The application of intelligent and soft-computing techniques to software engineering problems: a review

Ramakanta Mohanty
Computer Science Department, Berhampur University, Berhampur 760 007, Orissa, India
E-mail: ramakanta5a@gmail.com

Vadlamani Ravi*
Institute for Development and Research in Banking Technology, Castle Hills Road #1, Masab Tank, Hyderabad 500 057, AP, India
E-mail: rav_padma@yahoo.com
*Corresponding author

Manas Ranjan Patra
Computer Science Department, Berhampur University, Berhampur 760 007, Orissa, India
E-mail: mrpatra12@gmail.com

Abstract: This paper presents a comprehensive review of the work done during 1990–2008 in the application of intelligent techniques to solve software engineering (SE) problems. The review is categorised according to the type of intelligent technique applied viz. (1) neural networks (NNs), (2) fuzzy logic, (3) genetic algorithm, (4) decision tree, (5) case base reasoning and (6) other techniques subsuming soft computing. Further, the source of the data set and the results whenever available are also provided. We find that NNs is the most often used non-parametric method in SE and there exists immense scope to apply other equally famous methods such as fuzzy logic, decision trees and rough sets. The review is going to be useful to researchers as a starting point as it provides important future research directions. For practitioners also, the review would be useful. This would eventually lead to better decision making in SE thereby ensuring better, more reliable and cost effective software products.

Keywords: SE; software engineering; software reliability prediction; software development cost estimation; NNs; neural networks; GA; genetic algorithm; decision tree; fuzzy logic; CBR; case-based reasoning; soft computing.

1 Introduction

The software engineering (SE) community is increasingly recognising the value of empirical evidence to support research and practice. Empirical evidence provides a means to evaluate the utility of promising research areas and to help practitioners to make informed technology adoption decisions. It aims to provide SE researchers and practitioners with appropriate knowledge and training in different methods and techniques to design, execute, analyse and report empirical research (Pedrycz, 2002). The goal of this review is to introduce a wide range of qualitative and quantitative methods, which can be used to gather and use empirical evidence to evaluate various SE technologies (i.e. methods, techniques and tools). It will also demonstrate different ways of applying empirical approaches to guide SE research (Basili, 1996). The review will provide a general introduction to intelligent techniques followed by a detailed discussion of controlled experiments, surveys, case studies, action research and systematic reviews. The review will also evaluate the strength and limitations of different methods and techniques.

Empirical software engineering (ESE) provides a forum for applied SE research with a strong empirical component, and a venue for publishing empirical results relevant to both researchers and practitioners. Empirical studies presented usually involve the collection and analysis of data and experience that can be used to characterise, evaluate and reveal relationships between software development deliverables, practices and technologies. Over time, it is expected that such empirical results will form a body of knowledge leading to widely accept and well-formed theories. ESE can also consist of
The application of intelligent and soft-computing techniques

case studies, field studies or industrial data from development and maintenance, and anecdotal evidence gathered as a by-product of an SE task. Data gathered from ESE can describe individual or group behaviour; it can be qualitative or quantitative and can be related to human, technical, economical or educational aspects of SE (Basili, 1996).

Software has been a major enabling technology for advancing modern society, and is now an indispensable part of daily life. Because of the increased complexity of these software systems, and their societal role, more effective software development and analysis technologies were needed (He and Yu, 2007). An important advantage of machine learning (ML) over statistical analysis as a modelling technique lies in the fact that the interpretation of production rules was more straightforward and intelligible to human beings than principal components and patterns with numbers that represented their meanings. Further, all the intelligent techniques are non-parametric unlike their statistical counterparts. A reliable and accurate estimate of software reliability and software development effort has always been a challenge for both the software industrial and academic communities. It is well-known that software project management teams can greatly benefit from knowing the estimated cost of their software projects. The benefits can have greater impact if accurate cost estimations are deduced during the early stages of the project life cycle. Estimating the process and performance behaviour of a software project, early in the software life cycle, is very challenging and often very difficult. Top down and bottom up are traditionally the most commonly used techniques to estimates the effort of software projects (AIAA, 1992; Lyu and Nikora (1992); Moody and Darken, 1989).

In SE, entity is categorised into three categories, that is, processes, products and resources. Processes are software related activities, such as constructing specification, detailed design or testing. Products refer to output of a particular process, that is, deliverables, documents that result from a process activity, such as a specification document, a design document or a segment of code. Resources are entities, which are required by a process to do a particular task, such as personnel, software tools or hardware. The aforementioned entities have internal and external attributes. Internal attributes describe an entity itself, whereas external attributes characterise the behaviour of an entity. ML methods have been utilised to develop better software products, to be part of software products, and to make software development process more efficient and effective (He and Yu, 2007; Zhang and Tsai, 2007).

We present a partial list of SE areas where machine-learning techniques have found applications (Zhang and Tsai, 2007):

- Predicting or estimating measurements of processes of internal or external attributes of processes, products or resource. These include: software quality, software size, software development cost, project or software effort, maintenance task effort, software resource, correction cost, software reliability, software defect, reusability, software release timing, productivity, execution times and testability of program modules.

- Discovering either internal or external properties of processes, products or resources. These include: loop invariants, objects in programs, boundary of normal operations, equivalent mutants, process models and aspects in aspect-oriented programming.

- Transforming products to accomplish some desirable or improved external attributes. These include; transforming serial programs to parallel ones, improving software modularity and mapping OO applications to heterogeneous distributed environments.
• Synthesising or generating various products which includes the test data, test resource, project management rules, software agents, design repair knowledge, design schemes, data structures, programs/scripts, project management schedule and information graphics.

• Reusing products or processes by including the similarity computing, active browsing cost of rework, knowledge representation, locating and adopting software to specifications, generalising program abstractions and clustering of components.

• To enhance the processes: these include: deriving specifications of system goals and requirements, extracting specifications from software, acquiring knowledge for specification refinement and augmentation, and acquiring and maintaining specification consistent with scenarios (Senyard et al., 2003).

• Managing products: these include: collected and managing software development knowledge, and maintaining software process knowledge.

The rest of this paper is organised in the following manner. In Section 2, an overview of review methodology is presented. Section 3, briefly describes the various stand-alone intelligent methods that are applied in this paper. Section 4 presents some of the important insights that emerged out of the review. Section 5 suggests some future directions in the field and concludes the review.

2 Review methodology

The review is conducted in two broad areas:
1 stand-alone intelligent techniques
2 hybrid techniques.

The intelligent techniques covered in the study belong to
1 different neural network (NN) architecture including multilayer perception (MLP) and cascade correlation NN
2 fuzzy logic
3 genetic algorithm (GA)
4 decision tree
5 case-based reasoning (CBR)
6 soft computing (hybrid intelligent systems).

The other techniques are
1 analogy based
2 support vector machine
3 SOM etc.

Table 1 presents the distribution of research papers in terms of techniques applied and the SE problems solved. Further, from Table 2, where journal-wise distribution of papers is presented, we found that the maximum number of papers appeared in IEEE Transactions on Software Engineering followed by Journal of Systems and Software. We did not include this papers presented in international conferences in this table.
The application of intelligent and soft-computing techniques

Table 1  Distribution of various intelligent techniques applied to software engineering problems

<table>
<thead>
<tr>
<th>Software engineering area</th>
<th>Type of intelligent technique used</th>
<th># papers</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuzzy logic</td>
<td></td>
<td>8</td>
<td>Chen (2008), Cimpan and Oquendo (2003), Lee et al. (2002), Rine et al. (1999), So et al. (2002), Zadeh (1994), Zhao et al. (2003) and Zimmermann (1996)</td>
</tr>
<tr>
<td>Fuzzy logic</td>
<td></td>
<td>6</td>
<td>Ahmed et al. (2005), Bajaj et al. (2006), Fahlman and Lebiere (1990), Mustilek et al. (2000), Xun and Khosgoflaa (2004) and Yuan et al. (2000)</td>
</tr>
<tr>
<td>Genetic algorithm</td>
<td></td>
<td>3</td>
<td>Burgess and Lefley (2001), Huang et al. (2007a,b) and Yang et al. (2006)</td>
</tr>
<tr>
<td>CBR</td>
<td></td>
<td>2</td>
<td>Delany and Cunningham (2000) and Sakran (2006)</td>
</tr>
</tbody>
</table>
Table 1  Distribution of various intelligent techniques applied to software engineering problems (continued)

<table>
<thead>
<tr>
<th>Software engineering area</th>
<th>Type of intelligent technique used</th>
<th># papers</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software project management</td>
<td>Neural networks</td>
<td>1</td>
<td>Idri et al. (2002)</td>
</tr>
<tr>
<td>Software testing</td>
<td>Neural networks</td>
<td>1</td>
<td>Khoshgoftaar and Szabo (1996)</td>
</tr>
<tr>
<td></td>
<td>Fuzzy logic</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2  Distribution of papers in various refereed journals

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Name of the journal</th>
<th>Total number of papers reviewed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IEEE Transactions on Software Engineering</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>IEEE Transactions on Software Reliability</td>
<td>04</td>
</tr>
<tr>
<td>3</td>
<td>IEEE Transactions on fuzzy system</td>
<td>01</td>
</tr>
<tr>
<td>4</td>
<td>IEEE Transactions on Neural Network</td>
<td>01</td>
</tr>
<tr>
<td>5</td>
<td>Journal of Systems and Software</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>European Journal of Operational Research</td>
<td>02</td>
</tr>
<tr>
<td>7</td>
<td>Expert Systems with Applications</td>
<td>06</td>
</tr>
<tr>
<td>8</td>
<td>Information and Software Technology</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>Fuzzy Sets and Systems</td>
<td>08</td>
</tr>
<tr>
<td>10</td>
<td>Annals of Software Engineering</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>IEEE Application of Software Engineering</td>
<td>01</td>
</tr>
<tr>
<td>12</td>
<td>ACM SIGSOFT Software Engineering</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>Int. J. Project Management</td>
<td>02</td>
</tr>
</tbody>
</table>

In this context, one may note that all the intelligent techniques barring fuzzy logic are data-driven in nature, all of them do possess some merits and demerits, which are presented in Table 3. A cursory glance at the Table 3 indicates when each technique is applicable. Under each category of intelligent technique, these papers are reviewed in the chronological order. Thus, the important dimension of the present review is the type of techniques applied. The review is conducted across other dimensions also such as

1  source of data
2  results obtained in each study.
The application of intelligent and soft-computing techniques

