*A Survey on applications of machine learning/Artificial intelligence in software engineering*

|  |
| --- |
| Rashid Ali |
| *Fast School of Computing,* |
| *National University of Computer and Emerging Sciences,* |
| *Punjab, Lahore 54700, Pakistan* |
| l165035@lhr.nu.edu.pk |

**Abstract**— Machine learning is practical for software engineering problems, even in data starved domains. When data is scarce, Knowledge can be farmed from seeds; i.e. minimal and partial descriptions of a domain. These seeds can be grown into large datasets via Monte Carlo simulations. The datasets can then be harvested using machine learning techniques. Examples of this knowledge farming approach, and the associated technique of data-mining, is given from numerous software engineering domains.

Machine learning techniques such as knowledge based systems, neural networks, fuzzy logic and data mining have been advocated by many researchers and developers as the way to improve many of the software development activities. As with many other disciplines, software development quality improves with the experience, knowledge of the developers, past projects and expertise. Software also evolves as it operates in changing and volatile environments. Hence, there is significant potential for using machine learning for improving all phases of the software development life cycle. This chapter provides a survey on the use of machine learning for software engineering that covers the main software development phases and machine learning methods such as natural language processing techniques, neural networks, genetic algorithms, fuzzy logic, ant colony optimization, and planning methods.

Keywords – Machine learning, computational intelligence, Software Engineering,

# **Introduction**

Machine learning is an important research direction in Articial Intelligence. Machine learning explores the construction and study of algorithms that can learn from data Machine learning is the study of systems models that, based on a set of data (training data), improve their performance by experiences and by learning some specific domain knowledge.

The history of the field of Artificial Intelligence (AI) is long and illustrious, tracing its roots back to the seminal work of Turing [1] and McCarthy [2]. The idea that machines can be intelligent has provided a staple diet for science fiction. Despite this, AI can also seem rather commonplace: Computational intelligence regularly provides examples of specific areas of intelligent behaviour for which machines comfortably surpass the performance of even the best human. Right from its intellectual origins in the 1950s the field stimulated philosophical as well as technological debate and raised much interest, not to mention a little concern, from the wider public. Software engineers, by contrast, are less used to seeing their work in the science fiction literature. They are typically focused on more prosaic and practical engineering concerns. Nevertheless, the software engineering research and practitioner communities have fallen under the ‘AI spell’. Artificial Intelligence is about making machines intelligent, while software engineering is the activity of defining, designing and deploying some of the most complex and challenging systems mankind has ever sought to engineer. Though software engineering is one of the most challenging of all engineering disciplines, it is often not recognised as such, because software is so well concealed.

There are three basic machine learning strategies:

* Supervised learning (the learning strategy that is supervised by a teacher)
* Unsupervised learning (learning without supervision)
* Reinforcement learning (learning by interaction with the environment).

***Machine learning techniques***

1. **Decision tree learning**

Decision tree learning uses a decision tree as a predictive model, which maps observations about an item to conclusions about the item's target value [3].

1. **Association rule learning**

Association rule learning is a method for discovering interesting relations between variables in large databases.

1. **Artificial neural networks**

An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is inspired by the structure and functional aspects of biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.

1. **Support vector machines**

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model thatpredicts whether a new example falls into one category or the other.

1. **Clustering (Unsupervised classification)**

Cluster analysis is the assignment of a set of observations into subsets (called clusters) so that observations within the same cluster are similar according to some predesignated criterion or criteria, while observations drawn from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data, often defined by some similarity metric and evaluated for example by internal compactness (similarity between members of the same cluster) and separation between different clusters. Other methods are based on estimated density and graph connectivity. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis.

1. **Reinforcement learning**

Reinforcement learning is concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. Reinforcement learning algorithms attempt to find a policy that maps states of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected

***Software engineering [5]***

Design, development, and maintenance of software. Application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.

## **Software development life cycle:**

* Software development life cycle:
* Requirement gathering and analysis
* Design
* Implementation or coding
* Testing
* Deployment
* Maintenance

## **Common software development metodologies:**

* waterfall
* prototyping
* Iterative and incremental development
* Feature driven development
* Spiral development
* Rapid application development
* Extreme programming
* Agile methodology

## **Software Engineering subdisciplines**

* **Requirements engineering**

The elicitation, analysis, specification, and validation of requirements for software.

