Improving Estimations in Software Projects with Data Mining Techniques

María N. Moreno García and Francisco J. García Peñalvo

Abstract

Estimation is a critical task in the software project management area. Early and reliable software estimations have a positive impact in software project scheduling, cost and quality. The drawbacks of the classical estimation methods used in numerous organizations have motivated the research in new techniques that avoid these problems. Some studies carried out have demonstrated that data mining algorithms provide predictive models with significant advantages with respect to traditional ones. In this work several data mining methods are described and used to create and validate models for software size estimation using attributes obtained in early stages of the life cycle.

Keywords*:* *Data mining, metrics, software size estimation.*

1. Introduction

Measurement is nowadays a usual management practice in mature software organizations since the information that it supplies is very valuable for making decision about many project management facets. Software metrics are used for two main purposes: evaluation and estimation. Metric data make software processes and products more visible and therefore, more controllable. They provide managers with assessments about software quality, use of resources, performance, plan compliance, etc. On the other hand, data collected from past projects can be used for building predictive models that relate early-stage software attributes with process and product features of later phases. These models allow the estimation of attributes such as quality, size, development effort and time for the actual project. Reliable and early measurement and estimation are decisive in software project management. Besides, poor estimates have a negative impact in software quality. This affirmation is corroborated by the significance that measurement and estimation have in the most recent quality standards.

Latest ISO (International Organization of standard) standards relative to quality management systems are the family of norms ISO 9000:2000 [ISO/IEC 9000, 2000], [ISO/IEC 9001, 2000], [ISO/IEC 9004, 2000], [ISO/IEC 90003, 2004]. Quality system requirements in ISO 9001:2000 are now distributed into four sections. One of them, "Measurement, analysis and it improvement", is dedicated exclusively to the measurement as an indispensable requirement for “continual improvement”. In the section of “management responsibility” explicit reference is made to the analysis of data on the performance of the quality management system. ISO/IEC TR 9126-3:2003 and ISO/IEC TR 9126-4:2004 standards provide specific metrics for measuring quality attributes.

With respect to the software process quality, models such as CMM [Paulk et to the., 1993], CMMI [CMMI-SW, 2002] and SPICE [ISO/IEC 15504, 1998, 1999, 2003, 2004], oriented to processes improvement, define principles and practices that drive to better software products and determine the maturity level according to the compliance of those principles. Software measurement appears in them as a key practice for achieving high levels of maturity. In these levels, measurement is not only used with evaluation purpose, but also with predictive purpose.

Many software companies implement software measurement and estimation as part of their management processes. However, most of them apply old methods that do not consider the best factors influencing the variable to be estimated, and they cannot be efficiently tailored to the peculiarities of the organization projects. In the last years, supervised data mining techniques are being used for addressing these drawbacks of classical estimation methods. The proposals in the literature have hardly been put into practice in software companies, in spite of the demonstrated utility and easy application of data mining methods in other areas such as commerce business, where they are widely used in order to improve competitiveness. In addition, the number of data mining tools available nowadays can facilitate the task.

The aim of this work is to emphasize the importance that early estimates have in the software projects as well as to show the advantages of data mining estimation models and their application facility. For illustrating that, several data mining algorithms are described and used for building and validating software size estimation models.

The rest of the paper is organized as follows. Section 2 summarizes existing related research. A shallow description of data mining techniques is presented in section 3. In Section 4 experimental data are explained. Section 5 includes the software size estimation models. Section 6 is dedicated to the visualization techniques and, finally, the conclusions are exposed in section 5.

2. Software size estimation

Software size estimation is a critical issue in the project management area. Good early estimations are essential for a reliable prediction of project effort and cost as well as for an efficient planning and scheduling.

Software size represents one of the most interesting internal attributes of a software product. Internal attributes can be measured in terms of the product itself, separate from its behaviour while external product attributes can be measured only with respect to how the product relates to its environment. The first ones, such as software size, are easier to measure than external ones. Software measurement presents the problem that it can only be carried out when the product is finished and then it is not very useful. For that reason, many metrics provide functional measures of the software size such as metrics of functions points [Albrecht, 1979], functions blocks [Hall et al., 2001], object points [Boehm, 1995] or Bang metrics [DeMarco, 1982]. Those variables obtained from software specifications should have a correspondence with the final product size expressed, for example, in lines of code. The recent *class point* approach [Costagliola et al., 2005], based on design documentation, is an alternative for object-oriented products.

Others estimations models as COCOMO [Boehm, 1995] use define links between such early software measures and the final product size or development time and effort.

Most of the classical estimation methods are based in regression techniques whose main weakness is the requisite of formulating a hypothesis about the relevant attributes for estimating the target variable and about the behavior of these attributes. The hypothesis is usually done in the form of a mathematical expression that relates the attributes with the variable to be estimated. The expression includes a number of parameters whose values are discovered in the regression process. Data mining techniques do not need any hypothesis due to the traditional verification oriented data analysis is transformed in an approach focused on the knowledge discovery. This work shows several data mining methods which we used to build and validate models for software size estimation. We processed data from 42 projects involving more than 100,000 lines of code (LOC) of a fourth-generation language. A component-based method [Verner and Tate, 1992] and a global method [Symons, 1991] were taken as reference [Moreno et al. 2002], [Moreno et al. 2004].

3. Data Mining Techniques

Data mining represents a shift from verification-driven data analysis approaches to discovery-driven data analysis approaches. In the former approach, a decision maker must hypothesize the existence of information of interest, collect this information, and test the posed hypothesis against the information collected. Discovery-driven approaches sift through large amounts of data and automatically (or semi-automatically) discover important information hidden in the data [Brykczynskki, 1999].

Data mining problems can be resolved by employing supervised and unsupervised algorithms. In the first case, a learning phase is necessary to build a predictive model from historical labeled data records, which is used later to make predictions about new, unlabeled data. Unsupervised algorithms are used in knowledge discovery modeling. This is a descriptive task whose objective is to detect patterns in actual data without need of previous learning.

