A Two Stage Zone Regression Method for Global Characterization of a Project Database

J. J. Dolado, University of the Basque Country, Spain

D. Rodríguez, University of Reading, UK

J. Riquelme, University of Seville, Spain

F. Ferrer-Troyano, University of Seville, Spain

J. J. Cuadrado, University of Alcalá de Henares, Spain

Abstract

One of the problems found in generic project databases, where the data is collected from different organizations, is the large disparity of its instances. In this chapter, we characterize the database selecting both attributes and instances so that project managers can have a better global vision of the data they manage. To achieve that, we first make use of data mining algorithms to create clusters. From each cluster, instances are selected to obtain a final subset of the database. The result of the process is a smaller database which maintains the prediction capability and has a lower number of instances and attributes than the original, yet allow us to produce better predictions.

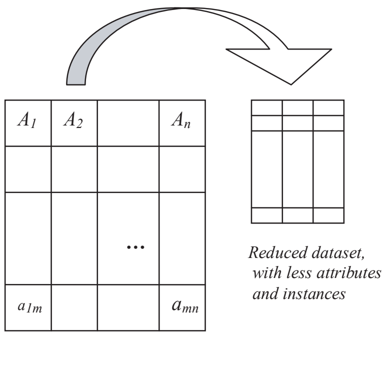
Introduction

Successful software engineering projects need to estimate and make use of past data since the inception of the project. In the last decade, several organizations have started to collect data so that companies without historical datasets can use these generic databases for estimation. In some cases, project databases are used to compare data from the organization with other industries, that is, benchmarking. Examples of such organizations collecting data include the International Software Benchmarking Standards Group (ISBSG, 2005) and the Software Technology Transfer Finland (STTF, 2004).

One problem faced by project managers when using these datasets is that the large number of attributes and instances needs to be carefully selected before estimation or benchmarking in a specific organization. For example, the latest release of the ISBSG (2005) has more than 50 attributes and 3,000 instances collected from a large variety of organizations. The project manager has the problem of interpreting and selecting the most adequate instances. In this chapter, we propose an approach to reduce (characterize) such repositories using data mining as shown in Figure 1. The number of attributes is reduced mainly using expert knowledge although the data mining algorithms can help us to identify the most relevant attributes in relation to the output parameter, that is, the attribute that wants to be estimated (e.g., *work effort*). The number of instances or samples in the dataset is reduced by selecting those that contribute to a better accuracy of the estimates after applying a version of the M5 (Quinlan, 1992) algorithm, called M5P, implemented in the Weka toolkit (Witten & Frank, 1999) to four datasets generated from the ISBSG repository. We compare the outputs before and after, characterizing the database using two algorithms provided by Weka, multivariate linear regression (MLR), and least median squares (LMS).

This chapter is organized as follows: the *Techniques Applied* section presents the data mining algorithm; *The Datasets* section describes the datasets used; and the *Evaluation of the Techniques and Characterization of Software Engineering Datasets* section discusses the approach to characterize the database followed by an evaluation of the results. Finally, the *Conclusions* section ends the chapter.

Figure 1. Characterizing dataset for producing better estimates



Techniques Applied

Many software engineering problems like cost estimation and forecasting can be viewed as *classification* problems. A classifier resembles a function in the sense that it attaches a value (or a range or a description), named the *class, C,* to a set of attribute values $A_1, A_2, \ldots A_n$, that is, a classification function will assign a class to a set of descriptions based on the characteristics of the instances for each attribute. For example, as shown in Table 1, given the attributes *size*, *complexity*, and so forth, a classifier can be used to predict the *effort*.