<table>
<thead>
<tr>
<th>Technology</th>
<th>Basic Idea</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 FL Models imprecision and ambiguity in the data using fuzzy sets and incorporates the human experiential knowledge into the model</td>
<td>Good at deriving human comprehensible fuzzy ‘if-then’ rules; It has low computational requirements</td>
<td>Arbitrary choice of membership function skews the results, although triangular shape is the most often used one. Secondly, the plethora of choices for membership function shapes, connectives for fuzzy sets and defuzzification operators are the disadvantages</td>
<td></td>
</tr>
<tr>
<td>2 NN Learn from examples using several constructs and algorithms just like a human being learns new things</td>
<td>Good at function approximation, forecasting, classification, clustering and optimisation tasks depending on the neural network architecture</td>
<td>The determination of various parameters associated with training algorithms is not straightforward. Many neural network architectures need a lot of training data and training cycles (iterations)</td>
<td></td>
</tr>
<tr>
<td>3 GA Mimics Darwinian principles of evolution to solve highly non-linear, non-convex global optimisation problems</td>
<td>Good at finding global optimum of a highly non-linear, non-convex function without getting trapped in local minima</td>
<td>Does take long time to converge; May not yield global optimal solution always unless it is augmented by a suitable direct search method</td>
<td></td>
</tr>
<tr>
<td>4 CBR Learns from examples using the Euclidean distance and k-nearest neighbour method</td>
<td>Good for small datasets and when the data appears as cases; similar to the human like decision-making</td>
<td>Cannot be applied to large datasets; poor in generalisation</td>
<td></td>
</tr>
<tr>
<td>5 SVM It uses statistical learning theory to perform classification and regression tasks</td>
<td>It yields global optimal solution as the problem gets converted to a quadratic programming problem; It can work well with few samples</td>
<td>Selection of kernel and its parameters is a tricky issue. It is abysmally slow in test phase. It has high algorithmic complexity and requires extensive memory</td>
<td></td>
</tr>
<tr>
<td>6 Decision trees They use recursive partitioning technique and measures like entropy to induce decision trees on a dataset</td>
<td>Many of them can solve only classification problems while CART solves both classification and regression problems. They yield human comprehensible binary ‘if-then’ rules</td>
<td>Over fitting can be a problem. Like neural networks, they too require a lot of data samples in order to get reliable predictions</td>
<td></td>
</tr>
<tr>
<td>7 SC Hybridises fuzzy logic, neural networks and genetic algorithms, etc. in several forms to derive the advantages of all of them</td>
<td>It amplifies the advantages of the intelligent techniques while simultaneously nullifying their disadvantages</td>
<td>Apparently, it has no disadvantages. However, it does require good amount of data, which is not exactly a disadvantage nowadays</td>
<td></td>
</tr>
</tbody>
</table>
Table 3 shows that datasets and techniques applied in different areas of SE application. Further, the review concentrated on peer reviewed journals/international conferences/edited volumes in the areas of NNs, fuzzy logic, GAs, CBR and decision trees. Moreover, in this paper, when multiple techniques are compared in their stand-alone mode, the technique proposed in this paper is taken as the main criterion and accordingly this paper is categorised in that family. For example, when in a paper, NN, FL and logit (logistic regression) are compared in their stand-alone mode and NN is proposed in that paper, then this paper is reviewed under the NN section.

Incidentally, our work is different from many reviews appeared recently and not related to SE. To mention one such work, Chen (2008) reviewed various artificial intelligence techniques viz. CBR, rule-based system, NN, GA, cellular automata, fuzzy models, multiagent systems, swarm intelligence, reinforcement learning and hybrid systems. They also provided briefly the mathematical background of each of these techniques and mentioned one application for each of these of techniques. However, the present review explicitly focuses on the applications of intelligent and soft-computing techniques in SE problems, whereas their paper is a very general one without focus on any particular domain.

2.1 Neural networks

NNs (Kolodner, 1993; Lyu, 1996) are named after the cells in the human brain that perform intelligent operations. The brain is made up of billions of neuron cells. Each of these cells is like a tiny computer with extremely limited capabilities. NNs are formed from hundreds or thousands of simulated neurons connected together in much the same way as the brain’s neurons. A NN is a system of massively parallel, interconnected computing units called neurons, arranged in layers. The NNs found extensive applications in all areas of science and engineering in general and ESE problems in particular. The multilayer perceptron (MLP) (Rumelhart, et al.1986), radial basis function network (RBFN) (Moody and Darken,1989), probabilistic neural network (PNN) (Specht, 1990), cascade correlation (Cascor) NN (Fahlman and Lebiere, 1990), learning vector quantisation (LVQ) (Gersho and Gary, 1992) and self-organising feature map (SOM) (Kohonen, 1997) are some of the popular NN architectures. They differ in aspects including the type of learning, node connection mechanism, the training algorithm, etc.

A NN by Park et al. (2006) is a powerful data analysis tool, that is, able to capture and represent complex input/output relationships. The motivation for the development of NN technology is to develop an artificial system that could perform ‘intelligent’ tasks similar to those performed by the human brain. NNs resemble the human brain in the following two ways:

1. NNs acquire knowledge through learning
2. A NN’s knowledge is stored within inter-neuron connection strengths known as synaptic weights.

The true power and advantage of NNs lies in their ability to represent both linear and non-linear relationships and learn these relationships directly from the data being modelled (Perlovsky, 2000). Traditional linear models are simply inadequate when it comes to modelling data that contains non-linear characteristics. The most common NN model is the MLP. This type of NN is known as a supervised network because it requires...
The application of intelligent and soft-computing techniques

a desired output in order to learn. The goal of this type of network is to create a model that correctly maps the input to the output using historical data, so that the model can then be used to produce the output when the desired output is unknown.

2.2 Fuzzy set theory

Fuzzy logic is a powerful problem-solving methodology with a myriad of applications in embedded control and information processing. Fuzzy provides a remarkably simple way to draw definite conclusions from vague, ambiguous or imprecise information. In a sense, fuzzy logic resembles human decision making with its ability to work from approximate data and find precise solutions. Unlike classical logic, which requires a deep understanding of a system, exact equations, and precise numeric values, fuzzy logic incorporates an alternative way of thinking, which allows modelling complex systems using a higher level of abstraction originating from our knowledge and experience. Fuzzy logic allows expressing this knowledge with subjective concepts such as very hot, bright red and a long time, which are mapped into exact numeric ranges.

Fuzzy logic has been gaining increasing acceptance during the past few years. There are over two thousand commercially available products using fuzzy logic, ranging from washing machines to high-speed trains. Nearly, every application can potentially realise some of the benefits of fuzzy logic, such as performance, simplicity, lower cost and productivity. Fuzzy set theory, proposed by Zadeh (1965) has found a number of applications. It is a theory of graded concepts. It provides a mathematical framework where vague, conceptual phenomena can be rigorously studied (Zimmermann, 1996). Fuzzy logic models human experiential knowledge in any domain. When applied to solve process control or prediction problems fuzzy logic takes the help of the knowledge of the domain expert and employs fuzzy mathematics to come out with fuzzy inference systems. Fuzzy logic can also be used to derive fuzzy ‘if-then’ rules from data to solve classification problems.

2.3 Genetic algorithm

GA was developed as an alternative technique for tackling general optimisation problems with large search spaces. They have the advantages that they do not need prior knowledge, expertise or logic related to the particular problem solution. But for most problems with a large search space, a good approximation to the optimum is a likely outcome. The basic ideas are based on the Darwinian theory of evolution, which in essence says that genetic operations between chromosomes eventually leads to fit individuals which are more likely to survive. Thus, over a long period of time, the population of spics as a whole improves. GA proposed by Holland (1975) is a most popular evolutionary algorithm. Solutions to a given problem are encoded into chromosome like data structure named genotype. The objective function of the problem to be solved is encoded into a special function called fitness function. This function embraces all requirements imposed on solutions to the problem. The genotypes after being decoded into phenotypes that are solutions in the problem domain constitute inputs to the fitness function. They are evaluated based on their ability to solve the problem. The result of evaluation is used in a selection process based on favouring individuals with higher fitness value. The selection process can be formed in numerous ways. One of them is called stochastic sampling with replacement. In this method, the entire population is
mapped onto a roulette wheel where the area corresponding to its fitness represents each individual. Repetitive spinning of the roulette wheel chooses individuals of the next population.

The basic process of the GA is as follows

1. generate at random a population of solutions, that is, a family of chromosomes
2. create a new population from previous one by applying genetic operators to the fittest chromosomes, or pairs of fittest chromosomes of the previous chromosomes
3. repeat step (2), until either the fitness of the best solution has converged or a specified number of generations have been completed.

The best solution in the final generation is taken as the best approximation to the optimum for that problem that can be attained in that run. The whole process is normally run a number of times, using different seeds to the pseudo-random number generator. The key parameters that have to be determined for any given problem are

1. the best way of representing a solution as a fixed length binary string
2. the best combination of genetic operators. For GA, reproduction, crossover and mutation are the most common
3. choosing the best fitness production to measure the fitness of a solution
4. trying to keep enough diversity in the solutions in a population to allow the process to converge to the global optimum but not converge prematurely to a local optimum by Purnapraja et al. (2006).

2.4 Decision trees

Decision trees, proposed by Gehrke et al (1999) and Quinlan (1987), form a part of ‘ML’, which is an important area of artificial intelligence. Majority of the decision tree algorithms are used for solving classification problems. However, algorithms like classification and regression trees (CART) can be used for solving regression problems also. All these algorithms induce a binary tree on the given training data, which in turn results in a set of ‘if-then’ rules. These rules can be used to solve the classification or regression problem. A number of algorithms are used for building decision tree including Chi squared automatic interaction detection (CHAID), CART, Quest and C5.0 (Rud, 2000).

2.5 Case-based reasoning

CBR, a branch of artificial intelligence, is intuitively similar to the cognitive process humans follow in problem solving (Kolodner, 1992; Watson and Marir, 1994). CBR is a problem-solving paradigm that in many respects is fundamentally different from other major AI approaches. Instead of relying solely on general knowledge of a problem domain, or making associations along generalised relationships between problem descriptors and conclusions, CBR is able to utilise the specific knowledge of previously experienced, concrete problem situations (cases). Finding a similar past case and reusing it in the new problem situation solves a new problem. A second important difference is that CBR also is an approach to incremental, sustained learning, since a new experience
The application of intelligent and soft-computing techniques

is retained each time a problem has been solved, making it immediately available for future problems. The CBR field has grown rapidly over the last few years, leading to commercial tools, and successful applications in daily use. When people confront a new problem, they often depend on past similar experiences and reuse or modify solutions of these experiences to generate a possible answer for the problem at hand. The hallmark of CBR is its capability to give an explanation for its decision based on previous cases. Citing relevant previous experiences or cases is a way to justify a position in human decision-making (Kolodner, 1993). Comprehensibility of the decision is often crucial in solving software reliability/software cost estimation problems. When a company is identified as failing, CBR can give examples of similar companies that failed in the past as a justification for its prediction. The heart of the CBR is the nearest neighbour algorithm proposed by Russell and Norvig (2003).

