Journal of Theoretical and Applied Information Technology
30th November 2013. Vol. 57 No.3
© 2005 - 2013 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

USE OF ARTIFICIAL INTELLIGENCE IN AUTOMATION OF SEQUENTIAL STEPS OF SOFTWARE DEVELOPMENT / PRODUCTION

[1]SENTHIL JAYAVEL, [2]S ARUMUGAM, [3]BRIJENDRA SINGH, [4]PRASHANT PANDEY, [5]AKASH GIRI, [6]AKSHAY SHARMA

[1, 4, 5,6] Vellore Institute of Technology, School of Computer Science and Engineering Vellore-63014, India

[2] Nandha College of Technology, Erode – 638052, India.

[3] Vellore Institute of Technology, School of Information Technology and Engineering, Vellore-63014, India

E-mail: [1]senthil.j.vit@gmail.com, [2]senthil.j.vit@gmail.com, [3]bobbybisht333@gmail.com, [4]prashantsp95@gmail.com, [5] akashgiri@gmail.com

ABSTRACT

Software development is a sequential process where the allied steps in the development lifecycle involve planning and modularization, requirement engineering, analysis of product viability, profits estimation, strategic decision making, maintenance strategies etc. Often, most of these phases are pretty complex and thereby, extremely difficult to handle solely through human intervention, mainly due to the size of the project, the number of factors to be taken into consideration at each modular level and the rapidly changing external environment.

In this paper we aim to provide an intuition on using Artificial Intelligence (AI) in the different phases of the software development lifecycle. Our paper focuses on a specific software development example for clarity and precision, but most of the techniques are highly general and scalable to any software development process.

Keywords: *Software Engineering; Artificial Intelligence; Machine Learning*

1. INTRODUCTION

As a firm goes through a software development process it faces many problems. The problems at forefront include the following [1]:

1. To group the work and subsequently the work force into coherent clusters to achieve work force modularity.
2. To measure the market value of the product based upon its features and quality.
3. To select the appropriate tools for development based on our functional and non-functional requirements.
4. To ensure proper maintenance of the product.

Often, most of these problems are either, not addressed properly or, addressed manually through human intervention which makes the product vulnerable to flaws thus leading to catastrophic consequences.

Though the above list is certainly not exhaustive, it identifies the major issues in the development process which are to be addressed. This paper identifies the above problem on a selected model and proposes robust learning algorithms at each level to deal with them. The advantage that learning algorithms have over manual decision making is that, not only are they fast and robust, they also adapt with the changing environment. Besides, if the size of the project is huge, manual evaluation can be almost unfeasible and sometimes impossible, making AI the only viable option.

2. ARTIFICIAL INTELLIGENCE IN SEQUENTIAL STEPS OF SOFTWARE DEVELOPMENT

In this section we describe AI algorithms to address to above stated problems. We begin the discussion of AI techniques [4], [5], [8] for work-force modularization, followed by cost prediction, requirement engineering and finally maintenance. We have included small subsets of data used in our experiments as tables in each section. The figures shown in each section are the result of applying the respective algorithms on our dataset.

2.1 Modularizing Design and Work-Force

The early phase of software lifecycle involves modularization i.e. dividing the project into smaller modules. Naturally, this division would imply a similar division of employee work force.Both these tasks can be thought of as an unsupervised machine learning algorithm which can be accomplished by a clustering algorithm.

Consider a project scenario where we have the identified the different entities of our software and we have defined the interaction between them. A feasible way to modularize the project could be done by observing the interaction between these entities. If there exists a strong interaction between some entities, they could be put together in one single module and likewise for other entities. This division would ensure that components which are highly interrelated are developed as a single module quite independent from those which have very less interaction with others. We can apply a simple K-means clustering or a single linked clustering algorithm to bring about this modularization where the similarity measure between entities will be proportional to the degree of interaction between them.