Table 1. Example of attributes and class in software engineering repository

A_1-*Size*	…	A_n- *Complexity*	*C - Effort*
a_{11}	…	a_{11}	c_1
…	…	…	…
a_{11}	…	a_{nm}	c_n

In this chapter, we have applied data mining, that is, computational techniques and tools designed to support the extraction, in an automatic way, of the information useful for decision support or exploration of the data source (Fayyad, Piatetsky-Shapiro, & Smyth, 1996). Since data may not be organized in a way that facilitates the extraction of useful information, typical data mining processes are composed of the following steps:

- **Data Preparation:** The data is formatted in a way that tools can manipulate it, merged from different databases, and so forth.
- **Data Mining:** It is in this step when the automated extraction of knowledge from the data is carried out. Examples of such algorithms and some usual representations include: C4.5 or M5 for decision trees, regression, and so forth.
- **Proper Interpretation of the Results:** Including the use of visualization techniques.
- **Assimilation of the Results.**

Within the available data mining algorithms, we have used M5 and linear regression classifiers implemented in the Weka toolkit, which have been used to select instances of a software engineering repository. The next sub-sections explain these techniques in more detail.

M5 and M5P

The main problem in linear regression is that the attributes must be numeric so that the model obtained will also be numeric (simple equations in *a* dimensions). As a solution to this problem, decision trees have been used in data mining for a long time as a supervised learning technique (models are learned from data). A decision tree divides the attribute space into clusters with two main advantages. First, each cluster is clearly defined in the sense that new instances are easily assigned to a cluster (leaf of the tree). The second benefit is that the trees are easily understandable by users in general and by project managers in particular. Each branch of the tree has a condition which reads as follows: *attribute* ≤ *value* or *attribute* > *value* that serve to make selections until a leaf is reached. Such conditions are frequently used by experts in all sciences in decision making.

Decision trees are divided into model trees in which each leaf represents the average value of the instances that are covered by the leaf and regression trees in which each leaf is a regression model. Examples of decision trees include a system called CART (Classification and Regression Trees) developed by Breiman (1984), ID3 (Quinlan, 1986) improved into C4.5 (Quinlan, 1993), and M5 (Quinlan, 1992) with the difference that in M5 the nodes represent linear regressions rather than discrete classes.

The M5 algorithm, the most commonly used classifier of this family, builds regression trees whose leaves are composed of multivariate linear models, and the nodes of the tree are chosen over the attribute that maximizes the expected error reduction as a function of the standard deviation of output parameter. In this work, we have used the M5 algorithm implemented in the Weka toolkit (Witten & Frank, 1999), called M5P. Figure 2 shows Weka's output for the M5P algorithm for one of the datasets that we used for this chapter. In this case, the M5P algorithm created 17 clusters, from LM1 to LM17. The normalized work effort (*NormWorkEff*) is the dependent variable, and a different linear model is applied depending on the number of Function Points (*FP*) and productivity (*NormPDR*). The clusters found can assign to the dependent variable either a constant or a linear equation (in the majority of the cases); in this case, each cluster or region is associated with linear equations (Figure 2, right column) In the example shown in Figure 2, the M5P algorithm created 17 leaves, and we will use *FP* and *NormPDR* to select the appropriate linear model. In this case, the tree

Figure 2. Weka's M5P output

```
FP <= 389.5 :
|   NormPDR <= 16.5 :
|   |   NormPDR <= 6.65 : LM1 (305/8.331%)
|   |   NormPDR >  6.65 : LM2 (318/5.402%)
|   NormPDR >  16.5 :
|   |   NormPDR <= 37.25 :
|   |   |   FP <= 147 : LM3 (124/5.751%)
|   |   |   FP >  147 : LM4 (107/10.878%)
|   |   NormPDR >  37.25 :
|   |   |   FP <= 161 :
|   |   |   |   NormPDR <= 56.9 : LM5 (34/8.323%)
|   |   |   |   NormPDR >  56.9 :
|   |   |   |   |   FP <= 57 : LM6 (9/17.555%)
|   |   |   |   |   FP >  57 : LM7 (34/27.858%)
|   |   |   FP >  161 : LM8 (37/58.384%)
FP >  389.5 :
|   NormPDR <= 8.05 :
|   |   FP <= 1520.5 :
|   |   |   NormPDR <= 2.65 : LM9 (62/3.559%)
|   |   |   NormPDR >  2.65 :
|   |   |   |   FP <= 610 : LM10 (50/6.205%)
|   |   |   |   FP >  610 :
|   |   |   |   |   NormPDR <= 5.7 : LM11 (38/5.155%)
|   |   |   |   |   NormPDR >  5.7 : LM12 (29/3.989%)
|   |   FP >  1520.5 : LM13 (47/23.679%)
|   NormPDR >  8.05 :
|   |   NormPDR <= 28.75 :
|   |   |   FP <= 1411.5 :
|   |   |   |   FP <= 674 : LM14 (67/11.599%)
|   |   |   |   FP >  674 : LM15 (45/26.176%)
|   |   |   FP >  1411.5 : LM16 (44/79.363%)
|   |   NormPDR >  28.75 : LM17 (40/160.629%)
```