* **Software design**

The process of defining the architecture, components, interfaces, and other characteristics of a system or component. It is also defined as the result of that process.

* **Software construction**

The detailed creation of working, meaningful software through a combination of coding, verification, unit testing, integration testing, and debugging.

* **Software testing**

An empirical, technical investigation conducted to provide stakeholders with information about the quality of the product or service under test.

* **Software maintenance**

The totality of activities required to provide cost-effective support to software.

* **Software configuration management**

The identification of the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the system life cycle.

* **Software engineering management**

The application of management activities planning, coordinating, measuring, monitoring, controlling, and reporting to ensure that the development and maintenance of software is systematic, disciplined, and quantified.

* **Software engineering process**

The definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle process itself.

* **Software engineering tools and methods**

The computer based tools that are intended to assist the software life cycle processes (Computer-aided software engineering) and the methods which impose structure on the software engineering activity with the goal of making the activity systematic and ultimately more likely to be successful.

* **Software quality management**

The degree to which a set of inherent characteristics fulfills requirements.

## **Software engineering activities**

* Requirements analysis
* cost/time estimation
* treacability
* Design
* decompose system into subsystems
* components, classes, state machines
* Coding
* Implement new functionalities
* Repair bugs
* Refactoring
* Source code repository
* Continuous integration
* Testing
* create test cases/unit tests
* integration testing
* system testing
* Deployment
* configuration management
* application monitoring
* Maintenance
* program comprehension
* refactoring/improving overall design
* fix bugs/create test cases
* concept location
* software visualization

# **LITERATURE REVIEW**

**Nicholas R Jennings [6]** analyzes how agent based software contributes in solving real world problems. The paper argues that even though usual methodologies for software testing are fruitful in deriving conclusive results they have two main drawbacks. Firstly, the rigid interactions amongst the computational entities and secondly, the organizational structure of systems cannot be efficiently represented using the traditional methods. The proposal made in this paper centers around ‘the adequacy hypothesis’ which suggests that agent based approached can sufficiently enhance the ability to model, build and design distributed and complex systems. It also introduces ‘the establishment hypothesis’ which predicts that agent based techniques will be successful in the realm of software engineering. Presenting a qualitative approach, Jennings suggests that the techniques for dealing with complexity include decomposition; abstraction and organization are well managed by agent based systems in the following manner. Agent based systems involve decomposition by partitioning the problem space in complex systems in terms of autonomous agents thus enabling high-level interactions and reducing the control complexity, which now lies in the individual components. Decision making is resolved as each component makes a selection determined by the problem solver for each localized environment. This flexibility to take decisions at run time about the nature and scope of functionalities simplifies complex systems. The abstractions of agent based approach serve to solve problems in complex systems naturally and agents efficiently handle organizational relationships as they provide the flexibility to form groups in isolation, which are then added into the system in an incremental manner. The two disadvantages of agents presented are the unpredictable outcomes and patterns of interactions and the difficulty in predicting the complete behavior of the system on the basis of constituting components. Yet, agent based systems stand to increase throughput, efficiency and robustness of a system.

**David Barstow[7]** indicates that software engineering activities are knowledge intensive so the capability of agents to serve as knowledge representation systems and the ability to provide explicit representation of knowledge serve to improve software engineering costs. Knowledge in a system is utilized for decision making, communication, inference and analysis. The artificial intelligence heuristic search paradigm, knowledge representation technique and rule based agents can be used to represent software engineering programming techniques, provide expertise, store and obtain knowledge about the application domain required during specification, requirement analysis and deployment as well as keep information regarding the history of the software by recording the motivations behind various decisions made during design, requirement analysis and implementation. The disadvantage of agent based approach is their inabilities to alone solvecompletely the software engineeringproblems without the needfor additionaltechniques to serve the purposesuch as database techniques, communication techniques and user interface techniques. However, Barstow suggests that agent based techniques described above will be deemed necessary to solve software engineering issues and enhance productivity of a system provided the research and experiments problems are addressed effectively.