Next, we come to work force modularization. Consider a large firm where different members of the firm have different area(s) of expertise. Let us assume that the firm maintains the record of their employee's field of expertise in the form of a 2D matrix of dimension m x n where m represents the total number of employees and n represents the total possible field of expertise. Furthermore, each entry in the matrix is a number between 0 and 1.1 represents that the employee is a complete expert in the given field and 0 represents that the employee has no knowledge in the given field. As such, each employee's skills will be an n dimensional vector. Having this data with us, a logical step would be to define a similarity measure amongst the employees and use it to form coherent clusters based on their field of expertise. For the purpose of simplicity let's assume that this similarity measure is inversely proportional to the element wise difference between two such vectors. Let's assume it as a measure of distance between the two employees. Concretely, consider that X1, X2, X3, X4 be the allied area of expertise. Say Employee1(E1) specializes in X1 and X3 while Employee2(E2) specializes in X2 and X4.Say, a third employee(E3) specializes again in X2 and X4 and a fourth employee(E4) in X1 and X3.Below are their feature vectors:

Table 1: Feature Vectors of Employees

	X1	X2	X3	X4
E1	0.95	0.3	9.7	1.2
E2	0.7	9.3	1.7	8.9
E3	0.5	8.9	0.7	9.2
E4	0.9	0.3	9.9	0.2

When we take the square element wise difference say, for E1 and E4 we get $(0.95-0.7)^2$

Table 2:Sample Dataset Of Employee's Field Of Expertise

C	Photoshop	PHP	Database Design	C++	CSS	HTML	XML	JAVA	Illustrator	JavaScript	XSLT	Pascal	Flash	Ajax	Data Analysis
8	1	2	0	7	1	0	1	9	1	0	0	10	0	0	2
1	9	2	0	3	10	0	1	3	10	1	1	0	8	0	0
0	1	8	0	0	0	9	1	1	1	10	3	1	0	8	1
1	3	2	10	1	0	2	9	0	2	0	9	0	0	1	8

+ $(0.3-9.3)^2$ + $(9.7-1.7)^2$ + $(1.2-8.9)^2$. We can clearly observe that E1 and E4 are very close together but pretty far from E2 and E3. Likewise, E2 and E3 are very close together but are very far away from E1 and E4. Now, when we apply a K-means clustering algorithm to the above, E1 and E4 will be grouped together and so would E2 and E3. This is exactly what we wanted. The algorithm groups workers with similar skills together and thereby helps to break the work force into coherent clusters. Notice that the similarity measure that we have defined here is very elementary, so as to maintain simplicity, a better measure would involve use of a more complex Gaussian similarity measure as in the case of Support Vector Machines.

As a concrete case, let's consider our example of a firm which is building a website for Hospital management system. We will use the same example in the next stages as well. We apply the above model to divide the work force in this firm. We have every employee's detail available with us. Say, our goal is to divide the work force into coherent groups of Designers, Programmers, Database Developers and Web Developers. Our features vector is an n dimensional vector consisting of all allied field of expertize viz. C , Photoshop, PHP, MySQL, Python, CSS, HTML5, XML, Java, Illustrator, JavaScript, XSLT, Pascal, Flash, jQuery, Data Analyst. There are m such vectors where m in our case represents our employee strength for the particular project. As mentioned above, each entry in this m x n matrix is a number between 0 and 1. Now, we run the K-means clustering algorithm on our dataset and we set the desired number of clusters (k) to 4. The results are shown in Figure 1. The algorithm automatically separates the Designers (experts in Photoshop, CSS, Illustratorand Flash), Core Programmers (experts in C, C++, Pythonand Java), Web programmers (experts in PHP, HTML5, JavaScript and jQuery) and Database Developers (experts in MySQL, XML, XSLT, Data Analyst). Table 2 shows a sample dataset for such a system. Figure 1 shows the possible effect of running a clustering algorithm on such a data set.

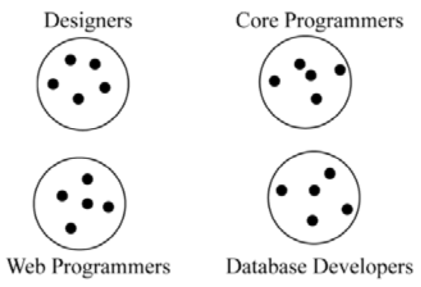

Figure 1: Effect Of Clustering Algorithm

2.1.1 Cost prediction

In this section we will deal with AI algorithms in order to predict the cost of product [3], [6]. The cost incurred in the development of a software product generally depends on many factors. Suppose we are building a website for an online Hospital management system. The cost of the system will depend on factors like number of users (a measure of traffic to the website), the total memory storage required for user data, the number of transactions allowed per second, etc. We call them features in this context. Certainly, there will be many such online sites on the web.

