

Productivity Factors in Software Development for PC Platform

Abbas Heiat, Nafisseh Heiat

Abstract

Identifying the most relevant factors influencing project performance is essential for implementing business strategies by selecting and adjusting proper improvement activities. The two major classification algorithms CRT and ANN that were recommended by the Auto Classifier tool in SPSS Modeler used for determining the most important variables (attributes) of software development in PC environment. While the accuracy of classification of productive versus non-productive cases are relatively close (72% vs 69%), their ranking of important variables are different. CRT ranks the Programming Language as the most important variable and Function Points as the least important. On the other hand, ANN ranks the Function Points as the most important followed by team size and Programming Language.

I. Introduction

Identifying the most relevant factors influencing project performance is essential for implementing business strategies by selecting and adjusting proper improvement activities. There is, however, a large number of potential influencing factors. There is, however, a large number of potential influencing factors. This paper proposes data mining approach for identifying the most relevant factors influencing software development productivity for PC platform. The method first determines the most efficient algorithms for classifying and establishing contributing factors. The evaluation of the algorithms indicates a different set of factors are relevant. Moreover, application of auto classification significantly improves the decision of choosing an algorithm in terms of accuracy and precision. Many software organizations are still proposing unrealistic software costs, work within tight schedules, and finish their projects behind schedule and budget, or do not complete them at all [12]. This illustrates that reliable methods for managing software development effort and productivity are a key issue in software organizations.

One essential aspect when managing development effort and productivity is the large number of associated and unknown influencing factors or productivity factors [23]. Identifying the right productivity factors increases the effectiveness of productivity improvement strategies by concentrating management activities directly on those development processes that have the greatest impact on productivity. On the other hand, focusing measurement activities on a limited number of the most relevant factors reduces the cost of quantitative project management [6].

Software development productivity is an

important project management concern. One study reports that a 20% improvement in software productivity will be worth \$45 billion in the U.S and \$90 billion worldwide [1]. As a result, a number of empirical studies of software productivity have appeared in the literature over the past three decades. Scacchi has published a report that examines empirical investigations in relation to software development attributes, tools, techniques, or some combination of these that have a significant impact on productivity of software production [2]. These studies focus on the development of large scale software development. Twelve major software productivity measurement studies are reviewed including those at IBM (Albrecht [3], [4]), TRW (Boehm [1], [5], [6], [7]), NASA (Bailey and Basili [8]), ITT (Vosburg et al [9]), and international projects (Lawrence [10], Cusumano and Kemerer [11]). In addition, Scacchi examines a number of other theoretical and empirical studies of programmer productivity, cost-benefit analysis, and estimation of software cost (Thadhani [12], Lambert [13], Cerveny and Joseph [14]).

Based on his survey, Scacchi identifies a number of software productivity attributes:

1. Computing resources and easy-to-access to support system specialists
2. Contemporary software engineering tools and techniques
3. System development aids for coordinating software projects
4. Programming languages
5. Software project Complexity, indicated by size of source code delivered,
6. functional coupling, and functional cohesion
7. Reuse software that supports the information processing tasks required by the application
8. Stable system requirements and specifications

9. Small, well-organized project teams
10. Experienced software development staff

However, Scacchi believes that it is not always possible or desirable to improve software productivity by cultivating the entire project characteristics listed above.

Discovering factors that influence software development productivity and relationships among them is difficult and complicated. Our Objective in this paper is to determine factors influencing productivity of the software development in PC platform environment by using appropriate data mining algorithms.

II. Productivity Measure of Software Development

In general, productivity is understood as a ratio of outputs produced to inputs used. However, researchers may use different outputs and inputs for measuring productivity. IEEE Standard 1045 calculates productivity in terms of effort as an input and lines of code or function points as output [16]. The two most common methods for measuring complexity or size of a software development project are Function Points and Lines of Code.

The main limitation of the LOC model is that it depends on the accuracy of an early estimate of lines of code. This estimate is usually based on the past experience of the systems analyst. Certainly, most organizations would find it difficult, if not impossible, to locate experienced analysts who could come up with an accurate estimate of the system size using a LOC model [17]. A third problem with LOC model is that it does not take into account the resources available to the systems development team. These include among other things the types of language used in coding, software tools, the skills and experiences of the team itself [18].