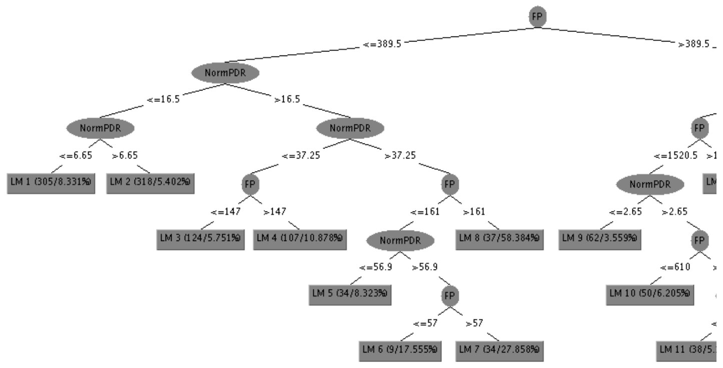
```
Where:

LM num: 1
NormWorkEff =
0.6701 * FP
+ 5.8478 * MaxTeamSize
+ 16.1174 * DevType=New_Development, Re-development
+ 16.4605 * DevPlatf=MR
+ 7.5693 * ProjElapTime
- 9.4635 * ProjInactiveTime
+ 0.4684 * TotalDefectsDelivered
+ 12.4199 * NormPDR
+ 461.1827
LM num: 2

...

LM num: 17
NormWorkEff =
22.1179 * FP
- 15457.3164 * VAF
+ 36.5098 * MaxTeamSizy
- 634.6502 * DevType=New_Development,Re-development
+ 37.0267 * DevPlatf=MR
+ 1050.5217 * LangType=2GL,3GL,ApG
+ 328.1218 * ProjElapTime
- 90.7468 * ProjInactiveTime
+ 370.2088 * NormPDR
+ 5913.1867
```

Figure 3. Graphical view of the M5P tree



generated is composed of a large number of leaves divided by the same variables at different levels. The tree could be simplified adding a restriction about the minimum number of instances covered by each leaf; for example, saying that there should be 100 instances per leaf will generate a simpler tree but less accurate.

Figure 3 also shows the tree in a graphical way. Each leaf of the tree provides further information within brackets. For example, for LM1, there are 308 instances and an approximate error in that leaf is 8.331%.

Constructing the M5 Decision Tree

Regarding the construction of the tree, M5 needs three steps. The first step generates a regression tree using the training data. It calculates a linear model (using linear regression) for each node of the tree generated. The second step tries to simplify the regression tree generated in the previous search (first post-pruning) deleting the nodes of the linear models whose attributes do not increase the error. The aim of the third step is to reduce the size of the tree without reducing the accuracy (second post-pruning). To increase the efficiency, M5 does the last two steps at the same time so that the tree is parsed only once. This simplifies both the number of the nodes as well as simplifying the nodes themselves.

As mentioned previously, M5 first calculates a regression tree that minimizes the variation of the values in the instances that fall into the leaves of the tree. Afterwards, it generates a lineal model for each of the nodes of the tree. In the next step, it simplifies the linear models of each node by deleting those attributes that do not reduce the classification error when they are eliminated. Finally, it simplifies the regression tree by eliminating subtrees under the intermediate nodes. They are the nodes whose classification error is greater than the classification error given by the lineal model corresponding to those intermediate nodes. In this way, taking a set of learning instances *E* and a set of attributes *A*, a simplified version of the M5 algorithm will be as follows:

```
Proc_M5 (E,A)
begin
R : = create-node-tree-regression
R : = create-tree-regression (E,A,R)
R : = simplify-lineal-models (E,R)
R : = simplify-regression-tree (E,R)
Return R
End
```

The regression tree, *R*, is created in a divide-and-conquer method; the three functions (create-tree-regression, simplify-lineal-models and simplify-regression-tree) are called in a recursive way after creating regression tree node by (create-node-tree-regression).