In predictive modeling there is a special attribute called the "label" that one intends to predict. By encoding the relation between the label and the other attributes, the model can make predictions about new, unlabeled data. The two most common supervised modeling methods are classification and regression. If the label is discrete, the task is called classification; if the label is continuous, the task is called regression.

The goal in *knowledge discovery* modeling is to discover rules and segments of the data that behave similarly (clusters). These are **unsupervised** tasks. Unsupervised modeling is a descriptive task, not a predictive task. Associations and clustering are two unsupervised modeling tasks.

The techniques listed in Table 1 are classified in these categories.

Supervised	**Unsupervised**
Decision tree	Deviation detection
Neural induction	Clustering
Regression	Association rules
Time series	Sequential pattern discovery

Table 1. Data mining techniques

Data mining algorithms can be complemented with data visualization techniques taking advantage of the human brain's amazing pattern recognition capability.

Usually the data is not organized in a way that will facilitate automated or semi-automated knowledge induction. Also, there may be irrelevant, missing, noisy and uncertain data in this raw data set.

4. Description and statistical treatment of the data

This section describes the data set used to build the models. It was obtained from experiments carried out by Dolado [Dolado, 2000]. The data originate from academic projects in which students developed accounting information systems that have the characteristics of commercial products. Each system includes some or all of the following subsystems: sales, purchases, inventories, financial statements, and production cycles. We have analyzed data from 42 projects written in Informix-4GL.

The information available was divided in two groups. One group containing global data from each project (42 records) and other group containing attributes from each component of the projects (1537 records). The code of the applications was classified into three types of components according to Verner and Tate [Verner and Tate, 1992]

Component attributes:

TYPECOMP: type of component (1: menu, 2: input, 3: report/query)

OPTMENU: number of choices (only for menus)

DATAELEMENT: number of data elements (only for inputs and reports/queries)

RELATION: number of relations (only for inputs and reports/queries)

LOC: lines of code

The distribution of attribute values for every type of component is shown in figures 1, 2 and 3. Note that most of the records correspond to small size components.

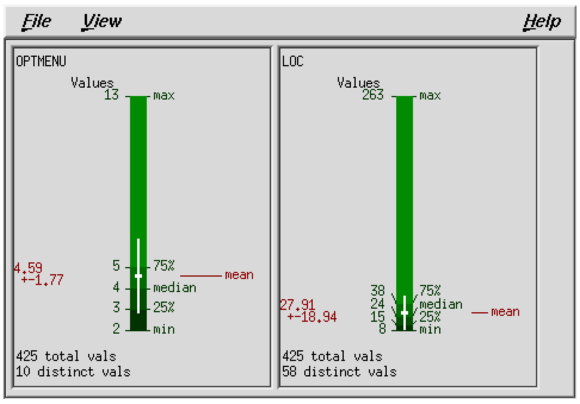


Figure 1. Statistical study for menus

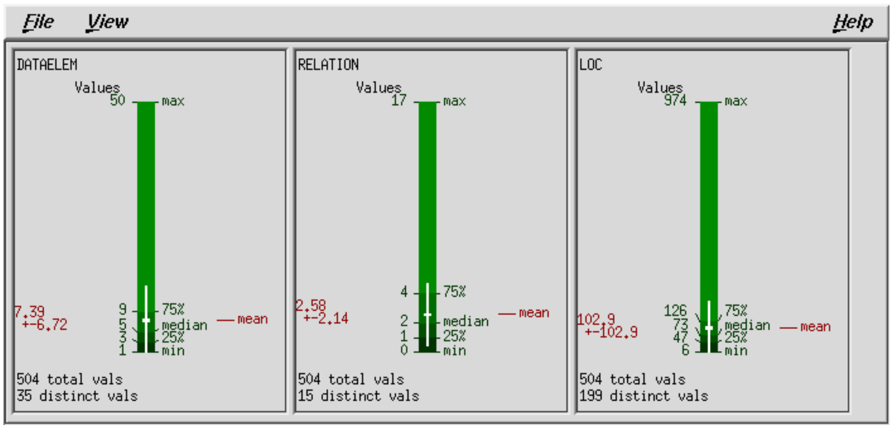


Figure 2. Statistical study for inputs

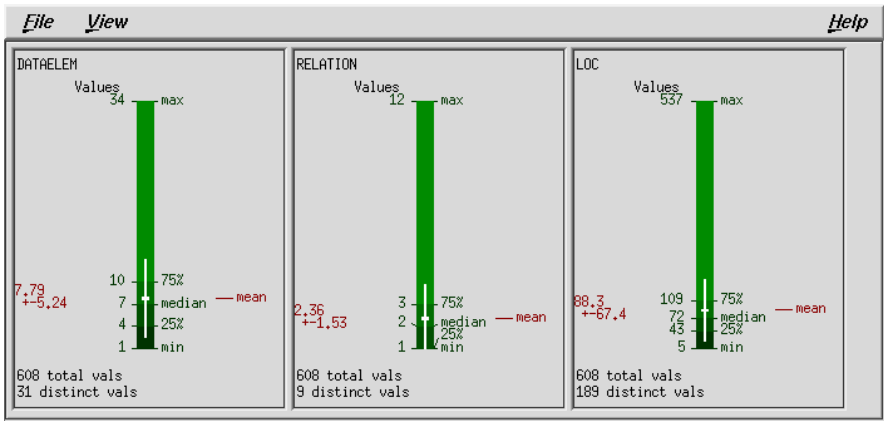


Figure 3. Statistical study for reports/queries

Project attributes

The attributes described below are used in Mark II method [Symons, 1991] to calculate functions points. That proposal consists of considering a system composed of logical transactions. A logical transaction is a unique input/process/output combination triggered by a unique event of interest to the user or a need to retrieve information. The central concept in this method is that of entity, that replaces the concept of logical file.