An alternative method for estimating systems development effort was developed by Albrecht [3]. Albrecht introduced the concept of Function Points (FP) to measure the functional requirements of a proposed system. In FP modeling the size of the system is determined by first identifying the type of each required function in terms of inputs, outputs, inquiries, internal files, and external interface files. To calculate the value of function points for each category, the number of functions in each category is multiplied by the appropriate complexity weight. The total systems effort is then calculated by multiplying the sum of function points for all categories by the Technical Complexity Factor (TCF). The TCF is determined by assigning values to 14 influencing project factors and totaling them. Readers unfamiliar with the FP model are referred to Albrecht and Gaffney [4]. Albrecht argued that FP model makes intuitive sense to users and it would be easier for project managers to estimate the required

systems effort based on either the user requirements specification or logical design specification [3]. Another advantage of the FP model is that it does not depend on a particular language. Therefore, project managers using the FP model would avoid the difficulties involved in adjusting the LOC counts for information systems developed in different languages. In this paper I have used the following equation for calculation of productivity: Productivity = Effort/Function Points

III. Methodology

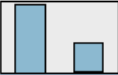



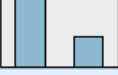

Data Mining may be defined as the process of finding potentially useful patterns of information and relationships in data. As the quantity of clinical data has accumulated, domain experts using manual analysis have not kept pace and have lost the ability to become familiar with the data in each case as the number of cases increases. Improved data and information handling capabilities have contributed to the rapid development of new opportunities for knowledge discovery. Interdisciplinary research on knowledge discovery in databases has emerged in this decade. Data mining, as automated pattern recognition, is a set of methods applied to knowledge discovery that attempts to uncover patterns that are difficult to detect with traditional statistical methods. Patterns are evaluated for how well they hold on unseen cases. Databases, data warehouses, and data repositories are becoming ubiquitous, but the knowledge and skills required to capitalize on these collections of data are not yet widespread. In this research As a First step we used Auto-Classification tool in SPSS Modeler which applies 11 different algorithms shown in Figure 1. The most efficient algorithms with highest accuracy rates are displayed based on current data set used for analysis.

Figure 1. Auto-Classification's Algorithms

Model type	
	C5
	Logistic regression
	Decision List
	Bayesian Network
	Discriminant
	KNN Algorithm
	SVM
	C&R Tree
	Quest
	CHAID
	Neural Net

The following is a brief description of the algorithms suggested and displayed by Auto-Classification as the most accurate models as shown in Figure 2.

Figure 2. The Most Efficient Algorithms

Use?	Graph	Model	Build Time (mins)	Max Profit	Max Profit Occurs in (%)	Lift{Top 30...	Overall Accuracy (%)	No. Fields Used	Area Under
<input checked="" type="checkbox"/>		 C&R Tree 1	< 1	5	2	1.271	72.059	7	0.603
<input checked="" type="checkbox"/>		 Neural Net 1	< 1	0	1	1.02	69.118	7	0.602
<input checked="" type="checkbox"/>		 C5 1	< 1	0	1	1	70.588	7	0.5

Decision Trees- Decision trees and rule induction are two most commonly used approaches to discovering logical patterns within medical data sets. Decision trees may be viewed as a simplistic approach to rule discovery because of the process used to discover patterns within data sets.

Decision tree is built through a process known as binary recursive partitioning. This is an iterative process of splitting the data into partitions, and then splitting it up further on each of the branches. Initially, you start with a training set in which the classification label (say, "productive" or "non-productive") is known (pre-classified) for each record. All of the records in the training set are together in one big box. The algorithm then systematically tries breaking up the records into two parts, examining one variable at a time and splitting the records on the basis of a dividing line in that variable (say, $FP > 30$ or $FP \leq 30$). The object is to attain as homogeneous set of labels (say, "productive" or "non-productive") as possible in each partition. This splitting or partitioning is then applied to each of the new partitions. The process continues until no more useful splits can be found. The heart of the algorithm is the rule that determines the initial split rule [14].

The process starts with a training set consisting of pre-classified records. Pre-classified means that the target field, or dependent variable, has a known class or label: "productive" or "non-productive". The goal is to build a tree that distinguishes among the classes. For simplicity, assume that there are only two target classes and that each split is binary partitioning. The splitting criterion easily generalizes to multiple classes, and any multi-way partitioning can be achieved through repeated binary splits. To choose the best splitter at a node, the algorithm considers each input field in turn. In essence, each field is sorted. Then, every possible split is tried and considered, and the best split is the one which

produces the largest decrease in diversity of the classification label within each partition. This is repeated for all fields, and the winner is chosen as the best splitter for that node. The process is continued at the next node and, in this manner, a full tree is generated.

Artificial Neural Networks (ANN) - Artificial neural networks are defined as information processing systems inspired by the structure or architecture of the brain (Caudill & Butler, 1990). They are constructed from interconnecting processing elements, which are analogous to neurons. The two main techniques employed by neural networks are known as supervised learning and unsupervised learning. In unsupervised learning, the neural network requires no initial information regarding the correct classification of the data it is presented with. The neural network employing unsupervised learning is able to analyze a multi-dimensional data set in order to discover the natural clusters and sub-clusters that exist within that data. Neural networks using this technique are able to identify their own classification schemes based upon the structure of the data provided, thus reducing its dimensionality. Unsupervised pattern recognition is therefore sometimes called cluster analysis [3], [16], and [17].