Once the tree has been built, a linear model for each node is calculated and the leaves of the trees are pruned if the error decreases. The error for each node is the average of the difference between the predicted value and the actual value of each instance of the training set that reaches the node. This difference is calculated in absolute terms. This error is weighted according to the number of instances that reach that node. This process is repeated until all the examples are covered by one or more rules.

Transformation of Nominal Attributes

Before building the tree, all non-numeric attributes are transformed into binary variables so that they can be treated as numeric attributes. A variable with *k* values is transformed into *k-1* binary variables. This transformation is based on the Breiman observation. According to this observation, the best splitting in a node for a variable with *k* values is one of the *k-1* possible solutions once the attributes have been sorted.

Missing Values

A quite common problem with real datasets occurs when the value of a splitting attribute does not exist. Once the attribute is selected as a splitting variable to divide the dataset into subsets, the value of this attribute must be known. To solve this problem, the attribute whose value does not exist is replaced by the value of another attribute that is correlated to it. A simpler solution is to use the prediction value as the value of the attribute selected or the average value of the attribute for all the instances in the set that do not reach the node, but can be used as the value of the attribute.

Heuristics

The split criterion of the branches in the tree in M5 is given by the heuristic used to select the best attribute in each new branch. For this task, M5 uses the standard deviation as a measure of the error in each node. First, the error decrease for each attribute used as splitting point is calculated.

Smoothing

In the final stage, a regularization process is made to compensate discontinuities among adjacent lineal models in the leaves of the tree. This process is started once the tree has been pruned and especially for models based on training sets containing a small number of instances. This smoothing process usually improves the prediction obtained.

Linear Regression and Least Median Squares

Linear regression (LR) is the classical linear regression model. It is assumed that there is a linear relationship between a dependant variable (e.g., effort) with a set of or independent variables, that is, attributes (e.g., *size in function points, team size, development platform,* etc.). The aim is to adjust the data to a model so that

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + + \beta_k x_k + e.$$

Least median squares (LMS) is a robust regression technique that includes outlier detection (Rousseeuw & Leroy, 1987) by minimizing the median rather than the mean.

Goodness of fit of the linear models is usually measured by the correlation, co-efficient of multiple determination R^2 and by the *mean squared error*. However, in the software engineering domain, the mean magnitude of relative error (*MMRE*) and *prediction at level l—Pred (l)*—are well known techniques for evaluating the goodness of fit in the estimation methods (see the *Evaluation of the Techniques and Characterization of Software Engineering Datasets* section).

The Datasets

The International Software Benchmarking Standards Group (ISBSG), a non-profit organization, maintains a software project management repository from a variety of organizations. The ISBSG checks the validity and provides benchmarking information to companies submitting data to the repository. Furthermore, it seems that the data is collected from large and successful organizations. In general, such organizations have mature processes and well-established data collection procedures. In this work, we have used release 8, which contains 2,028 projects and more than 55 attributes per project. The attributes can be classified as follows:

- Project context, such as type of organization, business area, and type of development;
- Product characteristics, such as application type user base;

- Development characteristics, such as development platform, languages, tools, and so forth;
- Project size data, which is different types of function points, such as IFPUG (2001), COSMIC (2004), and so forth; and
- Qualitative factors such as experience, use of methodologies, and so forth.

Before using the dataset, there are a number of issues to be taken into consideration. An important attribute is the quality rating given by the ISBSG: its range varies from A (where the submission satisfies all criteria for seemingly sound data) to D (where the data has some fundamental shortcomings). According to ISBSG, only projects classified as A or B should be used for statistical analysis. Also, many attributes in ISGSB are categorical attributes or multi-class attributes that need to be pre-processed for this work (e.g., the project scope attribute which indicates what tasks were included in the project work effort—planning, specification, design, build, and test—were grouped. Another problem of some attributes is the large number of missing instances. Therefore, in all datasets with the exception of the "reality dataset", we have had to do some pre-processing. We selected some attributes and instances manually. There are quite a large number of variables in the original dataset that we did not consider relevant or they had too many missing values to be considered in the data mining process. From the original database, we only considered the IFPUG estimation technique and those that can be considered very close variations of IFPUG such as NESMA.