2.6 Support vector machine (SVM)

SVM introduced by Vapnik and Haykin (1998) uses a linear model to implement non-linear class boundaries by mapping input vectors non-linearly into a high-dimensional feature space. In the new space, an optimal separating hyper plane (OSH) is constructed. The training examples that are closest to the maximum margin hyper plane are called support vectors. All other training examples are irrelevant for defining the binary class boundaries. SVM is simple enough to be analysed mathematically. In this sense, SVM may serve as a promising alternative combining the strengths of conventional statistical methods that are more theory-driven and easy to analyse and ML methods that are more data-driven, distribution free and robust. Recently, The SVM has been used to financial applications such as credit rating that SVM is comparable to and even outperform other classifiers including BPNN, CBR, MDA and logit in terms of generalisation (Cristiamini and Taylor, 2000).

2.7 Soft computing

The paradigm of soft computing or computational intelligence refers to the seamless integration of different, seemingly unrelated, intelligent technologies such as fuzzy logic, NNs, GAs, ML (CBR and decision trees subsumed), rough set theory and probabilistic reasoning in various permutations and combinations to exploit their strengths. This term was coined by Zadeh (1994) in the early 1990s to distinguish these technologies from the conventional ‘hard computing’, that is, inspired by the mathematical methodologies of the physical sciences and focused upon precision, certainty and rigor, leaving little room for modelling error, judgement, ambiguity or compromise. In contrast, soft computing is driven by the idea that the gains achieved by precision and certainty are frequently not justified by their costs, whereas the inexact computation, heuristic reasoning and subjective decision making performed by human minds are adequate and sometimes superior for practical purposes in many contexts. Soft-computing views the human mind as a role model and builds upon a mathematical formalisation of the cognitive processes those humans take for granted (Venkatachalam, 1993). Within the soft-computing paradigm, the predominant reason for the hybridisation of intelligent technologies is that they are found to be complementary rather than competitive in several aspects such as efficiency, fault and imprecision tolerance and learning from examples (Venkatachalam, 1993). Further, the resulting hybrid architectures tend to minimise the disadvantages of
the individual technologies while maximising their advantages. Some of the soft-
computing architectures employed are neuro-fuzzy, fuzzy-neural, neuro-genetic, genetic-
fuzzy, neuro-fuzzy-genetic, rough-neuro, etc. Multiclassification systems or ensemble
classifiers are also treated as soft-computing system.

3 Review of techniques applied to SE problems

3.1 NNs applied to predict software reliability

NNs have drawn people’s attention in recent years and a good amount of efforts have
been devoted to them Khosgoftaar and Szabo (1996), Khosgoftaar et al. (1993, 1997),
Khosgoftaar and Rebours (2003) to handle numerous factors in SE and approximate any
non-linear continuous function in theory. In these works, it was argued that NN method
could be applied to estimate the number of software defects, classify program modules,
and forecast the number of observed software failures, and they often provided better
results than existing methods. In this section, we will review briefly the works carried in
the areas of software reliability. Cai et al. (1991) presented a review on software
reliability modelling. The review discussed about the different type of probabilistic
software reliability model and their shortcomings. They developed a simple yet powerful
fuzzy software reliability model, which was powerful alternative to other methods. Then,
Karunanithi et al. (1991) employed the use of feed forward NNs to predict the software
reliability. They used five different types of models such as exponential model,
logarithmic model, inverse polynomial mode, delay S-shape and power model. They used
three different types of dataset Data 1 (Matsumoto et al., 1988), data 2 (Musa et al.,
1987), Data 3 (Ohba, 1984a). These datasets represents the failure history of the systems.
The dataset consists of two observations: one representing the cumulative execution time
and other represent the accumulated number of fault observed. They trained the network
with the execution time as the input and observed fault count as the associated output
response. They found that the NN models were consistent in prediction and their
performance is comparable to that of other parametric models. Later, Karunanithi et al.
(1992b) illustrated the usefulness of connectionist model for software reliability
predictions. The connectionist model is compared with five reliability growth models viz.
exponential model, logarithmic model, the delayed S-shaped f-model (Ohba, 1984a,b;
Ohba et al., 1982), inverse polynomial model and power model. This connectionist
approach offers easy construction of complex models and estimation of parameters as
well as good adaptability for different dataset. They used dataset collected from different
software systems to compare the models. Based on the experiment, they showed that the
scaled representation of input–output variables provided better accuracy than the binary
coded (or grey coded) representation. The experimental result obtained by them showed
that the connectionist networks had less end point prediction errors than parametric
models. Thereafter, Karunanithi et al. (1992a) presented a solution to the scaling
problems in which they used a clipped linear unit in the output layer. The NN could
predict positive values in any unbounded range with a clipped linear unit. They
demonstrated the applicability of the NNs structure with 03 dataset. The predictive
capability of NN models with clipped linear output units and network with sigmoid
output units are compared using three dataset, data 1 (Ohba, 1984a), data 2 (Ohba,
1984b), Data 3 (Ohba et al., 1982). They used different types of analytical models such as
The application of intelligent and soft-computing techniques

logarithmic model, inverse-polynomial, power model, delay S-shaped and exponential model. Among the models, the Jordan network model exhibits a better accuracy than the feed forward model.

Subsequently, Khoshgoftaar et al. (1992) explored the use of NN for predicting the number of faults in a program. They used static reliability modelling and compare its performances in terms of prediction quality of fit with that of regression models. They used dataset obtained from Ada development environment for the command control of a military data line communication system. They found that the absolute relative error of NN is less compared to regression model. Then, Khoshgoftaar and Szabo (1994) used the principal component analysis (PCA) on NN for improving predictive quality. They used regression modelling and NN modelling to predict the reliability and the number of fault. The datasets used for the prediction were collected from large commercial software system. They developed three different regression models as quality metric domain (QM) regression model, design change request (DCR) metric regression mode and problem tracking request (PTR) model used. They found that regression model did not correspond well in linking complexity and fault. They developed two NN models, that is, model $\text{NN}_{\text{raw trained}}$ using ten raw software complexity metric with 40 hidden layers and output neuron and $\text{NN}_{\text{mod}}$ was trained using domain metric with six domain metric as input with 24–40 hidden and neuron output. They also used QM, DCR and PTR NN for experiment. They found that NN model using domain metric as input outperformed both regression model NN using raw input metric.

Further, Khoshgoftaar and Szabo (1996) used NN to investigate the application of PCA to predict the number of faults. The datasets used for prediction of faults collected from commercial software systems. They extracted PCA from these measures. They trained two NNs, one with observed data (raw) and other with principal components. They compare the predictive quality of two models and found that PCA yielded better results. Furthermore, Khoshgoftaar et al. (1997) reported a case study of NN modelling techniques developed for early risk assessment of latent defect (EMERALD) for improving the reliability of telecommunication software products. They applied NN techniques to find out fault prone modules and also to detect the risk of operational problems. They used a variety of classification techniques to identify fault prone software including discriminant analysis, classification tress and pattern recognition. The dataset used for prediction was taken from a large telecommunication system. They compared the NN model with non-discriminant model and found that NN model provided better accuracy. Later, Khoshgoftaar et al. (2000) reported the implementation of CART to evaluate software quality models over several releases. The case study developed in two classifications tree models based on four consecutive releases of very large telecommunication systems. The model 1 used the measurement of first release of training datasets of 11 important predictions. The model 2 used measurement of the second release as the training dataset but having 15 predictions. Analysis of the models yielded insight into various software development practices. They found that both modules had accuracy that would be useful to predict the different new models releases about the accuracy.

Meanwhile, Houchman and Hudepohl (1997) compared the performances of discriminant analysis back propagation neural network (BPNN) and evolutionary NN approach on datasets collected from telecommunication systems with 12 million lines of code written in Pascal like proprietary language. The PCA was carried out to split the dataset into training, testing and validation test without significant loss of information.
The dataset contained 6,972 modules, which was classified into fault prone based on number of fault by using data splitting algorithm. Based on experimental results, they found that ENN’s showed superior performance on the range of data configuration used.

On the other hand, Dohi et al. (1999) used MLP and recurrent NN to estimate the software release time for cost minimisation problem. They interpreted cost minimisation problem as a graphical one and showed that cost minimisation problem could be reduced to time series forecasting problem. They used datasets cited by Lyu (1996) and found that the predictive performance of the optimal software release time by MLP was better than the existing parametric SRGMs. Sitte (1999) compared the performance of two different methods of software reliability prediction, that is, NN and recalibration for parametric models. Both methods were claimed to predict as good as conventional parametric models. Each method applied its own, predictability measure impeding a direct comparison. He used a common predictability measure and common datasets for comparing the prediction results. NN were not only much simpler to use than the recalibration method, but that they were equal or better trend predictors. The NN prediction was further improved by preparing the data with a running average, instead of the traditionally used averages of grouped data points. Adnan et al. (2000) reported the use of MLP for software reliability. BPN algorithm was used for NN implementation, which composed of eight sets of data. They used the SE failure data from NASA, which exhibited features of a reliable and sizable software. Cai et al. (2001) aimed to quantify software reliability status and behaviour by handling dynamic software reliability data. They used MLP to handle time between successive software failures observed in successive time intervals. They further observed that

1. the NN approach is more appropriate for handling dataset with ‘smooth’ trends than with large fluctuations
2. The training results are much better than the test results in general
3. The empirical probability distribution of test data resembles that of the training data.

A NN can qualitatively predict what it has learnt. The result showed that the NN approach does not generate satisfactory quantitative results of the software reliability modelling. Patra (2003) used a MLP to predict for long-term software failure. He estimated a long-term mean time to failure (MTTF) for the dataset with the cumulative execution of time as an input and cumulative number of failures as output. The results obtained through proposed method were compared with computer-aided software reliability estimation (CASRE) tool. He observed that the information provided by the MTTF was more accurate as compared to existing methods. Ho et al. (2003) used modified Elman Recurrent NNs in modelling and predicting software failure. They compared the recurrent architecture with MLP and the Jordan recurrent model. The predictive performances were evaluated using the mean absolute deviation and directional change accuracy. The dataset used for analysing the failure data was taken from a complex military computer system. They compared the analytical model and NN models and found that NN models outperform traditional software.