Supervised learning is essentially a two stage process; firstly training the neural network to recognize different classes of data by exposing it to a series of examples, and secondly, testing how well it has learned from these examples by supplying it with a previously unseen set of data. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. It provides projections given new situations of interest and answers "what if" questions.

There are disadvantages in using ANN. No explanation of the results is given i.e. difficult for the

user to interpret the results. They are slow to train due to their iterative nature. Empirical studies have shown that if the data provided does not contain useful information within the context of the focus of the investigation, then the use of neural networks cannot generate such information any more than traditional analysis techniques can. However, it may well be the case that the use of neural networks for data mining allows this conclusion to be reached more quickly than might ordinarily be the case.

In the last two decades, Artificial Neural Networks have been used for predictions in diverse applications. In recent years, a number of studies have used neural networks in various stages of software development. Hakkarainen et al, estimated software size by training an ANN. They used structured specification descriptions as input and Demarco Function Bang, Albrecht's Function Points and Symon's mark II Function Points size metrics as output. The results of their study indicated that ANN could be used successfully to estimate software size [20]. Srinivasan and Fisher compared two approaches 1) a back propagation neural network and 2) Regression Trees, using Boehm's historical database. Their experiments indicated that neural network and regression trees are competitive with model-based approaches [21]. Finnie and Wittig applied artificial neural networks (ANN) and case-based reasoning (CBR) to estimate software development effort [22]. They used a data set from the Australian Software Metrics Association. ANN was able to estimate software development effort within 25% of the actual effort in more than 75% of the cases, and with a MAPE of less than 25%. Carolyn Mair et al, used 67 software projects derived from a Canadian software house to evaluate prediction performances of regression, Rule Induction (RI), CBR and ANN techniques [23]. The results of the study showed considerable variation between the 4 models. MAPE for RI ranged from 86% to 140%. MAPE for regression ranged from 38% to 100%. MAPE for CBR ranged from 43% to 80% and for ANN ranged from 21% to 66%. MAPE results suggest that ANN seem to be the most accurate and RI is the least accurate technique [23]. Shukla conducted a large number of simulation experiments using genetically trained neural networks. He used a merged database comprising 63 projects, and Kemerer database comprising 15 projects. The results indicated a significant estimation improvement over Quick Propagation Network and Regression Trees approaches. Shukla concluded that there is still a need to apply neural networks to diverse projects with wide range of attributes because it is "unclear which techniques are most valuable for a given problem. ..., experimental comparison using rigorous evaluation methods is necessary" [24].

The Multilayer Perceptron (MLP) is one of the most widely implemented neural network topologies. In terms of mapping abilities, the MLP is believed to be capable of approximating arbitrary functions. This has been important in the study of nonlinear dynamics, and other function mapping problems. MLPs are normally trained with the back propagation algorithm. Two important characteristics of the Multilayer Perceptron are:

It's smooth nonlinear Processing Elements (PEs). The logistic function and the hyperbolic tangent are the most widely used. Their massive interconnectivity i.e. any element of a given layer feeds all the elements of the next layer.

1. The Multilayer Perceptron is trained with error correction learning, which means that the desired response for the system must be known. Back propagation computes the sensitivity of a cost function with respect to each weight in the network, and updates each weight proportional to the sensitivity.

IV. DATA

The data used in this research project was collected by The International Software Benchmarking Standards Group (ISBSG). The group gathered information from 1238 software projects from around the world. Projects cover a broad cross-section of the software industry. In general, they have a business focus. The projects come from 20 different countries. Major contributors are the United States (27%), Australia (25%), Canada (11%), United Kingdom (10%), Netherlands (7%), and France (7%). Major organization types are insurance (19%), government (12%), banking (12%), business services (10%), manufacturing (10%), communications (7%), and utilities (6%).

Projects types include enhancement projects (50%), new developments (46%), and 4% are re-developments. Application types consist of Management Information Systems (38%), transaction/production systems (36%), and Office Information Systems (5%). Nearly 3% are real-time systems.

Over 70 programming languages are represented. 3GLs represent 57% of projects, 4GLs 37%, and application generators 6%. Major languages are COBOL (18%), C/C++ (10%), Visual Basic (8%), Cobol II (8%), SQL (8%), Natural (7%), Oracle (7%), PL/I (6%), Access (3%), and Telon (3%). Platform for projects include mainframe projects (54%), midrange (24%), and microcomputers (22%).

Sixty-two (62%) of projects use a standard methodology that was developed in-house, 21% use a purchased methodology. Only 12% do not follow a methodology. The use of CASE tools ranges from 21% of projects using upper CASE, down to 10% for

integrated CASE tools. CASE tools of some type are used in 51% of projects. Traditional system modelling techniques (data modelling, process modelling, event modelling, business area modelling) are used in 66% of projects. They are the only techniques listed in 27% of projects; 39% use a combination of traditional modelling and other techniques. The most common single technique is data modelling, used in 59% of projects. RAD/JAD techniques are used in 28% of projects. Object oriented techniques are used in 14% of projects. Prototyping is used in 29% of projects.