We have used four datasets selecting different attributes including the one provided in the "reality tool" by ISBSG. In our study, we have selected *NormalisedWorkEffort* or *SummaryWorkEffort* as dependent variables. The normalized work effort is an estimate of the full development life cycle effort for those projects covering less than a full development life cycle while the summary work effort is the actual work effort carried out by the project. For projects covering the full development life cycle and projects where the development life cycle coverage is not known, these values are the same, that is, work effort reported. When the variable summary work effort is used, the dataset included whether each of the life cycle phases were carried out, such as, planning, specification, building and testing.

DS1: The reality dataset is composed of 709 instances and 6 attributes (*DevelopmentType, DevelopmentPlatform, LanguageType, ProjectElapsedTime, NormalisedWorkEffort, UnadjustedFunctionPoints*). The dependent variable for this dataset is the *NormalisedWorkEffort*.

DS2: The dataset DS2 is composed of 1,390 instances and 15 attributes (*FP, VAF, MaxTeamSize, DevType, DevPlatf, LangType, DBMUsed, MethodUsed, ProjElapTime, ProjInactiveTime, PackageCustomisation, RatioWEProNonPro, TotalDefectsDelivered, NormWorkEff, NormPDR*). The dependent variable for this dataset is the *NormalisedWorkEffort*.

DS3. The dataset DS3 is composed of 1,390 instances and 19 attributes (*FP, SummWorkEffort, MaxTeamSize, DevType, DevPlatf, LangType, DBMUsed, MethodUsed, ProjElapTime, ProjInactiveTime, PackageCustomisation, Planning, Specification, Build, Test, Impl, RatioWEProNonPro, TotalDefectsDelivered, ReportedPDRAdj*). In this case, we did consider the software life cycle attributes (*Planning, Specification, Build, Impl, Test*), and, therefore, we were able to use the summary work effort (*SummWorkEffort*) as the dependent variable.

DS4. The dataset DS4 is very similar to DS3 but it uses the unadjusted function points (*UnadjFP*) and the value adjustment factor (*VAF*) instead of the adjusted function points (*FP*). It is also composed of 1,390 instances. The 20 attributes are *VAF, SummWorkEffort, MaxTeamSize, DevType, DevPlatf, LangType, DBMUsed, MethodUsed, ProjElapTime, ProjInactiveTime, PackageCustomisation, Planning, Specification, Build, Test, Impl, RatioWEProNonPro, TotalDefectsDelivered, UnadjFP,* and *ReportedPDRAdj*. It also uses the summary work effort (*SummWorkEffort*) as the dependent variable.

Evaluation of the Techniques and Characterization of Software Engineering Datasets

We compare the benefits of the techniques by using linear regression and the least median square as prediction techniques before and after characterizing the database using the classical mean magnitude of relative error (MMRE) and Pred(%). In software engineering, the standard criteria for a model to be acceptable are *Pred(25)* ≥ 0.75 and *MMRE* ≤ 0.25.

- *MMRE* is computed as $\frac{1}{n} \cdot \sum_{i=1}^{n} \left| \frac{e_i - \hat{e}_i}{e_i} \right|$, where in a sample of size n, $\hat{e}_i$ is the estimated value for the *i-th* element, and e_i is the actual value.
- *Pred(%)* is defined as the number of cases whose estimations are under the *%*, divided by the total number of cases. For example, *Pred(25)=0.75* means that 75% of cases estimates are within the inside 25% of its actual value.