Suppose we have training set consisting of each of such features (n features) for m different websites. Also, we have the known output (in our case the cost incurred) for each of them. Our problem now transcends to a supervised learning problem. A smart way to predict the cost of the product would be to run a regression algorithm upon the data sets such a linear regression or polynomial regression and predict the estimated price of our product based on the training data.

Taking the example of the Hospital management website, let's assume that our feature vector consists of an m x n matrix where n is the total number of features. In our case n comprises the number of users of the product, the total memory storage required for user data, the number of transactions allowed per second, the domain name category and total access to the website per month. Having these feature vectors and m training examples which are the records of such features from the existing models each of which has a known cost Yi, we apply a regression algorithm over the training data and using gradient descent we converge to a global minima for the mean square error function, thereby training our parameter vector THETA (Θ). Table 3 below depicts the sample of data set used for the purpose of cost prediction. Having obtained Θ we now feed our feature vector to the algorithm which then gives us the cost for our product as shown in figure 3. Here the feature vector is compressed to a single dimension along x axis (with loss of some information) in order to facilitate plotting. Figure 4 depicts the plot of the cost function (J) against the number of iterations via gradient descent. As evident the cost function reduces with each iteration and finally converges to global minima.

Table 3: Sample Dataset For Cost Prediction

Users	Storage	Transactions per second	Domain Name	Access/month	Cost
100	500	10	10	100000	500000
25	125	3	4	30000	125000
300	1500	30	10	300000	1400000
150	1000	20	8	150000	950000
50	250	6	8	30000	300000

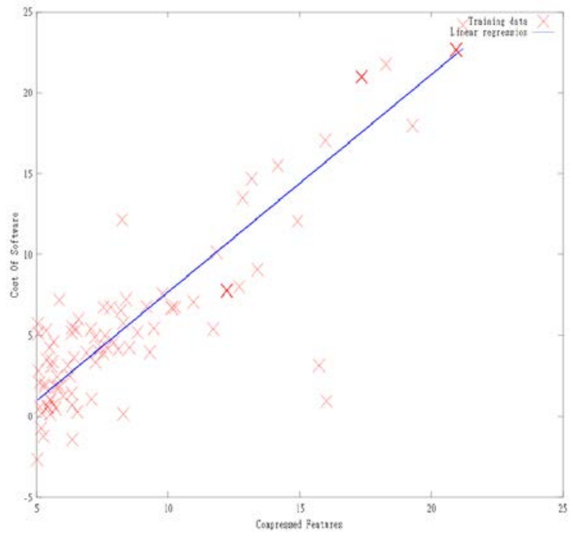

Figure 3: Result Of Linear Regression

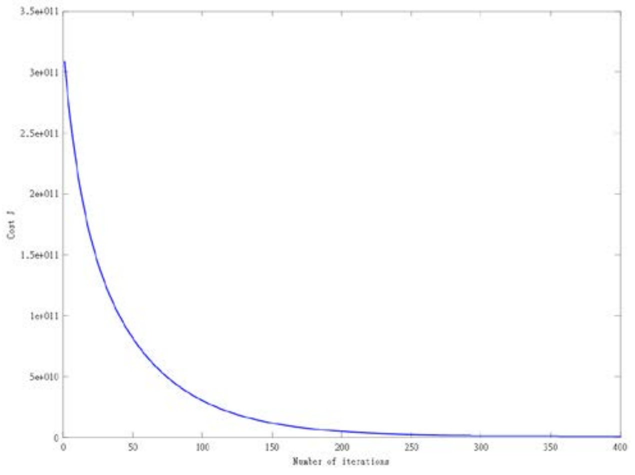

Figure 4: Result Of Gradient Descent On Cost Function After Each Iteration

Note that the features used in our training set are just for understanding. In reality, the cost of such software will depend on many other factors.