Data in the ISBSG database had to be cleaned and pre-processed in order to get, relevant and complete data for analysis. Records with missing value of attributes were excluded and the character values of text attributes or variables were transformed to numeric values. Function points

count, total work effort in hours, team size, development platform (mainframe, mid-size, PC), language type (3GL, 4GL, Application Generator etc.), whether a software development methodology was used, programming language and development type (new, enhancement, etc.) attributes were considered for analysis. Productivity attribute was calculated by dividing total work effort in hours by count of function points. Once the data was pre-processed, 468 usable projects were available for analysis.

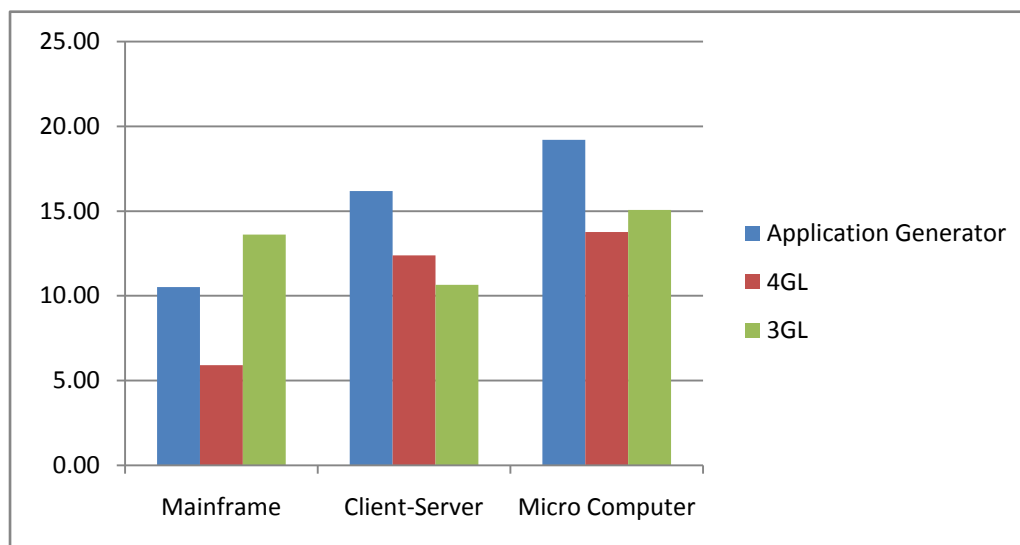
V. Data Analysis

In a previous study we created a pivot table that displayed the productivity of software development by platform and programming type. Table 1 and Figure 3 shows the results.

Table 1. Pivot table for Average Productivity, Development Platform, Language Type, and Methodology

Methodology	(All)			
Average of Productivity Development Platform	Language Type Application Generator	4GL	3GL	Grand Total
Mainframe	10.51	5.91	13.62	8.04
Client-Server	16.18	12.39	10.65	14.19
Micro Computer	19.19	13.77	15.07	17.78
Grand Total	16.72	10.28	14.71	14.24

Figure 3. Average Productivity, Development Platform, Language Type, and Methodology



As the displayed information indicates, in terms of hours spent per function point, the Main Frame platform is most productive and the PC platform the least productive among the three platforms. This result is in conflict with findings of previous research. Therefore, the motivation for further and

more detailed study of PC platform and determining which development factors are contributing or not contributing to productivity in PC development environment.

Data in the ISBSG database had to be cleaned and pre-processed in order to get, relevant and

complete data for analysis. Records with missing value of attributes were excluded and the character values of text attributes or variables were transformed to numeric values. Function points count, team size, development platform (mainframe, mid-size, PC), languagetype(3GL,4GL,Application Generatoretc.), whether a software development methodology was used, programming language and development type (new, enhancement, etc.) attributes were considered for analysis. Productivity attribute was calculated

by dividing total work effort in hours by count of function points. Once the data was pre-processed, 468 usable projects were available for analysis.

An exploratory study was conducted using regression analysis on numeric productivity values. As the following tables indicate only the programming language statistically is significant (t of -2.717) and in an inverse relationship with productivity. Adjusted R² is 0.047 which is really low and means that linear regression cannot explain variations in productivity.