Figure 4. M5P output

```
            UnadjustedFunctionPoints <= 343 : LM1 (510/53.022%)
            UnadjustedFunctionPoints >  343 : LM2 (199/318.225%)
________________________________________________________________
where:

LM num: 1
            NormalisedWorkEffort =
                    90.5723 * DevelopmentPlatform=MF,MR
                    + 63.5148 * LanguageType=ApG,3GL,2GL
                    + 628.9547 * LanguageType=3GL,2GL
                    + 184.9949 * ProjectElapsedTime
                    + 10.9211 * UnadjustedFunctionPoints
                    - 545.8004

LM num: 2
            NormalisedWorkEffort =
                    10189.7332 * DevelopmentPlatform=MF,MR
                    - 5681.5476 * DevelopmentPlatform=MR
                    + 155.8191 * LanguageType=ApG,3GL,2GL
                    + 5965.379 * LanguageType=3GL,2GL
                    + 551.4804 * ProjectElapsedTime
                    + 4.3129 * UnadjustedFunctionPoints
                    - 8118.3275
```

We will now explain how we proceeded using the reality dataset as it is the smallest of the four datasets used. Once we had our datasets ready, we applied the M5P algorithm using the Weka Toolkit. The M5P algorithm created two clusters, LM1 and LM2 (the other three datasets created a much larger number of clusters). The *NormalisedWorkEffort* is the dependent variable and a different linear model is applied depending on the *UnadjustedFunctionPoints* variable. The clusters found can assign to the dependent variable either a constant or a linear equation (in most cases). For example, for the reality dataset, M5P has produced only two branches that are interpreted as follows: if *UnadjustedFunctionPoints* is less than 343 then we apply LM1 to calculate the *NormalisedWorkEffort* (see Figure 4).

The categorical data of the linear regression function obtained by Weka is calculated substituting the value for the appropriate value wherever it occurs. For example, if we had an instance with *DevelopmentPlatform* equals to MF, *LanguageType* equals to *ApG* and *UnadjustedFunctionPoints* less than 343 then the linear equation to apply would look like this:

```
LM num: 1
        NormalisedWorkEffort =
               90.5723 * MF=MF,MR
               + 63.5148 * ApG=ApG,3GL,2GL
               + 628.9547 * ApG=3GL,2GL
               + 184.9949 * ProjectElapsedTime
               + 10.9211 * UnadjustedFunctionPoints
               - 545.8004
```

For evaluating each categorical expression, if the value of the category on the left hand side is equal to any of the categories on the right hand side of the equation, then we substitute the entire equation with value 1; otherwise with the value 0. Following the example, we obtain:

```
LM num: 1
        NormalisedWorkEffort =
               90.5723 * 1, MR
               + 63.5148 * 1
               + 628.9547 * 0
               + 184.9949 * ProjectElapsedTime
               + 10.9211 * UnadjustedFunctionPoints
               - 545.8004
```

From each cluster only those instances that were within the 25% of the actual value, that is, *Pred (25)*, are selected to be part of the characterized database.

Afterwards, we applied LR and LSM in all datasets before and after selecting instances. In the case of the reality dataset, the number of instances was reduced from 709 to 139 projects. We also created another dataset by selecting 139 instances randomly from the entire dataset (709 instances). Table 2 compares the *MMRE*, *Pred(25),* and *Pred(30)* results for the reality dataset where the columns named *Before* are the results obtained using the entire dataset; *After* columns are the results when applying LR and LSM with only the selected instances. Finally, *Random* columns are the result when we randomly selected a number of instances equal to the number of instances of the characterized dataset (139 in the case of the reality dataset). For the reality dataset, M5P allowed us to reduce the number of instances in the dataset from 709 to 139 (570 instances).

Table 2. DS1 dataset results (reality dataset)

	LMS			***Linear Regression***		
	Before (790 inst)	***After (139 inst)***	***Random (139 inst)***	***Before (790 inst)***	***After (139 inst)***	***Random (139 inst)***
MMRE	1.07	0.19	0.98	2.41	0.44	3.18
Pred(25)	0.18	0.73	0.24	0.14	0.34	0.10
Pred(30)	0.23	0.79	0.26	0.18	0.41	0.12

Table 3. DS2 dataset results

	LMS			***Linear Regression***		
	Before (1390 inst)	***After (1012 inst)***	***Random (1012 inst)***	***Before (1390 inst)***	***After (1012 inst)***	***Random (1012 inst)***
MMRE	0.63	0.45	0.75	2.10	1.12	2.05
Pred(25)	0.39	0.36	0.34	0.22	0.23	0.23
Pred(30)	0.47	0.42	0.41	0.27	0.27	0.27

Table 3 shows the result for the DS2 dataset. M5P allowed us to reduce the number of instances in the dataset from 1,390 to 1,012 (378 instances).

