperature. We should make a mix of data before applying any learning process. Getting a random sample jumps this step, so data can be presented any order in the database.

The disadvantages, we spend more time to sampling. Direct applications to this kind of sampling are any one where don't require regular sampling, because we said above, no random data brings loss of information.

Finally, stratified sampling is shown in figure 3B. This kind of sampling is the nearest to getting representative samples of the database, that is, with the same data distribution as the database with regard to essential data properties. For example, in a data bank about census, a sample with same ratios of men and women is more reliable than another that doesn't take care about any features (sex, age, city, region...).

Given database to stratify in accordance with the feature *i*, we notice that there are N_i different values for this feature. Let us name them like n_1, n_2, n_3 … Then according to the feature *i*, we consider N_i types of elements in the database. Moreover, the database has the following composition respect of this attribute *i*: p_1% elements with type n_1, p_2% elements with type n_2, and so on. An ideal stratified sampling will produce samples with p_1% elements of type n_1, p_2% elements of type n_2, …. Many times, to fit the percent the percentage exactly is unfeasible because the percents not always tally with integer numbers of elements in the database. In practice, small deviations are irrelevant in sampling.

When the values of the attribute *i* are continuous numbers and they keep an order relation, then we can work with intervals instead discrete values. That means, we divide the range of possible values of feature *i* by N_i intervals. Each interval determines a type of element according to *i*. This processing simplifies the number of different types.

We must pay attention how many types are there (value of N_i) and which proportion ranges each type over the total. If there are many types of elements according to the attribute *i*, and in small percentages, then it is meaningless the stratification based on this attribute. In our example of census, it corresponds to stratify according to the first name or the passport number. Observe that the passport number is a *superkey* in the database and a stratified sampling wouldn't be efficient. We recommend working with values of N_i as the most of 10% of the database.

In the same way, we can talk about several features and make the sampling more complex. In this case, the type of element is defined by the values of the selected features, so the number of possible types is much larger. Like an example, we consider a new stratification according three features called *i, j*, and *k*. Assume that there are N_i, N_j and N_k intervals or discrete values for each feature, respectively. So initially, we had N_i types for *i*, N_j types for *j* and N_k types for feature *k*. In a triple stratification we will have $N_T=N_i*N_j*N_k$. We comment before that we shouldn't have a large N_T because we miss the sense of the stratified sampling, which is to have representative samples of the database. N_T shouldn't be larger than 10% of the database.

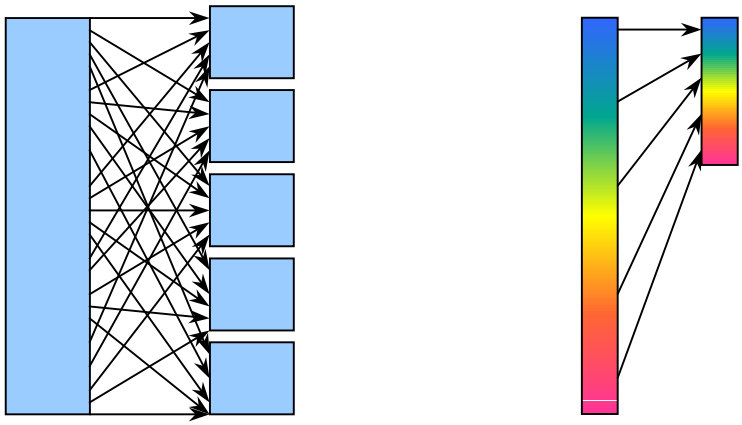

Figure 3: Selection Data: A) Partition with a regular sampling. B) Stratified sampling.

SANDF PLUG-IN

In this section, we explain the DataEngine plug-in SandF version 1.0. This module is executed in DataEngine like a *user-defined function block*. SandF performs the sampling of database. To use SandF, users should create a DataEngine card. DataEngine loads the input data matrix, and gives it to SandF. We are studying the chance to apply SandF over the database before execute DataEngine and offer directly the samples to the DataEngine system. This operation will take quite less processing time. The extern program uses OLE DB and ADO interfaces.