Table 2. Regression Analysis Model

Coefficients(a)						
Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	18.184	6.684		2.720	.007
	FP	-.002	.001	-.085	-1.227	.221
	TeamSize	-.046	.253	-.012	-.180	.858
	LanguageType	-1.676	1.712	-.076	-.979	.329
	Methodology	6.368	4.536	.099	1.404	.162
	DevelopmentType	.750	1.691	.031	.444	.658
	ApplicationType	.347	.798	.030	.434	.664
	Programming	-.657	.242	-.208	-2.717	.007

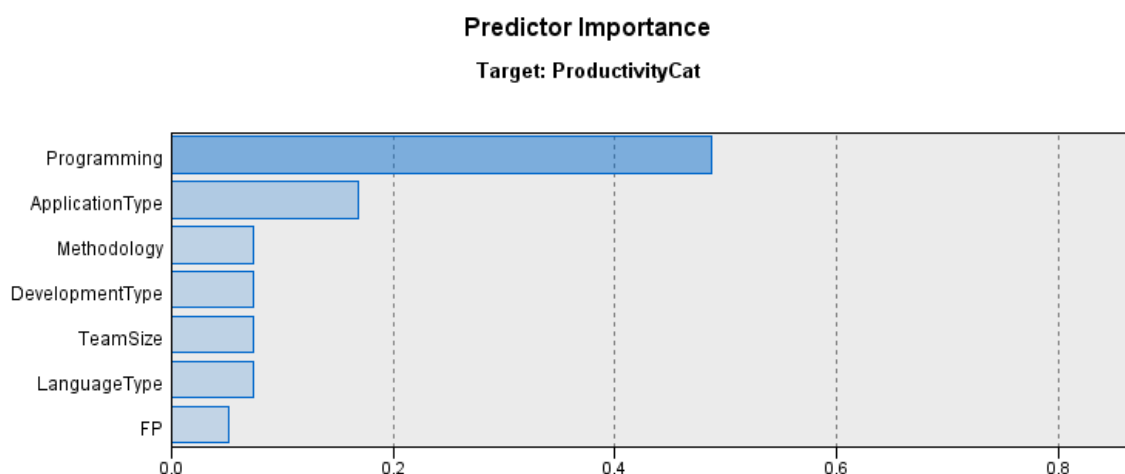
a. Dependent Variable: Productivity

VI. Decision Tree Analysis

Before using this algorithm data has been balanced and partitioned into training and testing samples. You can use Balance nodes to correct imbalances in dataset. In our dataset balancing is used in order to make the number of productive and non-productive cases close to equal. SBGI Dataset is partitioned into 70% for training and 30% for testing the models. Figure 4 shows that programming

language is the most important variable and function point is the least important. While the importance of programming language is theoretically make sense and confirms the previous findings, the lack of importance for function points is surprising. The function point represents the complexity and size of a software project and you may expect that it should have a great influence on productivity of software development.

Figure 4. Importance of variables based on CRT Analysis



The following diagrams display the confusion matrix, gain chart and decision tree rules created by the model. Both the confusion matrix and the gain chart indicate CRT as a good model with 72% accuracy.