Witold[8] explores the role of computational intelligence in the field of software engineering and provides three regions where this intelligence can be applied. First, it is suggested that fuzzy models could be applied to software processes. It uses the concepts of neural networks and granular computing and indicates a high level of compatibility between software engineering paradigms and the foundation of granular computing. Second, it suggests that high dimensional software data can be visualized using self organizing maps in neural networks. Third, it stresses on the ability to create logic models of software quality. It introduces the idea of search based software engineering where in a metaheuristic searching algorithm could be used to find software solutions that are both feasible and possible. Witold goes on to express that computational intelligence has applications in a variety of aspects of software engineering, which involves testing, reliability and quality prediction, estimating cost, classifying software components and even creating models of software data.

**Mark Harman[9]** suggests that agent based algorithms can be employed effectively in software engineering issues with contributions in three main fields. Firstly, learning, classification and predictive methods help in the modeling and determination of software costs and model. This makes use of machine learning algorithms such as cased based reasoning, rule induction and the formulation of artificial neural networks to make project predictions and foresee defects. Secondly, effective optimization techniques and computationalsearch methodologies can be integrated into a field call Search Based Software Engineering. This field designs software engineering issues as optimization issues, which are then, tackled using computational searching algorithms. The use of this type of engineering has a variety of applications in software testing, maintenance as well as requirements analysis and design. Thirdly, fuzzy and probabilistic works such as Bayesian scheme can be adapted to software engineering to determine software reliability and fits into the real world problems, such as analysis of software users, which are generally of fuzzy nature with incomplete and noisy data. Harman expands that artificial intelligence can be incorporated into software engineering to give insight to software engineers, search for software development strategies, exploit multicore computation, compile smart optimizationinto deployment software, toadapt software products and use cases to better suit agent oriented algorithmsas well as obtain a harmonious trade off between human intervention and automation.

**F.Meziane [10]** comments that in the current scenario of growing modernization and development in the smart-technical centre, artificial intelligence techniques like knowledge based systems, fuzzy logics and neural networks and data mining techniques have been adapted my several researchers and professionals to improve software development activities. Thus there is a noteworthy potential for using AI in almost all the phases in software development life cycle. The paper stated underlines the usage and vitality of Artificial Intelligence for Software Engineering and describes the important phases of software development and methods of AI, some being neural networks, genetic algorithms, ant colony optimization and some planning techniques. As per the surveyconducted in this paper, the use of Genetic Algorithm is the most prevailing. Neural networks have been adapted for risk appraisal but Bayesian networks are more transparent and thus alluring to practice by the managers of the projects. Similar views were presented to the usage of Case based reasoning. The requirement and design phases have highlighted the importance to identify the errors that are faced in the initial stages during the development of a software. The survey has clearly suggested that though there is a remarkable progress in the applications of AI in the software engineering field yet there is a need of large scale evaluations and extensive researches to understand the coherence of different algorithms and approaches.

# **WHEN DOES AI FOR SE WORK WELL?**

The areas in which AI techniques have proved to be useful in

software engineering research and practice can be characterized ‘Probabilistic Software Engineering’, ‘Classification, Learning and Prediction for Software Engineering’ and ‘Search Based Software Engineering’. In Fuzzy and probabilistic work, the aim is to apply to Software Engineering, AI techniques developed to handle

real world problems which are, by their nature, fuzzy and

Probabilistic. There is a natural fit here because, increasingly,

Software engineering needs to cater for fuzzy, ill-defined, noisy and incomplete information, as its applications reach further into our messy, fuzzy and ill-defined lives. This is not only true of the software systems we build, but the processes by which they are built, many of which are based on estimates. One example of a probabilistic AI technique that has proved to be highly applicable in Software Engineering has been the use of Bayesian probabilistic reasoning to model software reliability[5], one of the earliest examples of the adoption of what might be called, perhaps with hindsight, ‘AI for SE’. Another example of the need for probabilistic reasoning comes from the analysis of users, inherently requiring an element of probability because of the stochastic nature of human behaviour . In classification, learning and prediction work there has been great interest in modelling and predicting software costs as part of project planning. For example a wide variety of traditional machine learning techniques such as artificial neural networks, cased based reasoning and rule induction have been used for software project prediction , ontology learning and defect prediction [11]. An overview of machine learning techniques for software engineering can be found in the work of Menzies. In Search Based Software Engineering (SBSE) work, the goal is to re-formulate software engineering problems as optimization problems that can then be attacked with computational search. This has proved to be a widely applicable and successful approach, with applications from requirements and design to maintenance and testing. Computational search has been exploited by all engineering disciplines, not just Software Engineering. However, the virtual character of software makes it an engineering material ideally suited to computational search. There