There is a variety of statistical techniques used in software quality modelling and models are based on statistical relationship between measures of quality and measurement of software metrics. But the relationship between static software metrics and quality factors are complex and non-linear. Quah and Thwin (2004) used general regression neural network (GRNN) and Ward NN to evaluate the capability of SQL
The application of intelligent and soft-computing techniques

metrics for finding out software development fault. Based on actual project defect data, the SQL metrics are empirically validated by analysing their relationship with the probability of fault detection across PL/SQL. SQL metrics were extracted from oracle PL/SQL code of a warehouse management database application system. The faults were collected from journal files that contain the documentation of all changes in source file. The experimental data were collected from a set of warehouse management application, that is, developed using C, JAM and PL/SQL languages. They used PCA to identify the underlying orthogonal dimensions that explain relations between variables in the dataset. From the experimental results they found that SQL metrics should be useful in predicting faults in PL/SQL files.

One of the important issues in training NN is generalisation. When training set is small and deteriorates by random noise, the network becomes over trained and fits to the noise, while over fitting the noisy data degrades the predictive accuracy of the network. Guo and Lyu (2004) proposed the use of pseudo-inverse learning algorithm (PIL) for MLP with stacked generalisation applied to SRGM to overcome the degradation predictive accuracy of the NNs. Their architecture has the same number of hidden neurons as the number of examples to be learned. The PIL algorithm eliminates learning errors by adding hidden layers. The PIL algorithm was feed-forward, fully automated, which included no critical user dependent parameters such as learning rate or momentum constants. Learning the previously trained weights in the network was not changed. The learning errors were minimised separately on each layer instead of globally for the network as a whole. The algorithm was more effective than the standard BPN and other gradient descent algorithm. The algorithm was tested on case studies with stacked generalisation application to software reliability growth modelling data. They used Sys 1 and Sys 2 (Dohi et al., 1999) software failure datasets for the above experiment.

Most of the published literature used single-input single-output NN architecture to build software reliability growth models (Karunanithi et al., 1991, 1992b). Recent studies focus on modelling software reliability based on multiple-delayed-input single-output NN architecture. Tian and Noore (2005a) proposed an on-line adaptive software reliability prediction model using evolutionary connectionist approach based on multiple-delayed-input single output architecture. They modelled the interrelationship among software failure time data instead of the relationship between failure sequence number and failure time data. Desiring an online failure prediction process, GA was used to globally optimise the NN. The optimisation process determined the number of delayed input neurons corresponding to the previous failure time data sequence and the number of neurons in the hidden layer as the training algorithm. They used four real time control and flight dynamic application dataset. The experimental result outperformed the existing NN model for failure time prediction.

Thereafter, Tian and Noore (2005b) proposed NN modelling approach for software cumulative failure time prediction based on multiple-delayed-input single-output architecture. GA was used to optimise the number of delayed input neurons and the number of neurons in the hidden layers of the NN architecture. They used the modification of Levenberg-Marquardt algorithm with Bayesian regularisation to improve the ability to predict software cumulative failure time. They compared the performance of prediction by using real-time control and flight dynamic application datasets with existing approaches. They found that the proposed approach yielded better accuracy in predicting software failure time.
Aggarwal et al. (2005a) proposed various training algorithms in a NN model and showed which algorithm was best suited for SE applications. They had compared a number of algorithms such as bias learning rules, BFGS Quasi-Newton method, Levenberg-Marquardt optimisation and BPN with Powell-Beale BPN algorithm, gradient descent with adaptive learning and Bayesian algorithm. The ISBSG dataset was used to train and test the NN. They found that Bayesian regularisation algorithm yielded best results.

Many object-oriented metrics for software quality prediction have been proposed in the last decade. Thwin and Quah (2005) presented the application of NN in software quality estimation using object-oriented metrics. They investigated two kind of operation. The first one was on predicting the number of defects in a class and the second one was predicting the number of lines changed per class. They used two types of NN models such as Ward NN and GRNNs. Object oriented design metrics concerning inheritance related measures, complexity measures, cohesion measures, coupling measures and memory allocation measures were used as the independent variable. GRNN network model was found to predict more accurately than Ward NN model. The NN model aimed to predict object oriented software quality by estimating the number of faults and the number of lines changed per class. They used software metrics including both object oriented metrics and traditional complexity metrics. They introduced Ward NN and GRNN to improve prediction results for estimating software quality. Ward NN was a BPN network with different activation function. They applied hidden layers slabs to detect different features in the mid range of data and a Gaussian component complement to other hidden slabs to detect the upper and lower extremes of data. Thus, the output layer would get different views of data. They combined the two feature sets in the output layer to a better prediction. They used quality evaluation system (QUES) dataset for the experiment. From the experiment it was found that the GRNN model predicted more accurately than Ward NN model.

There has been extensive work in measuring reliability using mean time between failure and MTTF. Gupta and Singh (2005) determined the estimation of software reliability by taking basic execution time model. In this model, they estimated the reliability after predicting the number of faults by using pattern mapping technique of artificial NNs. The pattern mapping showed the interpolative and extrapolative behaviour of NNs. Su et al. (2007) proposed a new NN for software reliability estimation. They predicted software reliability by designing a dynamic weighted combinational model (DWCM), which was demonstrated through real software failure datasets taken from Musa et al. (1987). They compared the DWCM with Goel–Okumoto model and Yamada delay (Yamada and Osaki, 1983a,b, 1985) model and found that DWCM outperformed them. Kanamani et al. (2007) used MLP and PNN as software fault prediction models using object-oriented metrics. The results obtained by these models were compared with that of the statistical methods (discriminant analysis and logistic regression) by computing five models quality parameters. Between the two, PNN was highly robust in nature, which was studied through the quality parameters viz. misclassification rate, correctness, effectiveness and efficiency.

Recently, Rajkiran and Ravi (2007b) proposed the use of wavelets neural networks (WNN) to predict software reliability. In WNN, they employed two kinds of wavelets – More wavelet and Gaussian wavelet as transfer function resulting in two variants of WNN. They compared the performance of WNN to that of multiple linear regression (MLR), multivariate adaptive regression splines (MARS), MLP, threshold accepting
The application of intelligent and soft-computing techniques

trained neural network (TANN), PI-sigma network (PSN), GRNN, dynamic evolving neuro-fuzzy inference system (DENFIS) and TreeNet in terms of normalised root mean square error (NRMSE) obtained on test data. They found that WNN outperformed all other techniques.

3.2 NNs on software effort estimation/cost estimation

Due to rapid increase in software development costs, software development imposes a heavy burden on companies seeking to implement enterprise information system. Before performing the costly software development processes, it is necessary to examine and evaluate the anticipated cost and profits of such projects. The estimated accuracy of software development cost plays an important role, both directly and indirectly, in a company’s decision on whether to make an investment or not. Such accurate estimation has a prominent impact on the success of projects. Various methods of software development effects have been proposed, that is, line of code (LOC) based, constructive cost model (COCOMO), function point-based regression model (FP), NNs model and case base reasoning.

Hsu and Tenerio (1991) used MLP and SOM to construct SE models from observation data. Out of two experiments conducted, the first experiment described the BPN and self-organising neural network (SONN) algorithm, which were used to develop effort estimation model for projects in the COCOMO database. The second experiment described the ability of these models to predict the effort required for new projects to be examined. The experimental result shows that the NN techniques were superior in performance, learning time and modelling power.

Venkatachalam (1993) explored the software cost estimation by use of the BPNN. The BPNN is constructed with 22 input nodes and 2 output nodes. The input nodes represented the distinguishing features of software projects and output nodes represented the effort required in terms of person month and the development time required to complete the project. The data used for estimation was taken from COCOMO database. He found that NN were accurately estimating software cost and development time. Wittig and Finnie (1997) evaluated the assessment of MLP for effort estimation on simulated data as well as actual data of commercial projects. This project data had large productivity variations, noise and missing data values, which enabled model evaluation under typical software development conditions. The results were encouraging, with network showing an ability to estimate development effort within 25% of actual effort more than 75% of the time for one large commercial dataset.

Later, Jun and Lee (2001) proposed a search method to the relevant level of effort estimation by using MLP. They adopted the beam search technique and case-set selection algorithm to the effort estimation. For the selected case set, eliminating the quantitative input factors with the same value could reduce the scale of MLP. They adopted beam search technique and devised the case set algorithm, to classify the case sets and corresponding NN model. Their proposal using the qualitative input factors from the model, resulted in the reduced NN model According to the paired t-test, they could prove that the quasi-optimal case-selective NN model could significantly reduce the errors more than the full NN model. Further, Leung (2000) provides a general overview of software cost estimation methods. Since a number of these models rely on a software size estimate as input, he first provided an overview of common size metrics. He classified the models...
Consequently, Heiat (2002) compared the prediction performances of MLP and RBF to that of regression analysis on the estimation of software development effort. In the experiments, he used the datasets from Kemerer, IBM and Hallmark datasets, which includes both third generation and fourth generation programming languages. They found that the combination of third generation language dataset produced improved performance over conventional regression analysis in terms of mean absolute percentage error.

However, Idri et al. (2002) interpreted the cost estimation model based on MLP. Their proposed idea comprises mainly of the use of a method that mapped the MLP to a fuzzy rule-based system. They used COCOMO 81 dataset for their case study. Further, Idri et al. (2004) compared the performances of the NN and RBFN model based on software cost estimation. They found that the variety of NN architecture did not estimate well due to their short coming of being ‘black boxes’ model. They used Benitez method to extract the ‘if-then’ fuzzy rules. These fuzzy rules expressed the information encoded in the architecture of the network, and the interpretation of each fuzzy rule determined by analysing its premise and its output. They also suggested another mapping method viz. the Jang and Sun method, to extract ‘if-then’ fuzzy rules from NN. This case study was based on the COCOMO’81 historical dataset. Recent research in the area of software cost estimation has been growing rapidly due to practicality and demand for it. Tadayon (2004) explored the use of expert judgement and NN referencing to COCOMO II approach to predict the cost of software. They found that expert judgement system was most common and successful technique used in software cost estimation.