SandF module has a single input and one or more outputs. The database to sampling makes up the input data matrix and SandF gets one or more samples, writing them over the output data matrices. The plug-in has a configuration window where users introduce the parameters of sampling to execute.

SandF version 1.0 distinguishes the three different actions expounded before: getting a single sample of a database, obtaining two data collections for Training and Test, and the third, performing a partition of the database in several groups to applying Cross Validation.

SandF presents different types to selecting data, like regular sampling, random and stratified sampling. SandF version 1.0 considers only one discrete feature of input data to stratify the data. Next versions will present the possibility to stratify according several features, both continuous and discrete.

The configuration dialog is a sheet dialog with five pages. Two of them are shown in figures 4 and 5. In the first page, the user sets the type operation to accomplish: a partition of the database, getting a single sample or getting the training and test subsets. The second page (see next figure) contains edit boxes to insert the information about how many samples to take. If the operation is partition, then the program will ask for the number of groups to obtain. In the other two cases, SandF will ask for the size (percentage) of the samples.

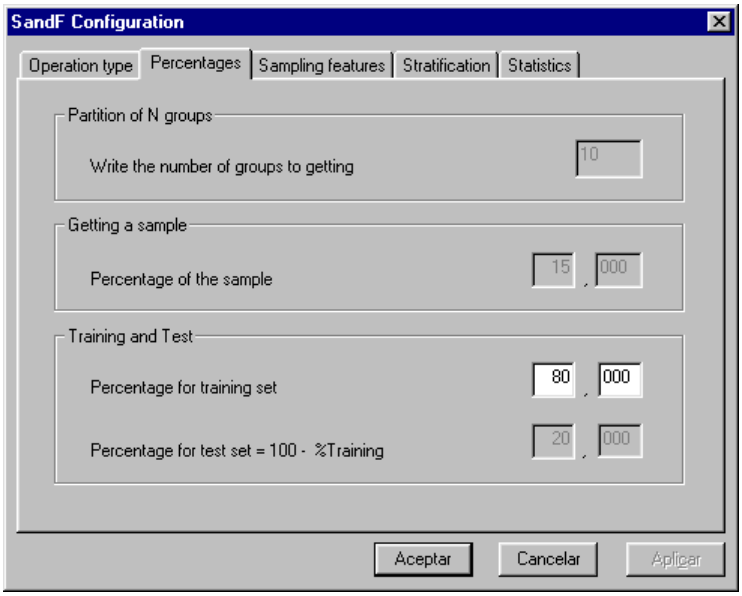

Figure 4: SandF Configuration, page “Percentages”

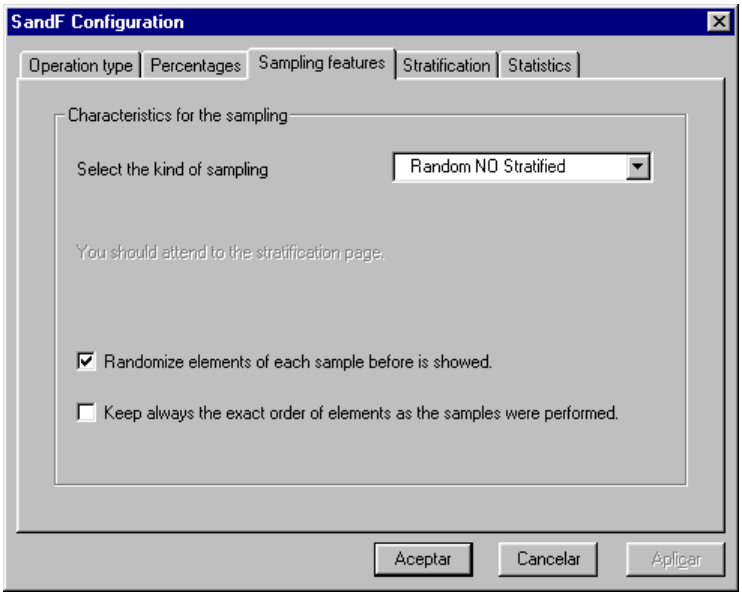

Figure 5: SandF Configuration, page “Sampling features”

The third page of the dialog sheet is shown in figure 5. Here users select what kind of sampling to perform: random/regular and yes/no stratified. Combining both classifications, we have four types of sampling:

- Random no stratified: all elements in database have the same probability to be chosen for the sample.
- Random stratified: each element of the same type has the same probability to be chosen.
- Regular no stratified: the output order of the elements in the sample will be the same as the order in database, but no strictly sequential.
- Regular stratified: the output order of elements of each type will be the same as in the database.