LOC: lines of code

NOC: number of components

NTRNSMKII: number of transactions MKII

INPTMKII: number of total inputs

ENTMKII: number of referenced entities

OUTMKII: number of total outputs (data elements over all transactions)

UFPMKII: number of unadjusted functions points MKII

The statistical study about these attributes is shown in figure 4. The projects have between 726 and 8,888 LOC. There are more records with low values of the attributes. However, high values have sufficient weight to influence in the induction of the models.

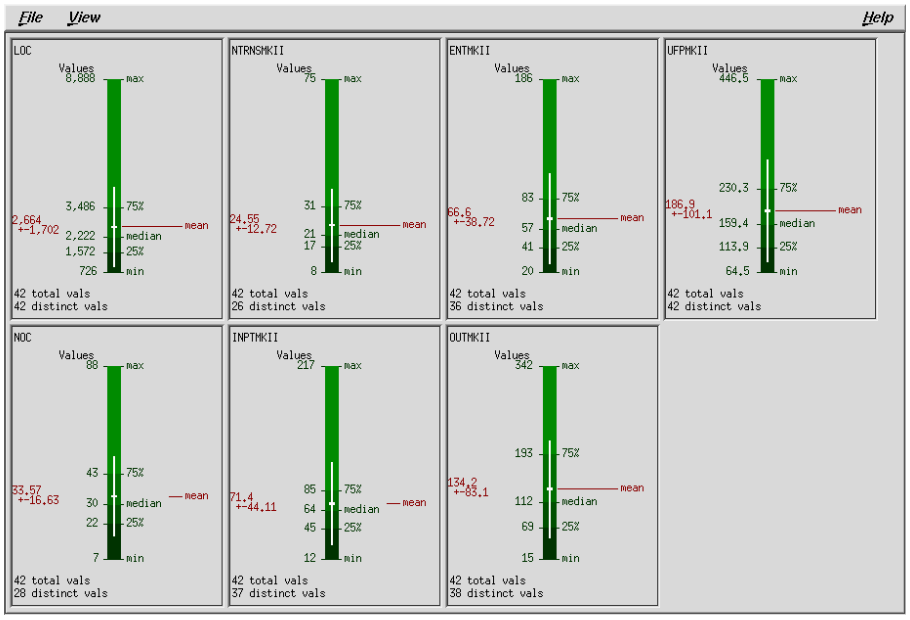


Figure 4. Statistical study of attributes of projects

5. Building software size estimation models

The Mark II prediction process consists of finding the equation that expands the project attributes that act as predictors into the final desired variable (LOC). This is different to the idea proposed by Verner and Tate. Their method sizes every type of component by examining the characteristics of each type and looking for its predictors of size. The system size is obtained by adding the component sizes. In both cases the goal is to estimate the final size in LOC, using as predictors some global measurements of the system obtained in the firs stages of application development.

Our proposal consists of combining component and project attributes in order to improve the estimation results.

Classification was the predictive technique that we applied. It is the process of automatically creating a classification model (classifier) from a training set. The records in the training set must belong to a small set of predefined classes (records in the data for which the label has been supplied). Once a classifier is generated, it can be used to classify or predict class probabilities for unlabeled records (records that are missing the label attribute). A classifier predicts one attribute (*label*) of a set of data, given several other attributes called the *descriptive attributes*.

In this study the attribute being predicted is LOC. At first, we applied several predictive data mining algorithms with project attributes. Best results were obtained when LOC was partitioned in three intervals ($LOC < 3000$, $3000 \leq LOC \leq 5000$ y $LOC > 5000$) in order to create the classes. However, smaller intervals gave poor results. The election of the intervals was made keeping in mind the size of the studied projects and the distribution of values. On the other hand, due to the fact that LOC is one of the independent variables used as input to equations for effort estimation, we have selected size intervals that produce effort estimations (by using COCOMO II method [Boehm, 1995]) with confidence intervals that do not overlap.

After that, we used component and project attributes in the induction of the models. The results showed considerable improvement even for smaller size intervals.

In all cases we applied the cross-validation technique to estimate the classifier error. The data was split into k mutually exclusive subsets (folds) of approximately equal size. The inducer was trained and tested k times; each time it was trained on all the data minus a different fold, then tested on that holdout fold. The estimated error rate is then the average of the errors obtained.

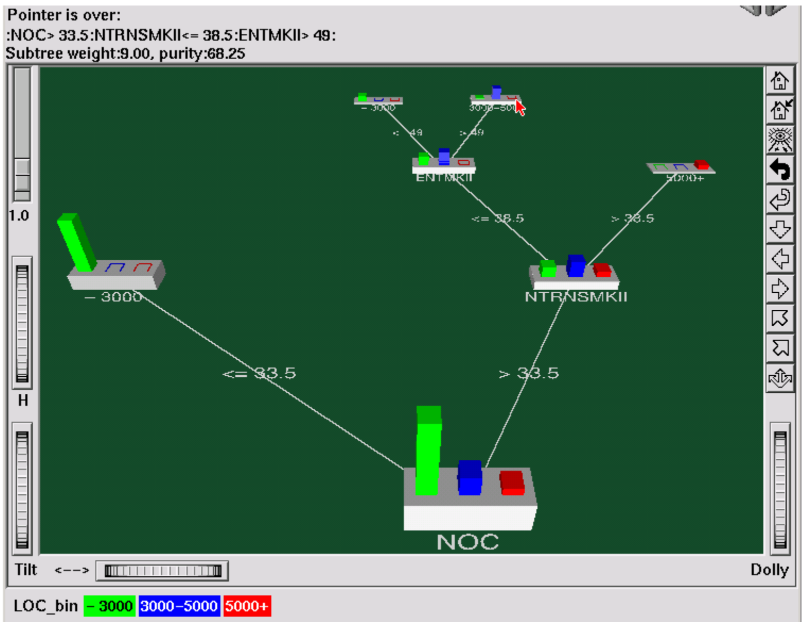
The models were created and visualized by using *Mineset*, a Silicon Graphics tool.