Further, Huang et al. (2003b) explored the effects of accuracies of the software effort estimation models established from clustered data by using the International Benchmarking Standard Groups (ISBSG) repository. The ordinary least square (OLS) regression method adopted to establish a respective effort estimation method in each clustered of datasets. For building effort estimation model, they used data with clustering different test drivers viz. function points (FP), max. time size (MTS), development type (DT), development platform (DP), language type (LT), used methodology (UM), methodology acquired (MA) and application type (AT). In the feature selection stage, they selected the statistical tests of Pearson correlation and one-way ANOVA were utilised to identify the significant effort drivers for building the software effort estimation model. They used three evaluation criteria viz. MMRE, MdMRE and Pred (0.25). The empirical experimental result showed that software effort estimation model established from the homogenous datasets did not produce more accurate results than models established from the homogenous datasets.

Recently, Dillibabu and Krishnaiah (2005) employed COCOMO II 2000 model to estimate cost in terms of effort spent on an embedded system project. They implemented COCOMO II 2000 on 10 projects. Out of which 8 projects are developed projects and 2 were of porting projects. They collected the actual effort from metric database of the company. The lines of code enumerated using ‘code count’ tool to achieve the logical source lines of code for each project. The calibration of COCOMO II 2000 carried out based on log approach and curve fitting approach. The curve fitting approach yielded better estimates of the model parameter.

The most important factor of an IT industry is to accurately estimate the efforts and necessary costs of a project in initial stage of development. Park and Baek (2008)
proposed NN-based effort estimation model where they identified six input variables through expert interview and regression analysis. They attempted to discover the best combination of input variables, which generates accurate effort estimation. To develop a NN-based effort estimation model, they collected 148 IT projects completed between 1999 and 2003 from Korean IT Project service vendors. They compared the performance of NN-based model with human expert judgement and two regression models. They found that NN-based model showed superior result compared to existing regression analysis. Tronto et al. (2007) used MLP and stepwise regression-based predictive model to estimate the size, effort, cost and time spent in the development process of software project management. They compared MLP and stepwise regression (SR)-based predictive model with that of AFF, SLIM and COCOMO model. They found that MLP and SR were competitive with other models.

Most recently, Vinay Kumar et al. (2008) proposed the use of WNN to forecast the software development effort. They used two types of WNN with Morlet function and Gaussian function as transfer function and used threshold acceptance training algorithm for wavelet neural network (TAWNN). They compared WNN with other techniques viz. MLP, RBFN, MLR, DENFIS and SVM in terms of mean magnitude of relative error (MMRE). They obtained datasets from Canadian Financial (CF) dataset and IBM data processing services (IBMDPS) dataset. From the experiment, they found that WNN-Morlet for CF dataset and WNN-Gaussian for IBMDPS outperformed all other techniques.

Thereafter, Ravi et al. (2009) proposed WNN to predict operational risk in banks and firms by predicting software reliability. They employed two types of WNN as transfer function viz. Morlet wavelet and Gaussian wavelet. They compared the MLR, MARS, BPNN, TANN, threshold accepting trained wavelet neural network (TAWNN), PSN, GRNN, DENFIS and TreeNet in terms of normalised root mean square. They observed that NRMSE values of different techniques gradually decreased with increase of lag number and WNN-based models outperformed all the individual techniques over all the lags for both Gaussian-based and Morlet-based wavelets NNs.

### 3.3 Fuzzy logic approach

Uncertainty is present in our everyday lives. It is also present in ESE. Real world concepts transition smoothly into one another rather than abruptly. Therefore, fuzziness as a means of modelling linguistic uncertainty can be very well used to model real world problems and also SE problems. The overall life cycle cost associated with product failure exceeds 10% of yearly corporations turn over. A major factor contributing to this loss is ineffective performance of software and system verification, validation and testing (VVT). In this review paper, we report the application of fuzzy logic approach as the second intelligent technique to the areas of software reliability, software cost estimation and software effort estimation, etc. and the datasets used by different authors.

Rine et al. (1999) used reusable software controller architecture for designing a specific domain of controllers (zero order, first order, ..., nth order). The controller architecture composed of two-sub models 1

---

1 model identifier, which was on-line fuzzy rule-based system, which took a sample from the input and output signal, tried to approximate the plant model behaviour
2 neuro-fuzzy controller, which has a fuzzy controller that has fuzzy rule-based system by using feed forward NN.

They used software modules by incorporating automatic training and adaptive control technologies instead of using the traditional manual. The fuzzy logic allowed developing SE principles of modularity and testability in order to stable the specification level and make them adaptable at the representation level.

Yuan and Ganeshan (2000) integrated fuzzy subtractive clustering with module order for software quality prediction. They experimented the number of faults by using fuzzy subtractive clustering and also predicted whether modules were fault prone or not by using module-order modelling. The dataset used for fault prone prediction is legacy telecommunication systems. Different types of modelling techniques were used for software quality prediction as discriminant power and optimal set reduction. The membership function associated with a given fuzzy set was of linear, sigmoid or quadratic type. They found that fuzzy subtractive clustering could discover modules, which have faults discovered by customer with useful accuracy prior to release.

So et al. (2002) proposed an automated and fuzzy logic-based approach to identify potential error prone software components using inspection data. The effectiveness of their approach was demonstrated on inspection data. The empirical evaluation on launch interceptor program (LIP) and conflict inspection data as well as \( \chi^2 \) analysis confirmed the validity. The advantageous of this model used to provide a natural mechanism to model fuzzy data. Fuzzy rule used in determining error prone modules were fuzzy and prototype system could be developed without having to have extensive empirical data and that the system performances could be continuously tuned as more inspection data.

To understand the economy of software development, which would reduce software production cost overrun or even project cancellation, it is required to develop different models for software cost estimation. Musflek et al. (2000) estimated the effort/cost required for development of software products with fuzzy model. They used fuzzy set-based generalisation of the COCOMO model (f-COCOMO). The size of software was taken as a fuzzy set which yields the cost estimation. They estimated the software cost by analysing the different membership functions such as triangular and parabolic fuzzy set. The f-COCOMO model emphasises the way of propagation of uncertainty and visualisation of the resulting effect. Then, Zhao et al (2003) used COCOMO-II for estimating software cost. They used lines of code and FP to assess software size. The entity relationship (ER) model was used in conceptual modelling (requirements analysis) for data-intensive systems. They used ER model for the estimation of software cost.

However, Lee et al. (2002) focused on approaches especially devoted to extending fuzzy logic to SE along with the following three directions

1 applying fuzzy logic to requirements engineering
2 software quality prediction
3 fuzzy object oriented modelling.

They also reviewed fuzzy current approaches in SE that could provide powerful tool for requirement engineering, formal specification, software quality prediction and fuzzy object orientation modelling.

The quality of the software development process depends on the quality software products. Cimpan and Oquendo (2003) used fuzzy logic approach for monitoring
software processes deviation. They implemented the online monitoring environment: general and adaptable (OMEGA) concept to monitor the models. The OMEGA monitoring definition language (MDL) used to define monitoring models as well as mechanism for the execution of such models. The executing monitoring model observed the processes and detected deviation between it and an expected behaviour. They applied fuzzy logic-based monitoring which allowed establishing the level of conformance between performed process and process model for different aspects of the process like progress, cost, structure (order between activities), etc. The fuzzy logic enabled to handle imprecise and uncertain as well as precise and certain information. The fuzzy set theory used in the information representation, while the possibility theory used in the reasoning input imprecise information. They also used conformance factor to measure the similarity and expressed to which extent the observed process behaved similarly to what expected. They used process for process monitoring (P2M) for monitoring the software process. P2M consists of three phases viz. definition, instantiation and enactment. The definition phase constructed to monitor the model, the instantiated phase is related to context of use. The enactment phase entailed the actual monitoring of a given software process. The fuzzy logic used to model the imprecise and uncertain aspects of data and rules. They used fuzzy set theory to represent uncertain and imprecise information.

Moreover, Xu and Khoshgoftaar (2004) presented an innovative fuzzy identification cost estimation modelling technique to deal with the linguistic data, and automatically generated fuzzy membership function and rules. A case study based on the COCOMO 81 database compared the proposed model with all three COCOMO models, that is, basic, intermediate and detailed. It was observed that the fuzzy identification model provided significantly better-cost estimation than the three COCOMO models. In a recent study, Ahmed et al. (2005) presented an adaptive fuzzy logic framework for software effort prediction. The training and adaptation algorithm implemented in the framework tolerated imprecision, explained prediction ordinals through rules, incorporated expert’s knowledge and offered transparency in the prediction system. The experiment was carried out on artificial datasets as well as the COCOMO database. From the experiment, they recorded improvement in the performance of their training procedure in the presence of counter-intuitive rule base and reduced RMSRE values during validation.

Software estimation is a challenging activity as no model can predict exactly, thereby leading to cost and schedule overrun. Bajaj et al. (2006) used fuzzy set theory to extend the bottom up approach to achieve better precision in the estimates. A bottom up approach was advised where a project is completely novel or there is no historical data available. After the high level design, for each component the estimates, the effort necessary to create the detailed design, code and unit testing were determined. Each team member had the responsibility of estimating the factors for assigned components, subcomponents that estimates the complexity and effort involved. They suggested a bottom up approach, based on estimating factors and factor counts. Each team member was responsible to estimate the effort, complexity for its own components or module. Thereafter, Engel and Last (2007) proposed a set of quantitative probabilistic model for estimating costs and risks stemming from carrying out any given VVT strategy. They demonstrated that quality costs in software-intensive projects were likely to consume as much as 60% of the development project. They suggested a new approach for fuzzy modelling of VVT by introducing the concept of a canonical VVT model to describe system process and risks cost.
3.4 Genetic algorithm

GA proposed by Goldberg (1989, 1993) is a stochastic optimisation algorithm loosely based on concept of biological evolutionary theory. The application of GA’s to solve combinatorial problems may require the development of new coding/representation schemes and fitness evaluation techniques to reflect the problem or it may involve the use of conventional coding schemes and currently available operators.

Burgess and Lefley (2001) evaluated critically the potential of genetic programming (GP) to improve the accuracy of software estimation. They compared GP with various statistical and ML techniques in terms of accuracy to reduce inaccurate cost estimation, misleading tendering bids and disabling the monitoring progress. They used Desharnai datasets of 81 software projects derived from Canadian software. The input variables were restricted to those available from the specification stage and significant effort was put into GP and comparison was made to offer a realistic and fair solution. GA-based software integrations investigate an approach of integrating software with a minimum risk using GA. This problem was initially proposed by the need of sharing common software components among various departments within the same organisation.