Table 4 shows the results for the DS3 dataset, and the number of instance was reduced by 375.

Table 5 shows the results for the DS4 dataset. In this case, the number of instances is reduced by 411.

Although the general estimation accuracy of LMS and LR in the datasets used is quite low using the software engineering criteria, their performance is always improved when

Table 4. DS3 dataset results

	LMS			***Linear Regression***		
	Before (1390 inst)	***After (1015inst)***	***Random (1015 inst)***	***Before (1390 inst)***	***After (1015 inst)***	***Random (1015 inst)***
MMRE	0.68	0.46	0.81	2.35	0.88	2.09
Pred(25)	0.32	0.38	0.29	0.22	0.25	0.22
Pred(30)	0.38	0.46	0.36	0.26	0.26	0.25

Table 5. DS4 dataset results

	LMS			***Linear Regression***		
	Before (1390 inst.)	***After (979 inst)***	***Random (1015 inst)***	***Before (1390 inst)***	***After (979 inst)***	***Random (979 inst)***
MMRE	0.68	0.45	0.90	2.43	0.86	2.23
Pred(25)	0.30	0.37	0.30	0.22	0.27	0.19
Pred(30)	0.36	0.43	0.36	0.26	0.32	0.24

Table 6. Differences when using LMS

	Diff instances	***Diff MMRE***	***Diff Pred 25***	***Diff Pred30***
DS1	570	0.88	0.55	0.53
DS2	378	0.18	0.03	0.05
DS3	375	0.22	0.06	0.08
DS3	411	0.23	0.07	0.07

selecting a fewer number of instances using M5P. Table 6 shows the differences before and after selecting the instances. It is worth noting that the best improvement is in the case where the difference in the number of instances is large. This seems to be quite logical as the larger the number of instances discarded by the data mining algorithm, the cleaner the dataset should be.

Conclusions

In this chapter, we characterized 4 datasets created from the ISBSG database selecting both attributes and instances so that project managers can have a better global vision of the data they manage. To achieve this, we first created several subsets of the ISBSG database using expert knowledge to select attributes. We then made use of Weka's M5P data mining algorithm to create clusters. From these clusters, only those instances that were within the 25% of the actual value are selected to be part of the estimation model. When we compared the goodness of using linear regression and the least median square as prediction techniques using the mean magnitude of relative error (MMRE) and Pred(%), the smaller dataset produces better or as least similar results. The result is a new database which represents the original database but with fewer number of attributes and instances so that the project manager can get a much better grasp of the information of the database, improving the performance of the rest of activities.

Further work will consist of using data mining techniques for characterizing not only the instances but also the attributes (in this work, the attributes were selected manually using expert knowledge), by using bi-clustering. More needs to be done for understanding and comparing different clusterization techniques to create segmented models and analyzing its usefulness for project managers.

Acknowledgments

This research was supported by the Spanish Research Agency (CICYT TIN2004-06689-C03).

References

Breiman, L. (1984). *Classification and regression trees*. Chapman & Hall/CRC.

COSMIC. (2004). *COSMIC-FFP measurement manual, version 2.1.* Common Software Measurement International Consortium.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM, 39,* 27-34.

IFPUG. (2001). *Function point counting practices, release 4.1.1.* Manual. International Function Point Users Group.

ISBSG. (2005). *International Software Benchmarking Standards Group (ISBSG).* Retrieved from http://www.isbsg.org/

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81-106.

Quinlan, J. R. (1992). Learning with continuous classes. In the *Proceedings of the Second Australian Conference on Artificial Intelligence*, Singapore.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Rousseeuw, P. J., & Leroy, A. M. (1987). *Robust regression and outlier detection*. New York: Wiley.

STTF. (2004). *Software Technology Transfer Finland (STTF).*

Witten, I. H., & Frank, E. (1999). *Data mining: Practical machine learning tools and techniques with Java implementations*. San Francisco, CA: Morgan Kaufmann.