5.1. Decision trees

The decision tree classifier assigns a label to each record, look at the attributes tested at the nodes and the values on the connecting lines. The Decision Tree leaves segment the data into clusters sharing the same classification rule (path that leads to each leaf). By looking at the leaves, it is possible to see clusters that share the same set of properties. Leaf nodes in a Decision Tree specify a class. The probabilities are determined by the weight of records at the leaves.

The splitting criterion used was the change in purity (that is, the entropy) between the parent node and the weighted average of the purities of the child nodes. The weighted average is based on the number of records at each child node. Purity values from 0 to 100 indicating the skewness of the label value distribution at the node. If a node has records from a single class, the purity is 100. If the label values have the same weight, the purity is 0 [Mineset user's guide, 1998].

Figure 5 shows the decision tree obtained with project attributes. NOC is the most important attribute for classification. Values of NOC <= 33.5 originate a leaf with purity 100. The only leaf with purity less than 100 (68.25) is pointed out by the cursor. The estimate classifier error is 9.0 ± 4.47 %. Similar results were obtained with other classifiers.



Pointer is over:
:NOC> 33.5:NTRNSMKII<= 38.5:ENTMKII> 49:
Subtree weight:9.00, purity:68.25

Figure 5. Decision tree for project attributes

In a second experiment, we used the file that contains data from components. We calculated the following attributes for each project:

NOC-MENU: total number of menu components

NOC-INPUT: total number of input components

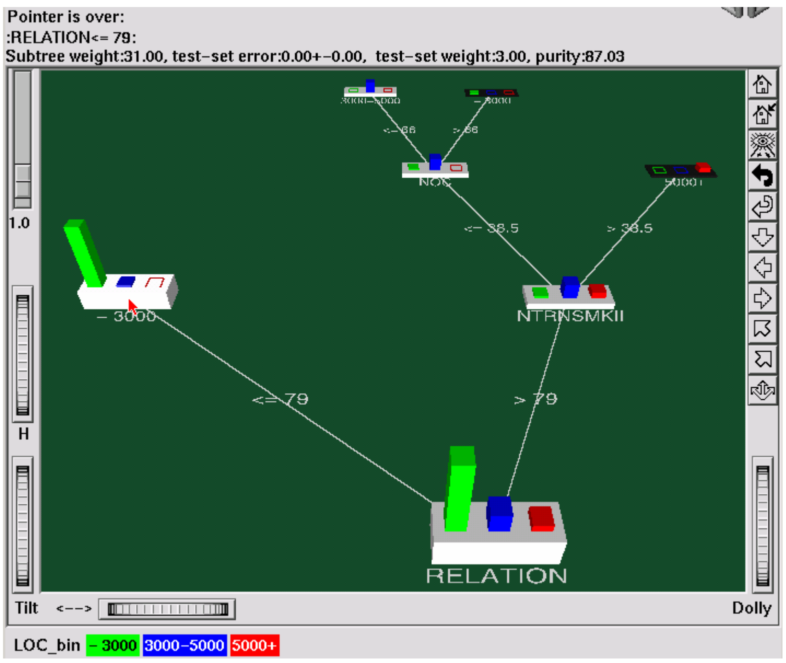
NOC-RQ: total number of report/query components

OPTMENU: total number of menu choices

DATAELEMENT: total number of data elements

RELATION: total number of relations

New models were built joining calculated and project attributes. Figure 6 shows a decision tree obtained for three size intervals. The total number of relations is now the most important attribute for classification. Other influential attributes are NTRSMKII and NOC. The only leaf with purity less than 100 (87.03) is pointed out by the cursor.



Pointer is over:
:RELATION<= 79:
Subtree weight:31.00, test-set error:0.00+-0.00, test-set weight:3.00, purity:87.03

Figure 6. Decision tree for combined attributes (three size intervals)

The decision tree for five uniform size intervals is shown in the figure 7. Attributes used for classification are the same as those in the previous example and DATAELEMENT (total number of data elements). Purity in leaves is 100 except in the one indicated by the cursor (88.10). Average normalized mean squared error varied with the test set size and with the random number used to carry out the partition. In many cases we obtained an error of 0%. Greatest errors were obtained with test sets containing records from large projects (classes with smaller number of records). Decimal numbers in the classification model are not significant, as they come from the construction of the uniform intervals or from the splitting algorithm.

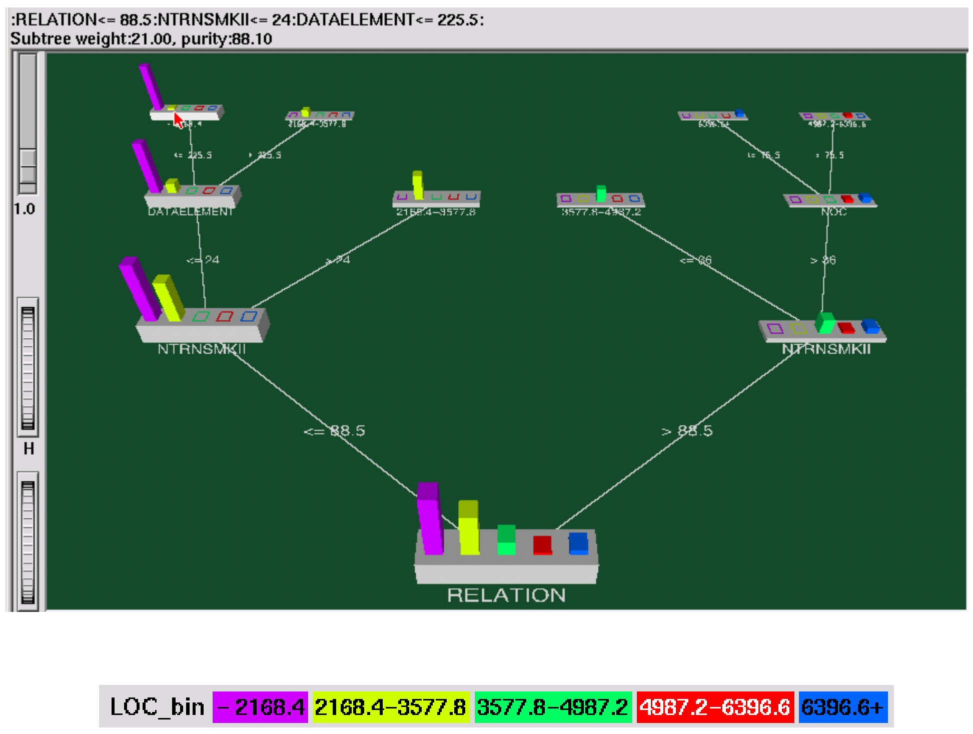


Figure 7. Decision tree for combined attributes (five size intervals)