Figure 5. CRT Confusion Matrix

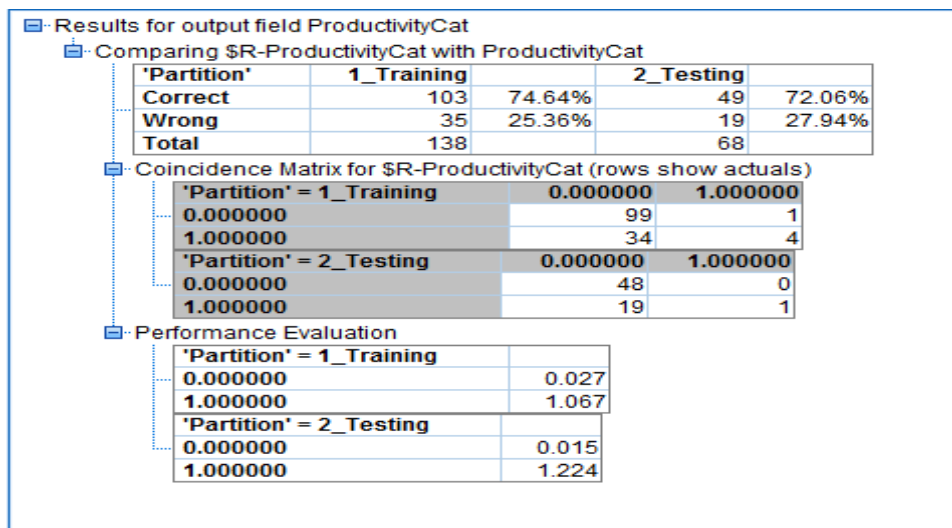


Figure 6. CRT Gain Chart

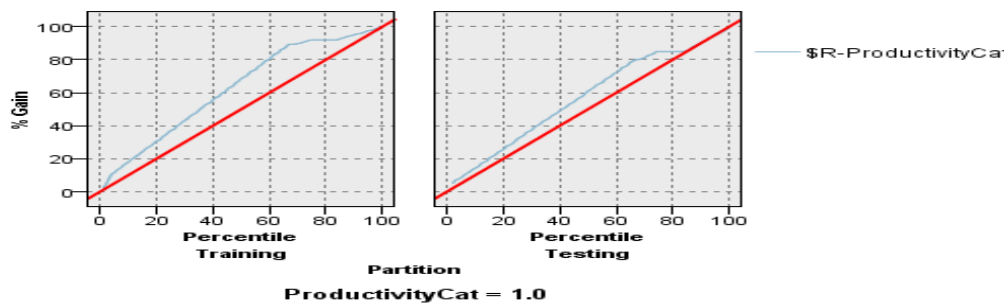
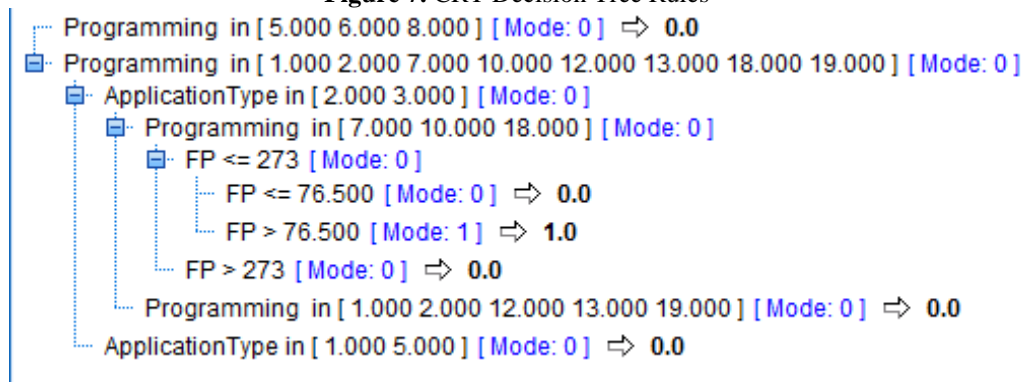


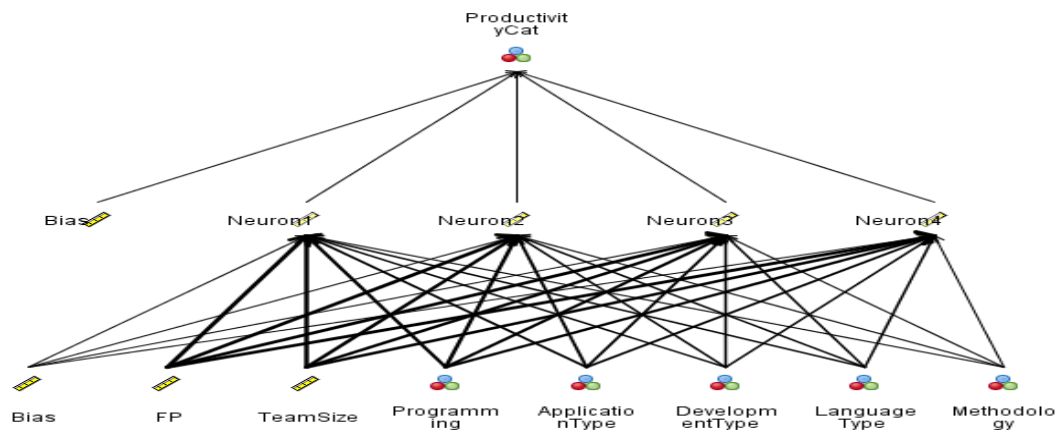
Figure 7. CRT Decision Tree Rules



VII. Artificial Neural Network Analysis

The same balancing and partition options used for ANN analysis. A Multilayer Perceptron network was used for this analysis. Figure 8 shows the basic structure of this network.

Figure 8. The Network Model



The ANN model resulted in a different ranking of important variables. Function Points is the most important followed by Team Size and Programming Language. This result seems to be similar to the results of previous results. Confusion matrix shows lower accuracy for ANN model (69%) than accuracy for CRT model (72%). Figure 11 represents the gain chart for ANN model.