Further, Huang et al. (2008) investigated the effect of estimation accuracy of the appropriate weighted similarity measures of effort drivers in analogy-based software effort estimation method. Three weighted analogy methods, namely, the unequally weighted, the linearly weighted and the non-linearly weighted methods were investigated. They used dataset obtained from the international software bench marking standards group (ISBSG) repository and the IBM DP services database. The experimental results showed that GA used to determine a suitable weighted similarity measure of software effort estimation model was a feasible approach to improve the accuracy of software effort estimates. Huang et al. (2007b) examined the potentials of software effort estimation model by integrating a GA to the gray rational analysis (GRA). The GA method was adopted to find the best fit of weights for each software effort driver in the similarity measures. The experiment indicated that GA obtained more precise estimates than CBR, CART and MLP.

Minohara and Tohma (1995) applied GA to the parameter estimation of the hyper-geometric distribution software reliability growth model (HGDM). They used pseudo datasets for the estimation. The experimental result compared with that of SRGM, calculus-based method imposed restriction on evaluation function making the estimation but HGDM method was more effective in parameter estimation and removes restrictions from SRGM.

Costa et al. (2005) explored GP as an alternative approach to derive SRGM. GP based on the idea of GAs and was acknowledged as a very suitable technique for regression problems. The main motivation to choose GP for this task was its capability of learning from historical data.

Consequently, Briand et al. (2002) implemented the GA to devise integration test orders in object oriented systems to reduce integration testing. They performed a case study on a system that contained 21 classes and a large number of dependency cycle. They investigated that the use of coupling measurement in the cost function (OCplx) and compare the OCplx values and the number of broken dependencies used as a cost function, attribute and method coupling were used in turn as cost function. They concluded from the case studies that using coupling measurement might lead to significantly different test orders, involving less coupling to be broken and leading to
lower integration test expenditures. The software application integration can be categorised into database integration, operating environment integration, user interface integration and so on. Yang et al. (2006) formulated the software integration problem as a search problem and solved by using GA. They selected Derbyshire Fire and Rescue Services (DFRS) as case study.

Jiang (2006) approached two major concerns of the problem in SE domain using GA. Firstly, he was trying to distinguish the problems that were difficult for GA to solve from the ones that are genetic solvable. Secondly, he identified and classified the problems in the SE domain and provides the definitions of the near optimal solutions. Further, Dai et al. (2003) presented a GA for testing resource allocation problem that can be used when the software system structure was complex, and there were multiple objects. They consider the system reliability and testing cost in the testing resources allocation problems. They developed GA to solve the complex problem of general parallel-series modular software system with multiple objectives.

3.5 Case-based reasoning

CBR (Kolodner, 1992, 1993) is relatively a simple concept. It involves matching the current problem against ones that have already been encountered in the past in the current context. It can be represented as a cyclical process, that is, divided into four-sub process is presented as by Aamodt and Plaza (1994)

1. retrieve the most similar case or cases from the case base
2. reuse the case to solve the problem
3. revise the proposed solution if necessary
4. retain the solution for future solving.

Therefore, CBR is a problem solving technique based on the reuse of past experiences. For this reason, there is considerable optimism about its use in difficult problem solving areas where human expertise is evidently experience based. Delany and Cunningham (2000) assessed the applicability of CBR to the difficult problem of early software project cost estimation. They suggested that the objective should be risk assessment rather than cost estimation because case representation was not available early in the project. They identified and presented a case representation capturing the availability prediction features for early estimation. They proposed a measure called the productivity coefficient, which gives a measure of the potential risk, revealed by the characteristics of a project compared with previous experience. They implemented CBR for software cost estimation early in the project life cycle. They showed the possibility of case representation that could be drawn from wider literature on cost estimation. It is impossible to identify features early in the life cycle that defined the size of project. Instead they proposed the alternative to pursue a solution that produced an estimate of potential risk for a specific project based on previous experiences.

Accurate software cost estimation is a vital task that affects the firm’s software investment decision before committing required resources to that project or bidding for projects. Sakran (2006) proposed an improved (CBR) approach integrated with multiagent technology to retrieve similar projects from multiorganisational distributed
datasets. He explored the possibility of building a software cost estimation model by collecting software cost data from distributed predefined project cost database.

3.6 Decision tree

Solutions to the problem of learning from examples will have far-reaching benefits, and therefore, the problem is one of the most widely studied in the field of ML (Selby and Porter, 1988). The approach based on decision trees is to provide insights through in-depth empirical characterisation and evaluation of decision trees for one problem domain, software resource data analysis. The purpose of the decision trees is to identify classes of objects (software modules) that had ‘high’ development effort or faults where ‘high’ was defined to be in the uppermost quartile relative to past data. The analysis focuses on the characterisation and evaluation of decision tree accuracy complexity and composition. The decision tree correctly identified 79.3% of the software modules that had high development effort or faults, on the average across all 960 trees. The decision trees generated from best parameter conditions correctly identified 88.4% of the modules on the average.

Selby and Porter (1988) presented the analysis of automatic decision tree generation. The purpose of decision tree was to identify certain classes of objects, such as software modules that are likely to be fault-prone or costly to develop.

Kokol et al. (2001) presented the use of evolutionary decision trees as a fault predictive approach. They used different types of software complexity metrics together with fractal software metric as the attributes for learning evolutionary decision trees. They reported that the \( \hat{a} \) metric as an attribute with other software complexity measures could be successfully used to induce decision trees for predicting dangerous modules. Redesigning such modules or devoting more testing or maintenance effort to them can largely enhance the quality and reliability.

Khoshgoftaar et al. (2000) employed CART algorithm to evaluate the software quality models over several releases. They used very large legacy telecommunication system to develop two classification trees for software quality. Both modules yielded better accuracy that would be useful to developers into various software development practices.

3.7 Soft computing

The paradigm of soft computing or computational intelligence refers to the seamless integration of different, seemingly unrelated, intelligent technologies such as fuzzy logic, NNs, GAs, ML (CBR and decision trees subsumed), rough set theory and probabilistic reasoning in various permutations and combinations to exploit their strengths.

Baisch and Liedtke (1997) compared the multilinear discriminant analysis and fuzzy expert system generated by GA for software quality prediction. They used complexity metrics to predict the quality of software development project. They applied prediction techniques on Alcatel 1000 S12 telecommunication systems. They classify the dataset into training data and test data. They used multilinear discriminant analysis in the training data to produce the change class and they applied genetic optimiser to extract fuzzy expert system for change prediction. They performed fuzzy expert system techniques for constructing quality-based productivity prediction model that identified the software
components, which posed to software potential quality problem. From the experiment they found that MDLA approach was better as compared to fuzzy ruled base.

Gray (1999) presented a simulation-based study on empirical modelling techniques using a size and effort software metric dataset. He accessed the prediction performances on hold out samples using sampling with replacement and without replacement. He applied different techniques such as MLP, regression tree, linear least-median square robust regression (LMS), linear least quantile square robust regression (LQS), linear least-trimmed squared robust regression (LTS) and M-estimator robust regression on datasets. He observed from the experiment that by comparing the different errors viz. RMSE, MMRE and pred (25), M-estimation model outperformed all other techniques.

Further, Chen and Rine (2003) employed GA, NN algorithm, Monte Carlo algorithm and their combination to improve the effectiveness and efficiency of automated maintenance of reusable components. To build a relationship between component abstract and concrete levels, each control software component is represented at the abstract level by means of a set of adaptive fuzzy membership functions. The testing phase used for identifying faulty fuzzy elements of a component while adopting phase used for modifying the membership function. This approach demonstrated by maintaining an automotive cruise controller software component. The next step was to apply train attitude control software components of a spacecraft with lab simulation. The experimental results showed that the off line training approval supports controller software components adaptation effectively and efficiently in terms of control process operation accuracy and effort spent.

Bellettins et al. (2007) addressed a set of related but distinct computing paradigms that included fuzzy logic, granular commuting, NNs and evolutionary computation. A common target of these paradigms was making human activities more tolerant of imprecision in uncertainty and partial truth. They illustrated the impact of soft-computing technique on SE research and practices. Specifically, they showed how SE task such as reuse-oriented classification (e.g. bug detection and correction effort prediction) project cost and time estimation planning and others can be handled by soft computing. Recently, Rajkiran and Ravi (2007a) developed an ensemble model, which accurately forecast software reliability. They used MLR, MARS, DENFIS and TreeNet to develop the ensemble. They designed and tested three linear and one non-linear ensemble. They concluded that the non-linear ensemble outperformed all other ensembles and also the constituent’s statistical and intelligent techniques. Mertoguno et al. (1996) used a method of designing and modelling of a neuro expert systems for prediction of software metrics. The neuro-expert system consists of two NN and expert systems with fuzzy reasoning in order to achieve a better evaluation of software metrics. They also described an approach using a MLP and fuzzy ruled-based systems for assisting in the analysis and application of the software metrics information. They described an approach to process the USA army software testing and evaluation program (STEP) software metrics. The STEP neural network interface (SNNI) used past project STEP metric datasets to train an MLR to predict final software projects status of an uncompleted project. The expert system (rule-based), which was an extension of SNNI, gives a second prediction and an interpretation of the metrics.

Various conventional model-based methods have limited success, whereas intelligent prediction using neuro-computing has proven its worth in many diverse applications. Shukla (2000) presented a novel genetically trained NN trained on historical data. He demonstrated substantial improvement in accuracy by the neuro-genetic approach as
compared to regression tree-based conventional approach as well as MLR. He used $n$-fold cross validation on various portions of merged COCOMO and Kemerer datasets incorporated from 78 real time projects. They found that genetically trained NNs (GANN) predicted significantly better predictor of software development effort than recursive partition regression (CARTx) and quick propagation trained neural networks (QPNN).

More interestingly, Huang et al. (2003a) proposed a novel neuro-fuzzy COCOMO to estimate software development effort by combining neuro-fuzzy technique with the popular COCOMO model. They used dataset obtained from COCOMO’81 database. They found that the neuro-fuzzy model yields better cost estimation accuracy than COCOMO’81 model. It could be a powerful tool to tackle important problems in SE. Further, Huang et al. (2007a) proposed neuro-fuzzy constructive model (COCOMO) to carry out the software estimation. The model dealt effectively with imprecise and uncertain input and enhanced the reliability of software cost estimation. In addition, it allowed input to have continuous rating values and linguistic values, avoiding the problem of similar projects having large different estimated costs. The validation using industry project data showed that the model greatly improves estimation accuracy in comparison with the well-known COCOMO model.