Rules induced:

IF (RELATION > 88.5 AND NTRNSMKII <= 36) THEN LOC_bin = “Green class”

IF (RELATION > 88.5 AND NTRNSMKII > 36 AND NOC > 75.5) THEN LOC_bin = “Red class”

IF (RELATION > 88.5 AND NTRNSMKII > 36 AND NOC <= 75.5) THEN LOC_bin = “Blue class”

IF (RELATION <= 88.5 AND (NTRNSMKII > 24 OR (NTRNSMKII <= 24 AND DATAELEMENT > 225.5)))

THEN LOC_bin = “Yelow class”

IF (RELATION <= 85.5 AND NTRNSMKII <= 24 AND DATAELEMENT <= 225.5)

THEN (LOC_bin = “Yelow class”(probability = 4.76%) OR LOC_bin = “Purple class” (probability = 95.24%)

5.2. Decision tables

The decision table presents correlations between pairs of attributes. The underlying structure used for classification is a decision tree which splits on a single attribute at each level of the tree hierarchy. Such a structure is called an Oblivious Decision Tree. That attribute is rendered discrete in an identical manner across the whole level. Accuracy is inhibited to a small degree by representing the Decision Tree in this way, but the advantage is a uniform structure in a compact tabular form, which can easily show correlation between pairs of attributes at adjacent levels in the tree [Mineset user’s guide, 1998]. The probability distribution (figure 8) for each rectangular block shows the proportion of records in each class considering only records having that particular combination of values. In the case of three size intervals, there are only two top level blocks with any ambiguity, meaning that more than one class is present (figure 8). The ambiguities disappear in the next levels (figure 9). Thus the complete classification for all records has been achieved. This also occurs when we have five classes (figure 10).

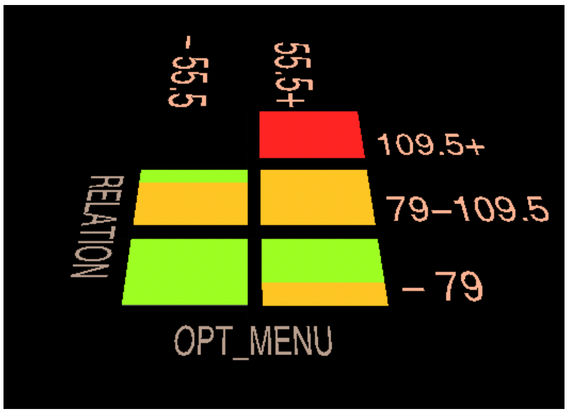


Figure 8.Top level of the decision table (three size intervals)

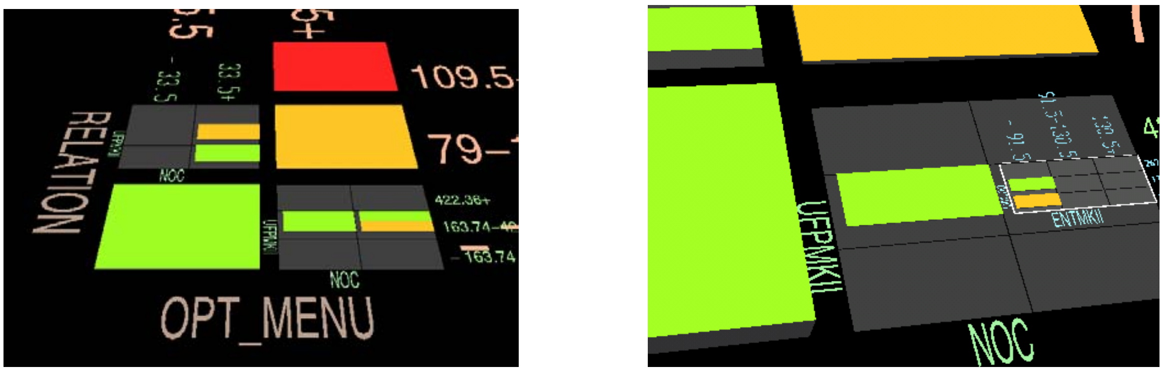
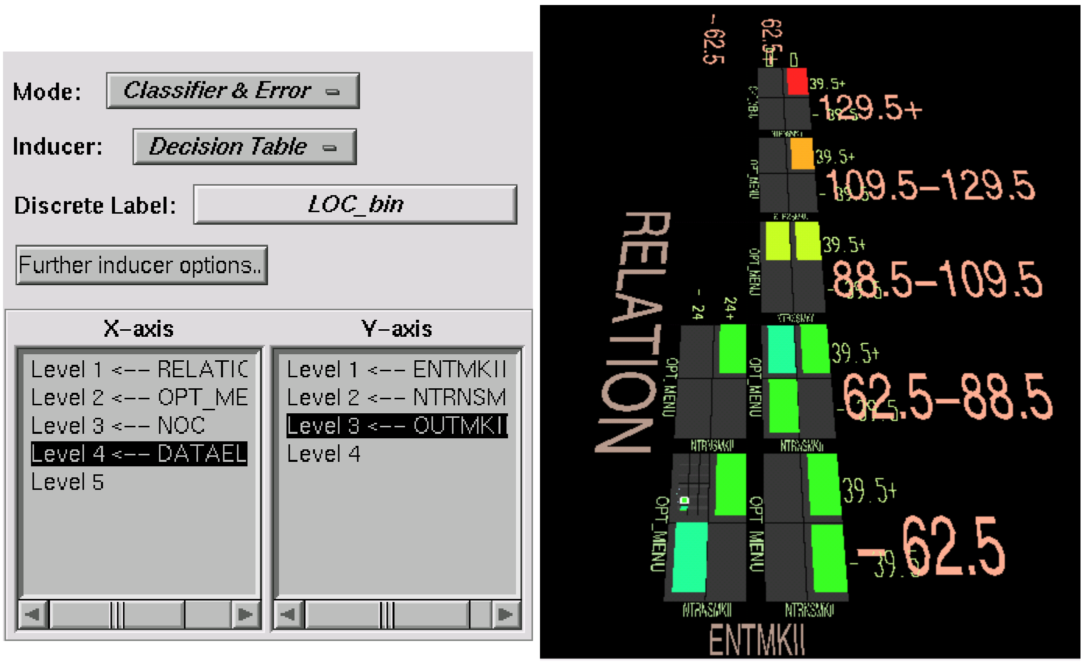