When you choose stratified sampling, a new page named “Stratification” appears. Choosing no stratified sampling this page will be eliminated of the dialog. Inside it, you write the features that determine the stratification. You must fill in this page before the configuration dialog is close. Otherwise the program will not run.

“Sampling features” page includes two exclusive options about how insert the elements in the final sample. They can be randomly distributed or they can be sequentially written as they were selected from the database. Users select one of the two check boxes. The second one is complementary with regular sampling to make sure that you can get the same samples running SandF over the same database and the same parameters.

SandF version 1.0 is protected against invalid values for the configuration parameters. If something during the process is wrong or inadequate, DataEngine will notify you with the appropriate message.

Finally, we show an example of Cross Validation in DataEngine. SandF make a partition of the database in four folders. Merge Rows function makes one set out of three. Then each new set is processed by the learning algorithm, in this case c4.5 with the RandT plug-in (Velasco and Somolinos, 1999).

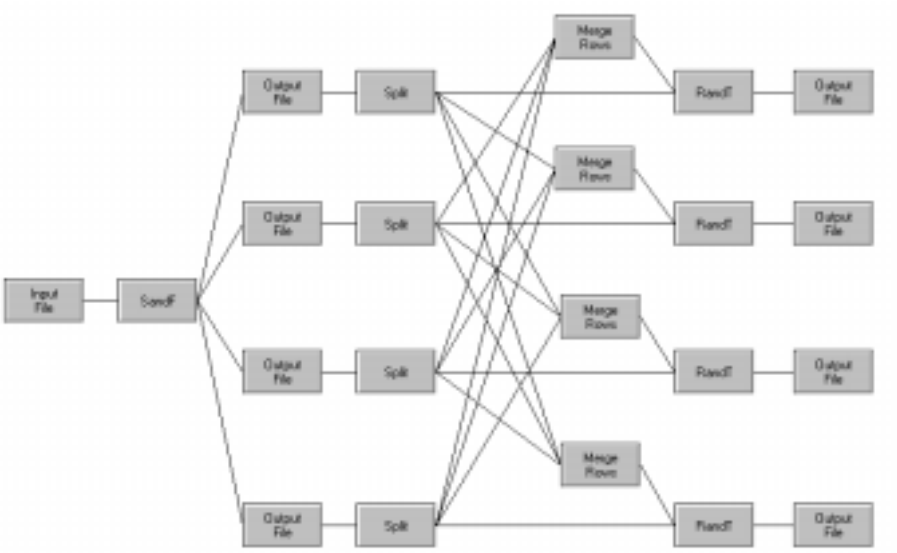

Figure 6: Cross Validation in DataEngine

CONCLUSIONS

The plug-in that is presented in this paper allows researchers to design different ways to access data. Sampling a large amount of data, selecting a training/test couple of data sets, or partitioning the information to apply cross-validation are usual tasks in data mining processes. This tool will make them easier.

REFERENCES

Domènech I Massons Josep M.1982. “Bioestadística. Métodos estadísticos para investigadores”. Herder. Barcelona.

Kohavi, R. “A study of cross-validation and Bootstrap for Accuracy Estimation and Model Selection”. Int. Joint Conf. on Artificial Intelligence, 1995 pp 1137-1143.

Neter John, Wasserman Willian, Whitmore G.A. 1978. “Applied Statistics”. Allyn and Bacon, Inc. Boston.

Quesada V., Isidoro A., López L.A. 1994. “Curso y ejercicios de Estadística”. Madrid.

Velasco J.R. and Somolinos, J.P. “RandT: A DataEngine Plug-in for Fuzzy Rules and Fuzzy Decision Trees Generation” 3rd International Data Analysis Symposium, Aachen, Germany.