2.1.1.1 Requirement engineering

Requirement Engineering involves elicitation of functional and non-functional requirements for the particular software [2], [7]. Once we have finalized the requirements, the next step is to select appropriate tools to fulfill these requirements. This is usually the tricky phase because very often we have a variety of similar tools to choose from. When the size of the software is large, such a selection becomes increasingly difficult. A good way to go about doing this is to model the product based on its similarity to other similar successful products whose requirements specifics are already available to us. For instance, consider again the case of building an online Hospital management system. Let's say we are stuck with a choice amongst among many server side scripting languages like JSP (Java Server Pages), ASP (Asynchronous Server Page), PHP (PHP: Hypertext preprocessor) and PERL. Similarly, for database system we have a choice between Oracle, DB2C, MYSQL and SQL server. Also, for the purpose of UI design we have a choice between Bootstrap, CSS3 and Foundation JS. Similarly, for the purpose of Backend manipulation we have a choice between Python, Ruby on Rails and Java. One way to approach this problem is to match our requirements with the successful models based on its similarity to them. The one which resembles our requirement most closely i.e. the one with the closest similarity measure then can be chosen as our sibling model. Since we already know the tools used by the sibling model we can select those tools for our model as well.

As a concrete example, lets us assume that our training set comprises of following features: Security, Data Size, transactions per second and User Interface. After requirement analysis, we have arrived at some values for these features for our website. Given an upper cap on the price and time duration to complete the project, each of these feature has a value between0 and 1 i.e. say for instance, if Security parameter has a value 1 it implies that system must use the best possible security tool, a value around 0.5 implies that security can be compromised in case of a tradeoff with other features such as User Interface Design or Datasize. We have the m x n matrix again, where n represents the above features and m is the number of examples (of the successful websites) in our training set. Suppose we define our requirement vector as

Requirement Vector:[0.9, 0.5, 0.7, 0.2, 0.6]

Table 4 below depicts the sample data set structure used in our hospital management example.

Table 4: Sample Dataset For Tool Selection

Security	Data Size	Transactions per second	User Interface	Model
0.7	0.9	0.7	1.0	1
0.5	0.3	0.6	0.7	2
0.8	0.5	0.7	0.2	3
0.4	0.2	0.9	0.8	4
0.1	0.9	0.1	0.3	5

Here, a high value for security quotient (0.9) implies that we want our website to be highly secure and a low value for UI quotient (0.2) implies we are not very much concerned with the User-Interface design. This is because for a Hospital management system security is of utmost importance as most of the data stored contains important patient diagnostics details. Also we may not be concerned on having a very rich UI for such a system. Having figured out such a vector during our Requirement Engineering phase, we now measure Gaussian similarity between our feature vector and all other feature vectors in our training set. The Gaussian similarity is defined as follows:

With m feature vectors each of dimension n, our training data consists of $x^{(1)}$ to $x^{(m)}$ where $x^{(i)}$is an n dimensional feature vector. We define f1, $f_{2,}$ f_3 and so on till f_m, as a measure of similarity between a feature x $^{(i)}$ and $x^{(j)}$. Concretely, we define the similarity function as follows:

f_1=similarity$(x^{(i)}, x^{(1)})$ =$e^{-(|| x-x^{(1)}||/\sigma^2)}$

f_2=similarity$(x^{(i)}, x^{(2)})$ =$e^{-(|| x-x^{(2)}||/\sigma^2)}$

f_3=similarity$(x^{(i)}, x^{(i)})$=$e^{-(|| x-x^{(i)}||/\sigma^2)}$

f_m=similarity$(x^{(i)}, x^{(m)})$=$e^{-(|| x-x^{(m)}||/\sigma^2)}$

And so on...

Here $||x^{(i)} - x^{(j)}||$ is the Euclidian distance between $x^{(i)}$and $x^{(j)}$ and σ is the parameter of the Gaussian distribution which is a constant. It is easy to observe from the similarity formula that when $x^{(i)}$ is close to $x^{(j)}$ the similarity function outputs 1 and when $x^{(i)}$ is far away from$x^{(j)}$it outputs 0.Hence it is a discrete measure of similarity between $x^{(i)}$ and$x^{(j)}$.

Note that this Gaussian similarity measure is same as that used to define the features in Support Vector Machines (SVMs) [4]. We now, calculate the similarity measure of our requirement vector from all the examples in the training data. The one, closest to our requirement vector is called the sibling vector. We then, choose our tools based on the tools used by the sibling vector. On comparing our requirement vector with our dataset we found our dataset closely matches (i.e. the Gaussian similarity measure tends towards 0) the following vector:

Requirement Vector: [0.9, 0.5, 0.6, 0.2, 0.7]

The above requirement vector in the training set corresponds to an online transaction based website. This close similarity can be attributed to the fact that both of these website have high security quotient and low UI quotient. Since, the tools and software used in the development of this site is already present in our database we can easily model our new software based on them.