Figure 9. Second and third levels of the decision table (three size intervals)


Mode: Classifier & Error

Inducer: Decision Table

Discrete Label: LOC_bin

Further inducer options..

X–axis	Y–axis
Level 1 <–– RELATIC	Level 1 <–– ENTMKII
Level 2 <–– OPT_ME	Level 2 <–– NTRNSM
Level 3 <–– NOC	Level 3 <–– OUTMKI
Level 4 <–– DATAEL	Level 4
Level 5	

Figure 10. Three levels of the decision table (five size intervals)

5.3. Evidence classifier

The evidence inducer, sometimes called Naive-Bayes (or Simple Bayes), builds a model that assumes the probabilities of each attribute value are independent given the class. It can help to understand the importance of specific attribute values for classification. Also, it can be used to gain insight into how classification is done.

Figures 11 and 12 show evidence classifiers built for our examples. The left side shows rows of cake charts, one for each attribute. For every value of an attribute in the data, there is one chart matching it in the row for the attribute. Given a record with an attribute value corresponding to a chart, the chart represents how much evidence the classifier "adds" to each possible label value. The right window of the screen shows the distribution of the classes in the selected block. When we have three classes, a single attribute as RELATION or NTRSMKII can be good to classify the records. However, the classification in five classes is more difficult, so a single attribute is not sufficient. In both cases RELATION is the best attribute for classification.

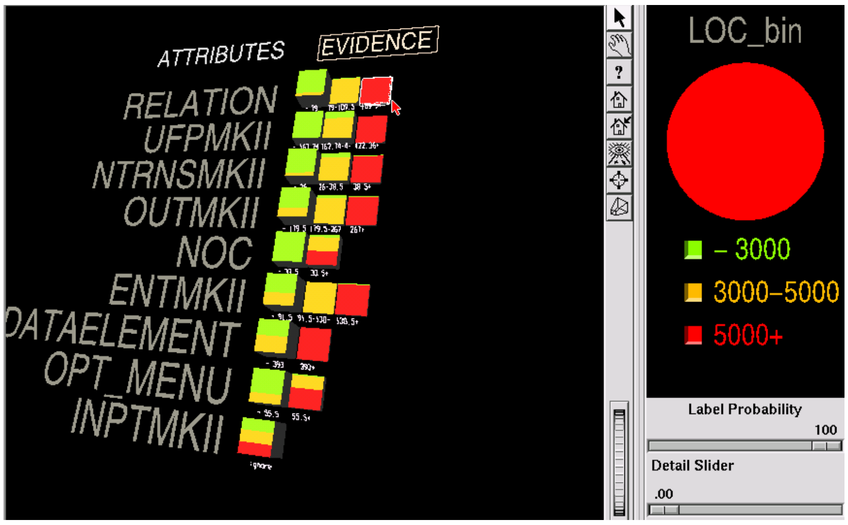


Figure 11. Evidence inducer (three size intervals)

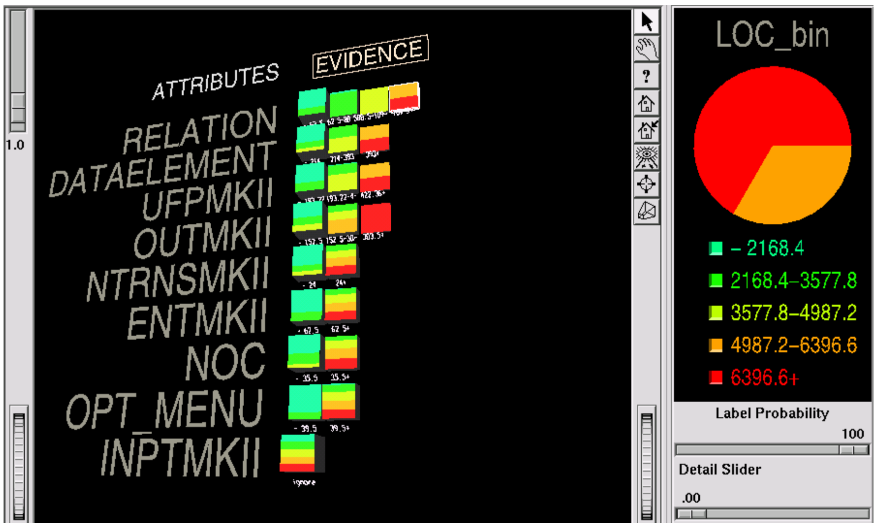


Figure 12. Evidence inducer (five size intervals)

6. Visualization techniques

Visualization techniques are among the most powerful devices for identifying patterns hidden in data. Visualizations are particularly useful for noticing phenomena that hold for a relative small subset of the data [Cabena et al.1998]. There are many ways of visualizing data. Three-dimensional scatterplot and splat graphs can be used for multidimensional data. Color, opacity and other features can be used to represent variables. In figures 13 and 14 we can observe the influence of several attributes in the software size. A correlation between MKII function points and LOC can be found. This indicates that the function points of the method Mark II are good size predictors.