Oh et al. (2004) used a concept of self-organising neuro-fuzzy networks (SONFN). It was a hybrid architecture combining neuro-fuzzy networks (NFL) and polynomial neural networks (PNN). They used NASA datasets and medical image system (MIS) dataset for cost estimation. From the experiment, they found that SONFN predicted best accuracy compared to others. Then, Pedrycz et al. (2007) reported the analysis of software quality data using a neuro-fuzzy model. They discussed specificity of software data related to the character of neuro-fuzzy processing. They showcased how self-organising maps help reveal and visualise a structure of software data. The experimental part of study was concerned with MIS datasets available in the literature on software quality and dealing with dependencies between software complexities measured characterising software modules and the ensuing number of changes (modifications) made to them.

Aggarwal et al. (2005b) proposed NN and fuzzy logic method to carry out sensitivity analysis. They developed a model consisted of four input viz. comment ratio, average cyclomatic complexity, average number of live variables and average life span of variables. Based on case study, they built a fuzzy logic and NN model to measure software maintainability and then sensitivity analysis was carried out. From sensitivity analysis, it was found that NN Bayesian regularisation training algorithm produced reduced condition numbers as compared to the fuzzy model having membership functions whose derivative showed discontinuities at some points. They concluded that NN using Bayesian regularisation training algorithm was more stable model than fuzzy model having membership function whose derivative showed discontinuities at some points.

Most recently, Vinay Kumar et al. (2009) employed soft-computing approaches to predict software development effort. They developed linear and non-linear ensemble models to forecast software development effort. They used COCOMO 81 dataset for the experiment. They found that non-linear ensemble with radial basis function NN as the arbitrator outperformed all other techniques.
3.8 Other type of intelligent techniques

The accurate prediction of software development effort has a critical effect on all stages of the software development cycle. Underestimates of resource requirements for a software project leads to underestimation of costs, unrealistic of time schedule, considerable work pressure on the engineers.

Sheppered and Schofield (1997) described an alternative approach by using analogies. The principle was to characterise projects in terms of features (e.g. the number of interfaces, the development method or the size of the functional requirements document). Completed projects were stored and then the problem becomes one of finding the most similar projects to the one for which a prediction was required. Similarity was defined as Euclidean distance in $n$-dimensional space where $n$ was the number of project features. Each dimension was standardised, so that all dimensions have equal weight. The known effort values of the nearest neighbours to the new project were then used as the basis for the prediction. The method was validated on nine different industrial datasets (a total of 275 projects) and in all cases analogy outperformed algorithmic models based on stepwise regression. From this work, they argued that estimation by analogy was a viable technique that, which can complement current estimation techniques.

Chiu and Huang (2007) investigated the effect of analogy-based estimation on the improvement of estimation accuracy in. They reported that by applying suitable linear model to adjust the analogy-based estimation was a feasible approach to improve the accuracy of software effort estimation. The data was used for the test was from IBMMDPS organisation and from major CF organisations. Reformat et al. (2007) focused on defect related activities that are core of corrective maintenance. Two aspects of these activities were considered namely

1. a number of software components that had to be examined during defect removing process
2. time needed to remove a single defect.

Analysis of the available datasets leads to development of data models, extraction of an IF_THEN rules from these models, and construction of ensemble-based prediction systems that were built based on these data models. The data was analysed using C 5.0 and new multilevel evolutionary algorithm.

Most datasets are proprietary in natures, making it impossible to replicate in these situations. As demonstrated by the NASA experiments not all data distributions result in similar results. This work extended previous research in datasets prediction by Khoshgoftaar and Rebours (2003) conducting nearest-neighbour analysis for gaining deeper understanding of how datasets relate to each other, and thus the need for developing more realistic empirical/ML-based models.

Though, evolutionary testing has performed well in many testing areas, there are still several open problems that need further research. The use of ML algorithm had proven to be of great practical value in solving a variety of SE problems including software predictions, cost and defect processes. Twala et al. (2007) experimented on the rule inductions (RI) providing some background and key issues on RI and further examined how RI is utilised to handle uncertainties in data. Application of RI in prediction and other software tasks was considered. Supports Vector Machines (SVMs) are employed to solve non-linear regression and time series problem. Pai and Hong (2006) investigated
the feasibility of use of SVMs to forecast software reliability. Simulated annealing (SA) algorithm was used to select the parameters of an SVM model. They used two numerical examples to demonstrate the forecasting performance of SVMs model, the first example taken as a rolling-based forecasting procedure and a one-step-ahead policy is applied to forecasting software reliability. In the second example, the software failure data obtained from Musa et al. (1987), Musa (1998) and Singpurwalla and Soyer (1985) are adopted to investigate the performance of the proposed model in forecasting software reliability.

4 Discussion

From Table 1, we can draw the following inferences. MLP being at the forefront of data-driven techniques has found a huge number of applications (approximately 60% of this papers reviewed here involved MLP) in SE. Fuzzy logic-based techniques, which are knowledge-driven techniques, constitute the second highest number. Then, GAs (subsuming GP) and soft-computing hybrids (neuro-fuzzy, neuro-genetic, etc.) have 12 and 13 papers, respectively. It is followed by CBR with four papers and decision tree with six papers. It is rather surprising that decision tree found so few papers, even though their potential is quite high and they are applied in many diverse disciplines. It is an encouraging trend that soft-computing hybrids slowly but steadily found acceptance among the SE researchers.

After going through all the research work, that is, reviewed in this paper, we make the following observations.

- Since many areas of SE namely software cost/development effort estimation, software reliability forecasting, etc. are highly empirical in nature, data-driven techniques are extremely useful in solving these problems. The most important areas of SE problems that are of practical significance are software development/cost estimation, software reliability prediction, etc. These problems can be solved by utilising the data from past software projects. It is clear that all the data-driven intelligent techniques viz. NN, decision trees, etc. can be used effectively to solve these problems. This can be corroborated from this review.

- Further, we note that the data-driven nature of SE problems is common to other engineering disciplines as well. However, the most striking feature of the SE problems is its heavy dependence on human experts, knowledge and experience of past software projects. As a result, the conspicuous presence of subjectivity and empiricism in SE discipline could not be neglected. Therefore, this call for the application of knowledge-driven technology like fuzzy logic to the SE problems. Even though, we found that a few papers reviewed here dealt with the application of fuzzy logic to SE problems, there is still a lot of scope to apply FL-based method in several ways.

- In view of foregoing discussion, we make the following remarks. The peculiar nature of SE problems is that they are both data-driven and knowledge-driven in different proportions. This curious mix indeed calls for the application of more and more soft-computing hybrids to the SE problems.

- One of the significant observations is that rough set theory by Pawlak (1982) is conspicuous by its absence in the present review, even though it found interesting
The application of intelligent and soft-computing techniques

applications in many diverse areas of science and engineering. The advantage of rough set-based techniques is that feature selection, classification/forecasting and generation of ‘if-then’ rules – all can be done in one go.

- SE is always a complicated domain, which is characterised by a large number of competing and inter-related constraints (Zhang and Tsai, 2007). With the rapid development of software scale and complexity, more and more SE problems, such as initial planning, requirements analysis, cost estimation, system integration, software testing, system maintenance and so on, become vague and hard to solve, which makes it more difficult to balance these problems in the limited budget and schedule. Since traditional methods may not meet the continuous increasing complexity, promising techniques based on artificial intelligence have been proposed and researched. Nowadays, many SE problems have been considered from a new perspective as search problems, where metaheuristics can be applied. Metaheuristic search techniques refer to GAs, SA, tabu search, differential evolution, ant colony optimisation and particle swarm optimisation, which search intelligently for optimal or near optimal solutions to a problem within the search space. ML methods have been utilised to develop better software products, to be part of software products, and to make software development process more efficient and effective (He and Yu, 2007; Zhang and Tsai, 2007).

- Software testing (Zhang and Tsai, 2007) is one of the significant components of SE with many complex and inter-related constraints. Efficient testing requires systematic and automatic test data generation to satisfy pre-defined standards. However, because of the increasing complexity of software systems, usual testing techniques have demonstrated their limitation in certain areas. There are several types of testing viz. functional testing, structural testing, safety testing and random testing and so on. Test data generation can be transformed into search problem in which metaheuristic techniques can be applied. Since the search space of software testing is usually large, non-linear, and discontinuous, local search-based methods, such as hill climbing, are inefficient to find good solutions. However, GAs are a kind of global search-base strategy, which has been proved suitable for software testing. Evolutionary testing is a promising testing technique, which utilises GAs to generate test data for various testing objectives. This new testing technique is regarded as complementary and supplementary to existing approaches (Zhang and Tsai, 2007).

- Further, this review provides the merits and demerits of the intelligent techniques covered here (see Table 3) and the source of data sets (see Table 4), wherever available. This kind of information would be immensely useful to beginners to the SE area, who pursue research in this direction as to which technique should be preferred in a given circumstance.
Table 4  Description of different methods and datasets

<table>
<thead>
<tr>
<th>Software engineering area</th>
<th>Types of intelligent technique applied</th>
<th>Methods applied</th>
<th>Datasets</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software reliability</td>
<td>Neural networks</td>
<td>BPNN</td>
<td>NASA</td>
<td>Adnan et al. (2000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FP, Bayesian regularisation</td>
<td>ISBSG Release 9</td>
<td>Aggarwal (2005a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Matlab, BPNN</td>
<td>Musa (1979)</td>
<td>Cai et al. (2001)</td>
</tr>
<tr>
<td></td>
<td>MLPN, RN</td>
<td>SYS1, SYS2, SYS 3 (Lyu, 96)</td>
<td>Dohi et al. (1999)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FP, MLP, LOC/RBF</td>
<td>IBM DP services, Kemmerer Hall mark datasets</td>
<td>Guo and Lyu (2004)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BPNN, SOM, SONN</td>
<td>Intermediate level COCOMO</td>
<td>Hsu and Tenorio (1991)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Elman-recurrent NN</td>
<td>Software failure data from complex military computer systems</td>
<td>Ho et al. (2003)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Discriminant analysis, ENN</td>
<td>Large telecommunication systems</td>
<td>Houchman and Hudepohl (1997)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OLS regression method</td>
<td>ISBSG database</td>
<td>Huang et al. (2007a,b)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BPNN, PNN</td>
<td>Developed by Pondichery Engineering College, India</td>
<td>Kanamani et al. (2007)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCA</td>
<td>Large commercial system</td>
<td>Khoshgoftaar and Szabo (1994)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCA</td>
<td>Multivariate datasets (Dhilon and Goldstein, 1984)</td>
<td>Khoshgoftaar and Szabo (1996)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NN</td>
<td>Ada development environment for the command, control of a military data link communication system (CCCS)</td>
<td>Khoshgoftaar et al. (1992)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BPNN, PCA</td>
<td>Very large telecommunication software</td>
<td>Khoshgoftaar et al. (1997)</td>
<td></td>
</tr>
</tbody>
</table>
Table 4  Description of different methods and datasets (continued)