2.1.1.1.1 Maintenance

Towards the tail of the software development lifecycle we have the maintenance phase. Though the maintenance phase is very specific to the software, in most cases we can use machine learning algorithms to effectively address this phase as well. Let's again take up the same example of building a website for Hospital management system. Once the product is complete and put on the server, our job is not done. A running website is susceptible to many threats. Some of them include denial of service attacks, virus and Trojan attack, an attempt to steal important Hospital data etc. One way to ensure safety and smooth functioning of the website is via anomaly detection i.e. we need to monitor our server in order to detect any fraudulent or anomalous behavior.

We can record activities like memory usage, network traffic, number of disk accesses per second, the CPU load on the server, number of mouse clicks per second for each session variable etc. Say, we have a training set that consists of n such features over m examples. Out of those m cases most are records of normal activity but some are records of fraudulent activity. Say, we represent a normal user activity by assigning an output value 0 ($y_{(i)}=0$) and an anomalous one by output 1($y_{(i)}=1$). We can now use an anomaly detection algorithm as follows. Having such a data set the next step would be to fit a Gaussian probability distribution over our training examples. Instances of perfectly normal activity will have high probability in such a distribution while anomalous activities will have very low probability. A natural step then would be to identify those very low probability regions on the bell-shaped curve.

In our example, say we our monitoring our server for abnormal activities. Suppose someone launches a Denial of Service (DOS) attack on our website. For the sake of simplicity, let's assume that our feature vector consists of only two features CPU load on the server and memory usage or disk access and m here are the users accessing our website each uniquely identified by their IP address. A DOS attack would imply that even though the memory access is low the CPU load would be very high because the attacker would simply bombard PING requests on our server without doing any actual activity on our website. Table 5 represents a sample of our training set. Figure 5 represents the CPU load versus memory used plot. Figure 6 shows instances of such activity as points outside the Gaussian curves. Clearly these IPs will have a very low probability distribution and will lie as an outlier on the plot as shown in the figure 7 marked with a cross symbol. An obvious step then would be to either block such IP addresses or ask the user to establish their identity for further access.

Table 5: Sample Dataset For Anomaly Detection.

CPU Load on system (%)	Memory Usage of system (%)
20	39
7	8
27	35
14	17
85	2

3. FUTURE WORK AND SCOPE

As mentioned above, the techniques discussed here are, by no means, exhaustive. There are many other phases in the software development life cycle where AI can be used. Some of them are described next at an abstract level for the sake of completeness.

Consider the problem of software testing, which involves generating test cases to check the functioning of each module. Machine learning can be used to develop those test cases by generating test sets for our module based on the history of similar modules. Artificial neural networks (ANN) are most helpful in this regard. During the requirement gathering phase we obtain an informal description of functional and non-functional requirements which then have to translate into formal specifications. It turns out that Natural Language Processing (NLP) can be used to bring about this conversion. AI can also be used during the design phase for prototyping from a training set of similar models.