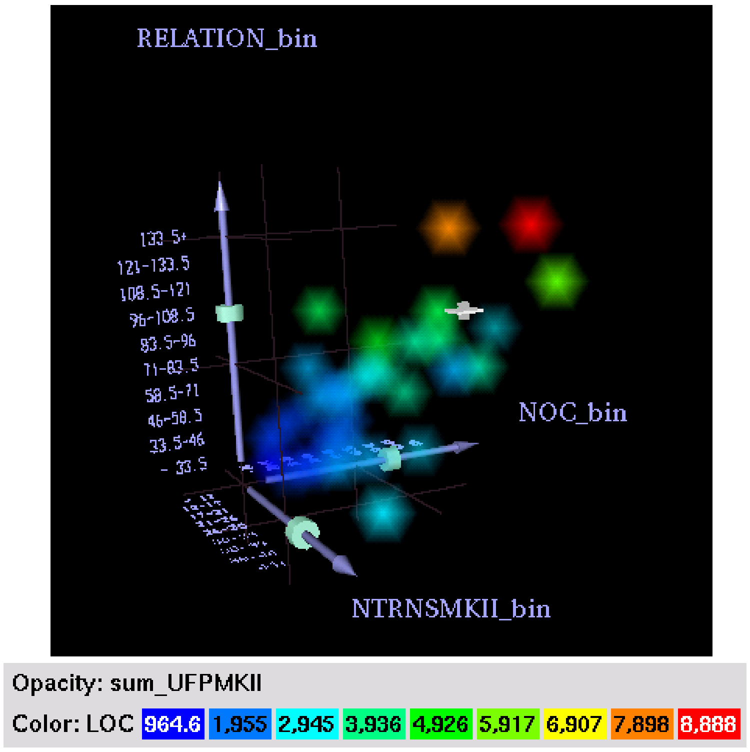


Figure 13. Splat graph for projects

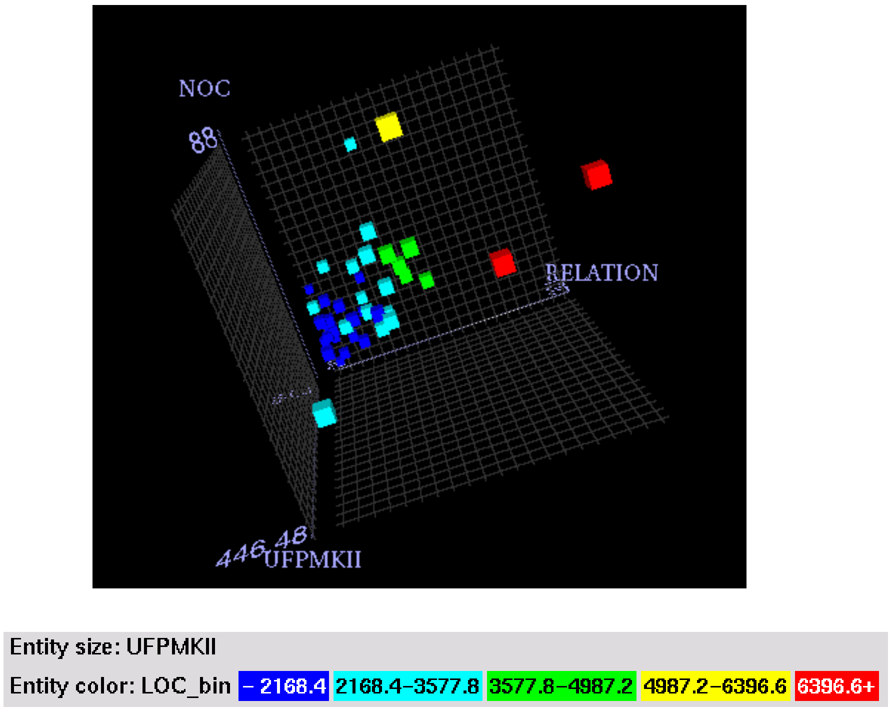


Figure 14. Scatterplot for projects

Graphics showing relations between attributes of each component type and component size are given below (Figures 15 and 16).