is a recent tutorial that provides a guide to SBSE .

# **Use of AI in Planning and Project Estimation**

Good project planning involves many aspects: staff need to be assigned to tasks in a way that takes account of their experience and ability, the dependencies between tasks need to be determined, times of tasks need to be estimated in a way that meets the project completion date and the project plan will inevitably need revision as it progresses. AI has been proposed for most phases of planning software development projects, including assessing feasibility, estimation of cost and resource requirements, risk assessment and scheduling. This section provides pointers to some of the proposed uses of knowledge based systems, genetic algorithms, neural networks and case based reasoning, in project planning and summarizes their effectiveness.

**Neural Networks**

Neural networks (NNs) have been widely and successfully used for problems that require classification given some predictive input features. They therefore seem ideal for situations in software engineering where one needs to predict outcomes, such as the risks associated with modules in software maintenance (Khoshgoftaar & Lanning, 1995), software risk analysis (Neumann, 2002) and for predicting faults using object oriented metrics (Thwin & Quah, 2002). The study by Hu, Chen, Rong, Mei & Xie (2006) is typical of this line of research. They first identified the key features in risk assessment based on past classifications such as those presented by Wallace and Keil (2004) and further interviews with project managers. They identified a total of 39 risk factors which they grouped into 5 risk categories: project complexity, cooperation, team work, project management, and software engineering. These were reduced to 19 linearly independent factors using principal component analysis (PCA). Projects were considered to have succeeded, partially failed, or failed. In their experiments, they tried both the use of a back propagation algorithm for training and use of GAs to learn networks, using 35 examples for training and 15 examples for testing. The accuracy they obtained using back propagation was 80% and that with a GA trained NN was over 86%, confirming that use of NNs for predicting risk is a worthy approach, though larger scale studies are needed.

**Case Based Reasoning**

It can be argued that successful project planning and management is heavily based on experience with past cases. It is therefore surprising that there are few studies that propose the use Case Based Reasoning (CBR) for project planning of software development. One of the few exceptions

is the study by Heng-Li Yang & Chen-Shu Wang (2008), who explore the combined use of CBR and data mining methods for project planning. They use a structured representation for cases, called Hierarchical Criteria Architecture (HCA), where

projects are described in terms of the customer requirements, project resources and keywords describing the domain. The use of HCA enables different weights to be adopted when matching cases, allowing greater flexibility depending on the preferences of the project manager. Given a

new project, first similar new cases are retrieved. Then, data mining methods, such as association rule mining, are used to provide further guidance in the form of popular patterns that could aid in project planning. In a trial, based on 43 projects,

Yang & Wang (2008), show how the combined use of CBR and data mining can generate useful information, such as “the duration of project implementation was about 26 days and 85% of projects of projects were completed on time”, which can be used to provide guidance when planning a similar project.

**Intelligence Computing for Requirements Engineering**

Pedrycz & Peters (1997)[12] stated that “The emerging area of Computational Intelligence (CI) provides a system developer with a unique opportunity of taking advantage of the currently developed and highly mature technologies”. They argue that each of the techniques developed in CI can play an important role in solving the traditional problems found in software engineering. In these sections we review some of the systems developed using CI techniques to support requirements engineering.