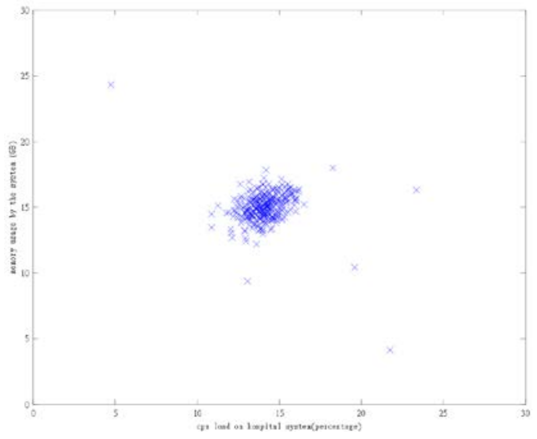

Figure 5: CPU Load Versus Memory Plot

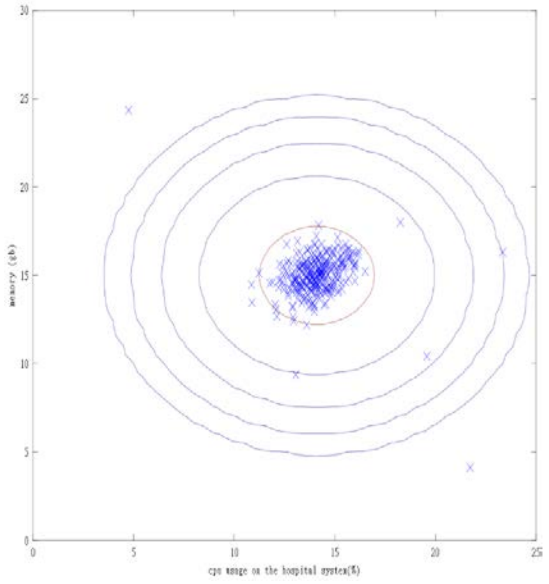

Figure 6: Gaussian Probability Distribution On The Dataset

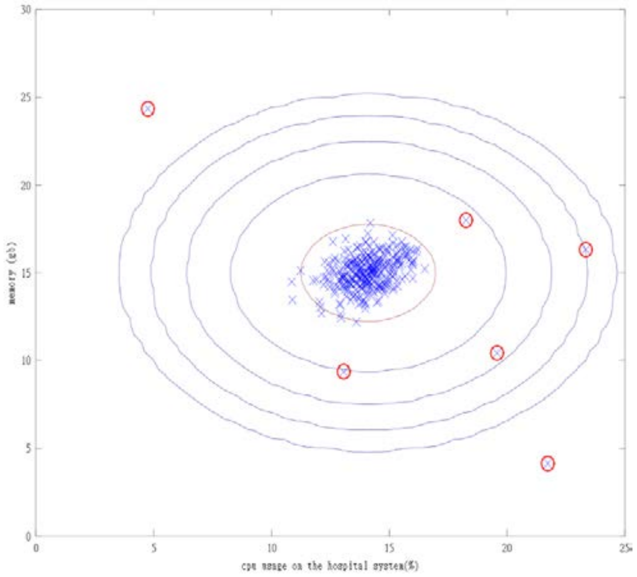

Figure 7: Identified Anomalies

4. LIMITATIONS

The paper only addresses a few out of the many possible approaches to using AI in the field of software engineering. The algorithms suggested here are only to provide an intuition on how to go about solving a problem in a particular state of the development life cycle using machine learning algorithms. They are by no means exhaustive. Generally, the success of an AI system depends heavily upon the kind of data available with us and the way we go about choosing our feature vector. Selection of features is a broad topic and domain specific, which is outside the scope of this paper. In case of unsupervised learning algorithms such as clustering, effectiveness also depends on the way we define the similarity measure between our feature vectors. The similarity measures defined in our examples above are quite abstract and at a very high level.

5. CONCLUSION

In this paper we provide solutions to different problems in a software development life cycle using Artificial Intelligence. Our aim here is to provide a general feel of how AI can be effectively applied in software engineering. As such, we have taken a concrete example to describe each technique discussed above and applied learning algorithms to demonstrate their functioning. Though these techniques are highly generic and scalable to other software development process, we have chosen a particular scenario to ensure clarity and proper understanding. An important point to note here is that the performance of these techniques, just like any other AI techniques is highly subjective to the type of software, the quality and amount of data available and the diligence with which we choose our features.

REFERENCES

[1] Justin S. Di Stefano and Tim Menzies,Lane machine learning for software engineering: case studies in software reuse.
[2]Du Zhang, Applying machine learning algorithms in software development.
[3]Mark Harman and Bryan F. Jones,Search-based software engineering (2001).
[4]Ralferbrich, ThoreGraepel and Klaus Obermayer, Support vector learning for ordinal regression (2005).
[5] Rich Caurana and AlexandruNiculescu-Mizil, Empirical comparison of supervised learning algorithm (2006).
[6] Derek Partridge, Artificial intelligence and software engineering (Glenlake Publishing Company Ltd.).
[7]Jonathan Onowakpo Goddey Ebbah Deploying Artificial Intelligence Techniques In Software Engineering
[8]Engr.Farah Naaz RazaArtificial Intelligence Techniques in Software Engineering (AITSE)