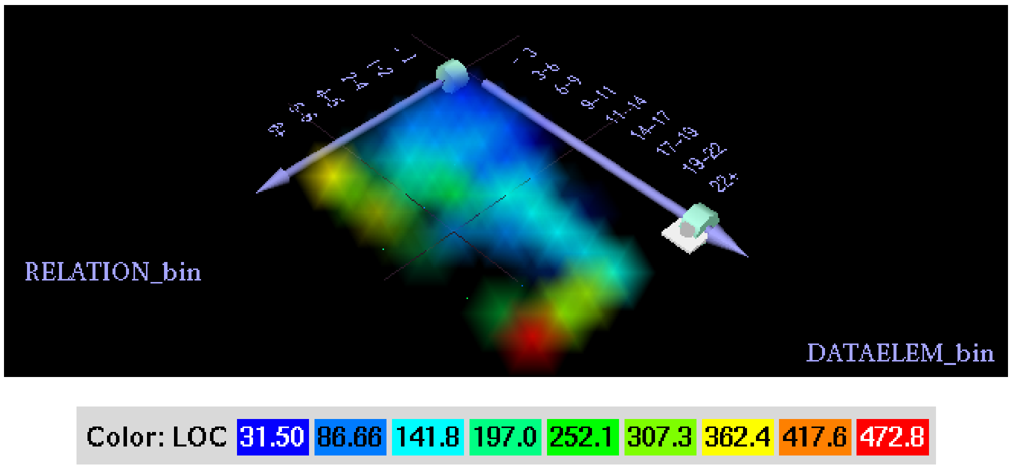


Figure 15. Splat graph for input components

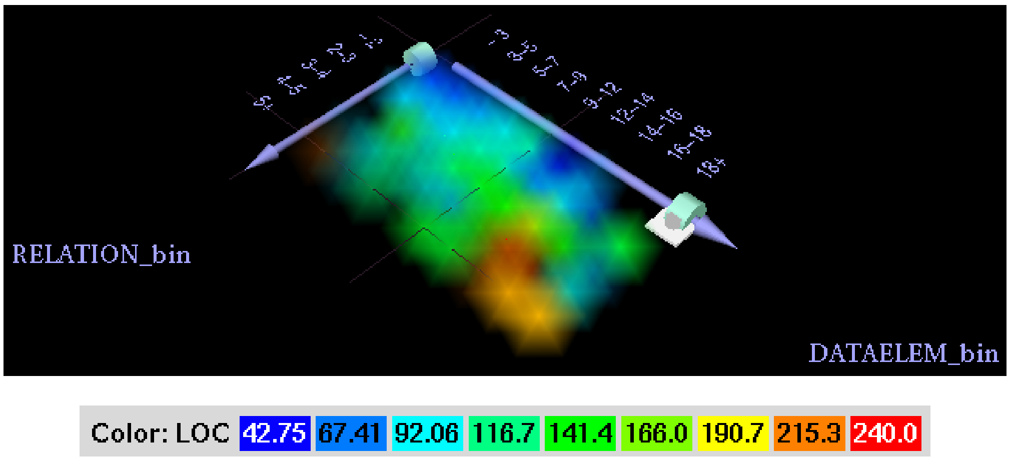


Figure 16. Splat graph for report/query components

7. Conclusions

In the paper several data mining methods are described and used to create and validate models for software size estimation. We have examined results of these techniques by using attributes obtained in early stages of the life cycle. Estimations models are induced automatically by means of the machine learning algorithms from the available attributes. A significant advantage with respect to classical methods is the discovering of the best attributes in discriminating the classes without need of formulating a hypothesis. The models contain the attributes more influencing in the estimation of the target variable, lines of code in the case of the present study. The experiments carried out have served to reveal which attributes or attribute combination are better to predict the final product size.

Another advantage of data mining models is that they are adaptable and non-parametric; therefore, they can be tailored to the data at a particular project or application domain. In addition, the techniques presented in the article are especially useful because they make explicit the relevant attributes for prediction.

References

A.J. Albrecht, Measuring application development, Proc. IBM Applications Development Joint SHARE/GUIDE Symposium, Monterey, CA, 83-92, 1979.

B.W. Boehm; B. Clark; E. Horowitz et al., Cost models for future life cycle processes: COCOMO 2.0, Annals Software Engineering 1(1): 1-24, 1995.

B. Brykczynskki, A survey of software inspection checklist, ACM Software Engineering Notes, 24(1): 82-89, 1999.

P. Cabena; P. Hadjinian; R. Stadler; J. Verhees and A. Zanasi, Discovering data mining. from concept to implementation, Prentice Hall, 1998.

CMMI-SW, Capability Maturity Model Integration (CMMI^{SM}) for Software Engineering, V1.1, Carnegie Mellon Software Engineering Institute, Marzo 2002.

G. Costagliola, F. Ferrucci, G. Tortora y G. Vitiello, Class Point: An Approach for the Size Estimation of Object-Oriented Systems, IEEE Transaction of Software Engineering, 31 (1): 52-74, 2005.

T. DeMarco, Controlling software projects, Yourdon Press, 1982.

J.J. Dolado, A validation of the component-based method for software size estimation, IEEE Transactions on Software Engineering 26(10): 1006-1021, 2000.

B. Hall, G. Orr and T.E. Reeves, A Technique for Function Block Counting, The Journal of System and Software, 57: 217-220, 2001.
ISO/IEC 9000:2000 "Quality Management Systems-Fundamentals and vocabulary", 2000.
ISO/IEC 9001:2000, "Quality Management Systems-Requirements", 2000.
ISO/IEC 9004:2000, "Quality Management Systems-Guidelines for performance improvements", 2000.
ISO/IEC 90003:2000, "Software Engineering-Guidelines for the Application of ISO 9001:2000 to Computer Software, 2004.
ISO/IEC 9126-3:2003, "Software engineering-Product quality-Part 3: Internal metrics", 2003.
ISO/IEC 9126-4:2004, "Software engineering-Product quality-Part 4: Quality in use metrics", 2004.
ISO/IEC 15504, "Information technology - Software process assessment-Part 1", 1998.
ISO/IEC 15504-2:2003, "Information technology-Process assessment-Part 2: Performing an assessment", 2003.
ISO/IEC 15504-3:2004, "Information technology-Process assessment-Part 3: Guidance on performing an assessment", 2004.
ISO/IEC 15504-5:1999, "Information technology-Software Process Assessment-Part 5: An assessment model and indicator guidance", 1999.
ISO/IEC 15504-7:1998, "Information technology-Software process assessment-Part 7: Guide for use in process improvement", 1998.
ISO/IEC 15504-8:1998, "Information technology-Software process assessment-Part 8: Guide for use in determining supplier process capability", 1998.
ISO/IEC 15504-9:1998, "Information technology-Software process assessment-Part 9: Vocabulary", 1998.
Mineset user's guide, v. 007-3214-004, 5/98, Silicon Graphics, 1998.
M.N Moreno, L.A. Miguel,, F.J. García, and M.J. Polo, Data Mining Approaches for Early Software Size Estimation, Proc. 3rd ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'02), 361-368, Madrid, Spain, 2002.
M.N Moreno, L.A. Miguel,, F.J. García, and M.J. Polo, Building Knowledge Discovery-Driven Models for Decision Support in Project Management, Decision Support Systems, 38, pp. 305-317, 2004.
M.Paulk, B. Curtis, M. Chrissis and C. Weber, Capability Maturity Model for Software: Version 1.1. Technical Report SEI-93-TR-24, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1993.
K. Srinivasan, and D. Fisher, Machine Learning Approaches to Estimating Software Development Effort, IEEE Transactions on Software Engineering, 21(2): 126-137, 1995.
C.R. Symons, Software Sizing and Estimating MKII FPA. John Wiley and Sons, 1991.
J. Verner and G. Tate, A software size model, IEEE Transaction of Software Engineering, 18 (4): 265-278, 1992.