The SPECIFIER system (Miriyala & Harandi, 1991) can best be viewed as a case based system that takes as input an informal specification of an operation where the pre and post conditions are given as English sentences. The verbs in the sentences are used to identify the concepts. The identified concepts are then used to retrieve associated structure templates. These structure templates have slots that define the

expected semantic form of the concepts and have associated rules that can be used to fill in the slots by using the informal specification. A set of rules is used to select specification schemas based on the identified concepts. The specification schemas are then filled by using the rules associated with the slots and the structures of the concepts. Once filled, the specification schemas produce formal specifications in a Larch-like languageWhen dealing with conflicts in requirements, we often drop one of the requirements or modify it to avoid the conflict. However, Yen & Liu (1995) stated that it is desirable “to achieve an effective trade off among conflicting requirements so that each conflicting requirement can be satisfied to some degrees, while the total satisfaction degree is maximized”. Hence they suggested that it is necessary to identify and assess requirements priorities. In their approach they use imprecise conflicting requirements to assess requirements priorities. Users are required to relatively order requirements and to decide how much important a requirement is with regards to other conflicting

requirements. They then used fuzzy logic and possibility theory to develop an approximate reasoning schema for inferring relative priority of requirements under uncertainty.

**TESTING**

Despite the wealth of research in the last two decades, software testing remains an area where, as cases of reported failures and numerous releases of software suggest, we cannot claim to have mastered. Bertolino (2007)[13] presents a useful framework for summarising the challenges that we face in addressing the problems of ensuring that systems are fit for purpose, suggesting further research on: (i) developing a universal theory of testing, (ii) fully automatic testing, (iii) design to facilitate testing and (iv) development of integrated strategies that minimise the cost of repeated testing. This section presents some pointers to attempts at using AI techniques to support particular aspects of the testing process, which has the potential to contribute towards a more integrated dream testing environment of the

kind proposed by Bertolino (2007).

# **Conclusion**

##### Software engineering and artificial intelligence are two fields, that when combined attempt to solve problems and make easier the life of developers, testers and analyzers. This automated approach is a potential strategy that could be exploited to benefit aspects of software engineering and solve the problems faced by software engineers. In this paper, we surveyed promising applications of AI techniques to the realm of software engineering.

##### **References**

1. A. M. Turing, “Computing machinery and intelligence,” Mind, vol. 49, pp. 433–460, Jan. 01 1950
2. J. McCarthy, “Programs with common sense,” in Proceedings of the Symposium on Mechanisation of Thought Processes, vol. 1. London: Her Majesty’s Stationery Office, 1958, pp. 77–84.
3. Google.com “What is machine Learning”.
4. The Role of Artificial Intelligence in Software Engineering Mark Harman CREST Centre, University College London, Malet Place, London, WC1E 6BT, UK.
5. <https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering>
6. Meziane, Farid, and Sunil Vadera. "Artificial Intelligence in Software Engineering: Current Developments and Future Prospects." In Machine Learning: Concepts, Methodologies, Tools and Applications, pp. 1215-1236, 2012.
7. Vijila, J. Maria Merceline, and R. Abinaya. "Deploying artificial intelligence techniques in software engineering."
8. Pedrycz, Witold. "Computational intelligence as an emerging paradigm of software engineering.”,14th international conference on Software engineering and knowledge engineering, pp. 7-14, 2002.
9. Harman, Mark. "The role of artificial intelligence in software engineering.", First International Workshop on Realizing AI Synergies in Software Engineering, pp. 1-6, 2012.
10. Wang, Y., “Convergence of Software Science and Computational Intelligence: A New Transdisciplinary research Field”,In Software andIntelligent Sciences: New Transdisciplinary Findings,pp. 1-13, 2012.
11. V. U. B. Challagulla, F. B. Bastani, I.-L. Yen, and R. A. Paul, “Empirical assessment of machine learning based software defect prediction techniques,” International Journal on Artificial Intelligence Tools, vol. 17,no. 2, pp. 389–400, 2008.
12. Pedrycz, W., & Peters, J. F. (1997). Computational intelligence in software engineering. Canadian Conference on Electrical and Computer Engineering, (pp. 253-256), St. Johns, Nfld., Canada. Washington DC: IEEE Press.
13. Bertolino, A. (2007). Software Testing Research: Achievements, Challenges, Dreams. In Proceedings of the IEEE International Conference on Software Engineering (pp. 85-103), Minneapolis, MN. Washington DC: IEEE Computer Society.