<table>
<thead>
<tr>
<th>Software engineering area</th>
<th>Types of intelligent technique applied</th>
<th>Methods applied</th>
<th>Datasets</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SVM, Bayesian statistical model</strong></td>
<td></td>
<td>Musa (1979) , Singpurwalia and Soyer (1985)</td>
<td></td>
<td>Pai and Hong (2006)</td>
</tr>
<tr>
<td><strong>Ward Net, GRNN</strong></td>
<td></td>
<td>Warehouse management application</td>
<td></td>
<td>Quah and Thwin (2004)</td>
</tr>
<tr>
<td><strong>BPNN, TANN, PSN, MARS, GRNN, MLR, TreeNets, DENFIS, Morelet-based WNN, Gaussian-based WNN</strong></td>
<td></td>
<td>Musa (1979)</td>
<td></td>
<td>Rajkiran and Ravi (2007b)</td>
</tr>
<tr>
<td><strong>BPNN, TANN, PSN, MARS, GRNN, MLR, TreeNets, DENFIS, Morlet-based TAWNN, Gaussian-based TAWNN, Morelet-based WNN, Gaussian-based WNN</strong></td>
<td></td>
<td>Musa (1979)</td>
<td></td>
<td>Ravi et al. (2009)</td>
</tr>
<tr>
<td><strong>BPNN, Goel–Okumoto model, Yamada delay S-shaped, inflection S-shaped model</strong></td>
<td></td>
<td>QUES (Wei and Henery, 1993)</td>
<td></td>
<td>Thwin and Quah (2005)</td>
</tr>
<tr>
<td><strong>Ward neural network, GRNN</strong></td>
<td></td>
<td>Real time and flight application Musa et al. (1987)</td>
<td></td>
<td>Tian and Noore (2005a,b)</td>
</tr>
<tr>
<td><strong>FL based</strong></td>
<td>FL based</td>
<td>COCOMO</td>
<td></td>
<td>Ahmed et al. (2005)</td>
</tr>
<tr>
<td><strong>GA, BPNN, Monte Carlo algorithm and their combination</strong></td>
<td></td>
<td>Automotive controller software</td>
<td></td>
<td>Chen and Rine (2003)</td>
</tr>
<tr>
<td><strong>FL based</strong></td>
<td><strong>CART</strong></td>
<td>Inspection data (Ebenau)</td>
<td></td>
<td>So et al. (2002)</td>
</tr>
<tr>
<td><strong>Qiuilan (ID3)</strong></td>
<td></td>
<td>Large legacy telecommunication system NASA production environment</td>
<td></td>
<td>Khoshgoftaar et al. (2000)</td>
</tr>
<tr>
<td><strong>GA, MLP</strong></td>
<td></td>
<td>Desharnais</td>
<td></td>
<td>Selby and Porter (1988)</td>
</tr>
<tr>
<td><strong>GP</strong></td>
<td></td>
<td>Musa (1980)</td>
<td></td>
<td>Burgess and Lefley (2001)</td>
</tr>
<tr>
<td><strong>GA</strong></td>
<td></td>
<td>Parallel-series modular software system COCOMO, ALBRECHT</td>
<td></td>
<td>Costa et al. (2005)</td>
</tr>
<tr>
<td><strong>GRA, CBR, CART, ANN</strong></td>
<td></td>
<td>Pseudo dataset</td>
<td></td>
<td>Dai et al. (2003)</td>
</tr>
<tr>
<td><strong>GA using hypergeometric distribution</strong></td>
<td></td>
<td>Pseudo dataset</td>
<td></td>
<td>Huang et al. (2007a,b)</td>
</tr>
<tr>
<td><strong>Fuzzy logic</strong></td>
<td></td>
<td>Pseudo dataset</td>
<td></td>
<td>Minohara and Tohma (1995)</td>
</tr>
</tbody>
</table>
Table 4 Description of different methods and datasets (continued)

<table>
<thead>
<tr>
<th>Software engineering area</th>
<th>Types of intelligent technique applied</th>
<th>Methods applied</th>
<th>Datasets</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX-based approach</td>
<td>Multilinear discriminant analysis, fuzzy rule based</td>
<td>Derbyshire fire and rescue services (DFRS)</td>
<td>Yang et al. (2006)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GA, Euclidean distance (AE), Manhattan distance (AMH), Minikowski distance (AMK)</td>
<td>Telecommunication projects</td>
<td>Baisch and Liedtke (1997)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NEURO-FUZZY</td>
<td>IBM data processing services (DPS), Canadian financial (CF)</td>
<td>Chiu and Huang (2007)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SONN, neuro-fuzzy network, polynomial neural network</td>
<td>US army software testing and evaluation program (STEP) metric data</td>
<td>Mertoguno et al. (1996)</td>
<td></td>
</tr>
<tr>
<td>Software cost estimation</td>
<td>Neural networks</td>
<td>Logical log approach, curve fitting approach</td>
<td>COCOMO II, 2000</td>
<td>Dillibabu and krishnaiah (2005)</td>
</tr>
<tr>
<td></td>
<td>MLP, regression trees, M-estimation</td>
<td>COCOMO-II</td>
<td>Gray (1999)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MLP, RBFN</td>
<td>IBM DP services, Kemerer dataset, Hall mark dataset</td>
<td>Heiat (2002)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BPNN, SONN</td>
<td>Intermediate level of COCOMO</td>
<td>Hsu and Tenorio (1991)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RBFN, Jang and Sun</td>
<td>COCOMO-81</td>
<td>Idri et al. (2002)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RBFN, BPNN, Jang and Sun</td>
<td>COCOMO-81</td>
<td>Idri et al. (2004)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BPNN</td>
<td>Korea electric power corporation, Samsung electronics datasets</td>
<td>Jun and Lee (2001)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCA</td>
<td>Kernel</td>
<td>Khoshgoftaar and Szabo (1996)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NN, multiple regression</td>
<td>COCOMO database (Boehm, 1981)</td>
<td>Tronto et al. (2007)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BPNN</td>
<td>COCOMO</td>
<td>Venkatachalam (1993)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Regression analysis, ANN</td>
<td>COCOMO 81, SLIM</td>
<td>Vinay Kumar et al. (2008)</td>
<td></td>
</tr>
</tbody>
</table>
Table 4  Description of different methods and datasets (continued)

<table>
<thead>
<tr>
<th>Software engineering area</th>
<th>Types of intelligent technique applied</th>
<th>Methods applied</th>
<th>Datasets</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BPNN</td>
<td></td>
<td>Desharnais</td>
<td>Wittig and Fennie (1997)</td>
</tr>
<tr>
<td>Soft computing</td>
<td>Fuzzy subtractive clustering</td>
<td>BPNN, TANN, PSN, MARS, GRNN, MLR, TreeNet, DENFIS</td>
<td>Legacy telecommunication system</td>
<td>Yuan and Ganesan (2000)</td>
</tr>
<tr>
<td></td>
<td>Neuro-fuzzy</td>
<td></td>
<td>MUSA (79)</td>
<td>Rajkiran and Ravi (2007b)</td>
</tr>
<tr>
<td></td>
<td>Neuro-fuzzy</td>
<td></td>
<td>ISBSG</td>
<td>Aggarwal et al. (2005a)</td>
</tr>
<tr>
<td></td>
<td>BPNN, GA</td>
<td></td>
<td>COCOMO-81</td>
<td>Huang et al. (2003a, 2007a)</td>
</tr>
<tr>
<td></td>
<td>BPNN</td>
<td></td>
<td>COCOMO-II</td>
<td>Shukla (2000)</td>
</tr>
<tr>
<td></td>
<td>MLP, RBF, MLR, DENFIS</td>
<td></td>
<td>COCOMO-81</td>
<td>Tadayon (2004)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Vinay Kumar et al. (2009)</td>
</tr>
</tbody>
</table>

As far as the future directions are concerned, we observe that

1. Decision trees can be applied more effectively and more often to solve SE problems such as software defects identification, software development cost estimation and software reliability prediction.

2. Fuzzy rule-based systems could be applied more often and exploited more effectively. As a consequence, we end up with either crisp or fuzzy expert systems involving human comprehensible ‘if-then’ rules.

3. Further, there are other NN techniques such as group method of data handling (GMDH), counter propagation neural networks (CPNN), Bayesian neural networks, probabilistic NNs that can be applied to solve many SE problems involving classification and forecasting tasks such as software defects identification, software development cost estimation and software reliability prediction, etc.

4. Rough set theory-based methods could also be applied with great success to solve classification problems in SE.

5. New soft-computing hybrids could be developed involving differential evolution, ant colony optimisation, NNs to solve problems such as software defects identification, software development cost estimation and software reliability prediction, etc. Further, the effectiveness of principal component NN, DEWNN could also be investigated.

6. Given the data dependent nature and performance of the metaheuristics, other metaheuristics such as differential evolution, ant colony optimisation, particle swarm optimisation could be attempted as an alternative to GA, in order to come out with better test data generation, which would be eventually useful in software testing.

7. Descriptive data analysis techniques such as data envelopment analysis, analytic hierarchy process, analytic network process, fuzzy multiattribute decision making
could be employed to rank software projects developed in the same domain in terms of quality attributes.

5 Conclusions

A comprehensive review encompassing the application of intelligent techniques such as NN, fuzzy logic, GA, decision trees, CBR and soft-computing hybrid to ESE is presented. This review is written by taking the technique employed as the important dimension. The source of the data set and the results whenever available are also provided in the review. This review indicates that out of all the machine intelligent techniques NNs was the most preferred method by the SE community to solve problems like software development cost estimation, software reliability prediction, etc. This review is expected to act as a starting point for any researcher who wishes to conduct research in ESE employing sophisticated techniques. For practitioners also, the review would provide useful insights into the popularity, merits and demerits of various intelligent techniques. Of particular significance is that the present review provides immensely useful future directions, which by themselves open up new areas of research in the entire gamut of SE at the interface of intelligent techniques and soft computing. This would eventually lead to better decision making in SE thereby ensuring better, more reliable and cost effective software products.

Acknowledgement

The authors thank the anonymous reviewers for their useful suggestions, which greatly improved the presentation of this paper.

References

The application of intelligent and soft-computing techniques


The application of intelligent and soft-computing techniques