Figure 9.Importance of variables based on ANN Analysis

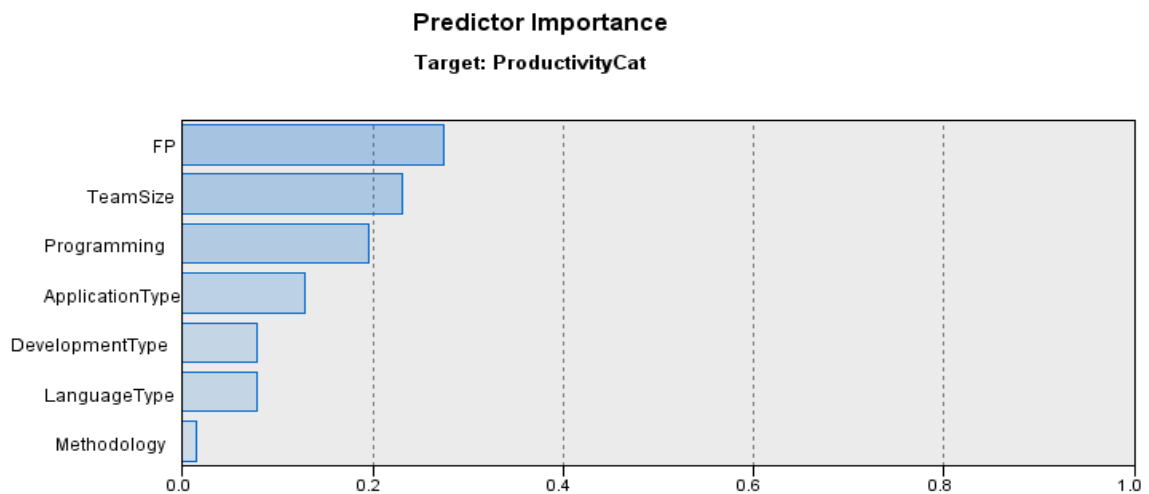


Figure 10. ANN Confusion Matrix

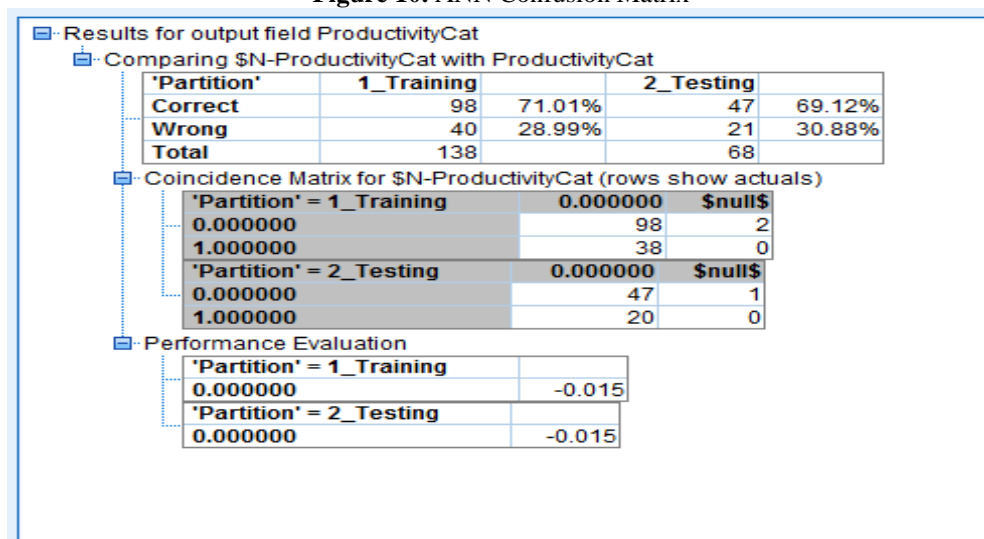
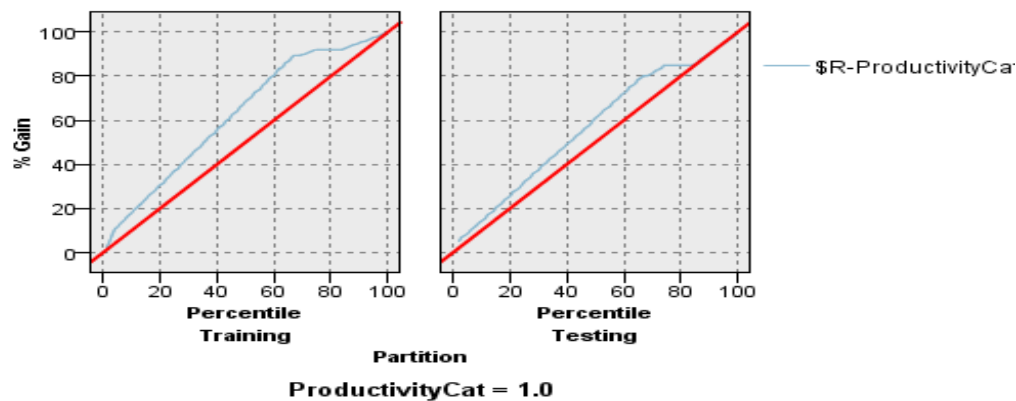


Figure 11.ANN Gain Chart



VIII. Conclusion

The two major classification algorithms CRT and ANN that were recommended by the Auto Classifier tool in SPSS Modeler used for determining the most important variables (attributes) of software development in PC environment. While the accuracy of classification of productive versus non-productive cases are relatively close (72% vs 69%), their ranking of important variables are different. CRT ranks the Programming Language as the most important variable and Function Points as the least important. On the other hand, ANN ranks the Function Points as the most important followed by team size and Programming Language.

Let us consider the results of CRT which is more accurate in terms of classification. In CRT model methodology, application type, language type, team size, and specially function points which represents the size and complexity of the software development are not indicated as important variable. Lack of or poor use methodology, which is also listed as the least important variable in ANN model, has clearly an effect on productivity of software development and may explain the lower productivity in PC platform application. However, the low importance of team size and function points is puzzling.

There is a need for using a larger sample size and may be from different repositories to validate or reject the results of this study. Conducting research on Mainframe platform and comparing the results with the results of the PC platform would also clarify further the productivity issue.

REFERENCES

[1] Boehm, B.W., "Improving Software Productivity", *Computer*, 20(8), 43-58, 1987.
 [2] Scacchi, W., "Understanding Software Productivity", *Advances in Software Engineering and Knowledge Engineering*, Volume 4, pp. 37-70, 1995.
 [3] Albrecht, A., "Measuring Application Development

Productivity", Proc. Joint SHARE/GUIDE/IBM Application Development Symposium (October, 1979), 83-92.

[4] Albrecht, A. and J. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", *IEEE Trans. Soft. Engr. SE-9(6)*, (1983), 639-648.
 [5] Boehm, B., *Software Engineering Economics* Prentice-Hall, Englewood Cliffs, NJ (1981)
 [6] Boehm, B., M. Penedo, E.D. Stuckle, R.D. Williams, and A.B. Pyster, "A Software Development Environment for Improving Productivity", *Computer* 17(6), (1984), 30-44.
 [7] Boehm, B. and R.W. Wolverton, "Software Cost Modelling: Some Lessons Learned", *J. Systems and Software* 1(1980), 195-201.
 [8] Bailey, J. and V. Basili, "A Meta-Model for Software Development Resource Expenditures", *Proc. 5th. Intern. Conf. Soft. Engr.*, IEEE Computer Society, (1981), 107-116.
 [9] Vosburg, J., B. Curtis, R. Wolverton, B. Albert, H. Malec, S. Hoben and Y. Liu "Productivity Factors and Programming Environments", *Proc. 7th. Intern. Conf. Soft. Engr.*, IEEE Computer Society, (1984), 143-152.
 [10] Lawrence, M.J., "Programming Methodology, Organizational Environment, and Programming Productivity", *J. Systems and Software*, 2(1981), 257-269.
 [11] Cusumano, M. and C.F. Kemerer, "A Quantitative Analysis of U.S. and Japanese Practice and Performance in Software Development", *Management Science*, 36(11), (1990),
 [12] Thadhani, A.J., "Factors Affecting Programmer Productivity During Application Development", *IBM Systems J.* 23(1),

- (1984), 19-35.
- [13] Lambert, G.N., "A Comparative Study of System Response Time on Programmer Development Productivity", IBM Systems J. 23(1), (1984), 36-43.
- [14] Cervený, R.P., and D.A. Joseph, "A Study of the Effects of Three Commonly Used Software Engineering Strategies on Software Enhancement Productivity", Information & Management, 14, (1988), 243-251.
- [15] Rajiv D. B., Srikant M. D., and Chris F. Kemerer, "Factors Affecting Maintenance Productivity: An Exploratory Study", Proceedings of the 8th International Conference on Information Systems (ICIS), Pittsburgh, Pennsylvania, pp. 160-175, December 1987.
- [16] IEEE Standard for Software Productivity Metrics, IEEE Std. 1045-1992, IEEE Standards Board, 1993.
- [17] Low, G. C., and D. R. Jeffery, "Function Points in the Estimation and Evaluation of the Software Process," IEEE Transactions of Software Engineering, January 1993, 64-71.
- [18] Vicinanza, S. S., T. Mukhopadhyay, and J. J. Prietula, "Software-Effort Estimation: An Exploratory Study of Expert Performance," Information Systems Research, 243-262, December 1991.
- [19] Bigus, J. P., Data Mining with Neural Networks, McGraw-Hill, New York, 1996.
- [20] Hakkarainen J., P. Laamamen and R. Rask, "Neural Networks in Specification Level Software Size Estimation", Neural Network Applications, P. K. Simpson, IEEE technology Update Series, pp. 887-895, 1993.
- [21] Srinivasan K. and D. Fisher, "Machine Learning Approaches to Estimating Software Development Effort", IEEE Trans. Soft. Eng., vol. 21, no. 2, Feb. 1995, pp. 126-137.
- [22] Finnie G.R. and G. E. Wittig, "AI Tools for Software Development Effort Estimation", Software Engineering and Education and Practice Conference, IEEE Computer Society Press, pp. 346-353, 1996.
- [23] Mair C., G. Kadoda, M. Lefley, K. Phalp C. Schofield, M. Shepperd and S. Webster "An Investigation of Machine Learning Based Prediction Systems", Empirical Software Engineering Research Group, <http://dec.bmth.ac.uk/ESERG>, 9 July, 1999.
- [24] Shukla K. K., "Neuro-genetic prediction of software development effort", Information and Software Technology, 42 (2000) 701-713.
- [25] Brainstorm, "Overview of Neural Computing", 2001, www.brainstorm.co.uk