Machine Learning in Software Engineering

1. Introduction:

Technology has a great impact on our lives. The role of software has changed remarkably over the last few years. It is because of the revolution of the hardware industry, improvements in memory, performance, architecture etc. Different kinds of software are integrated into machines to comfort our daily life needs. Software engineering plays an important role in building up a good product. The software itself is more than a piece of program code. It consists of executable programming code, libraries, APIs, and documentations. The development of the products using all the scientific standardizations, definitions and principles is engineering. According to IEEE, software engineering is defined as:

"The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software. "

Software engineering combines process, method, and tools that helps to build complex systems with high consideration of quality and time. However, the need for this increases by showing betterment in large size, scalability, cost, and quality management of software. Software engineering has been divided into multiple subcategories e.g. software requirement, software design, software testing, software maintenance and software quality etc. For performing all these tasks, systematic way is followed that consists of rules, models, architectures and different standards based on the customer/user or product demands.

Machine learning (ML) is a buzzword of this era. Artificial intelligence (AI) is a well-known and mature field in computer science domain. Machine learning is a part of AI. Machine learning helped to solve complicated and hard problems efficiently. Machine learning mainly relies on the data. ML algorithms learn-from the existing data and predict the unseen problem solution. Its learning process has a great inspiration from human learning i.e. learning from the examples. Ml algorithms have proved to have a great impact in different fields e.g. business, medical, software engineering, computer security, data and communication networks etc. For the leaning process algorithm, another key factor is the feature. Features correspond to the characteristics of the learning and provide the base to the algorithm. In addition, feature helps to abstract the complexities of the information provided for the learning and shortening training times. In short, features help to reduce time and complexity of the models for learning.

ML algorithms can be divided into two categories; supervised learning and unsupervised learning. In supervised learning, data set is labeled that means it has some example with the defined features and output. New input or query will predict from the learning of the labeled data. All regression and classification algorithm came under the umbrella of supervised learning e.g. Logistic Regression, Decision Trees (DT), Support Vector Machine (SVM), Nearest Neighbors (NN), Naive Bayes, Random Forest and Artificial Neural Network (ANN) etc. On the other hand, unsupervised learning is from the unlabeled data. It covers all clustering algorithms e.g. k means clustering and hierarchical clustering etc.

In the world of ML and SE, one intersect region shows the application of ML in SE. It has shown the implications in software requirement engineering, software development, software cost and effort estimation, software verification and software quality. We have survey last seven-year (2011-2017) top tier software engineering conferences that included International Conference on Software Engineering (ICSE), Foundation of Software Engineering (FSE), Automated Software Engineering (ASE), IEEE transaction on Software Engineering and ACM SIGPLAN on programming languages and implementation.

Based on the information provided in [8], the search formulated on the core areas of the SE.This review search is based on the following criteria:

I. **Seach Engine**: Targeted Conference (ICSE, ASE, FSE)
II. **Year of Publication:** 2011-2017
III. **Information Gathering Strategy**: Read Title, Abstract and Conclusion section and decision to exclude/include paper.
IV. **Research areas:**

- Requirement Engineering
- Development Cost
- Defect Detection
- Traceability links
- Maintenance or task effort
- Repair and Performance
- Testability of prog and modules
- Software Quality

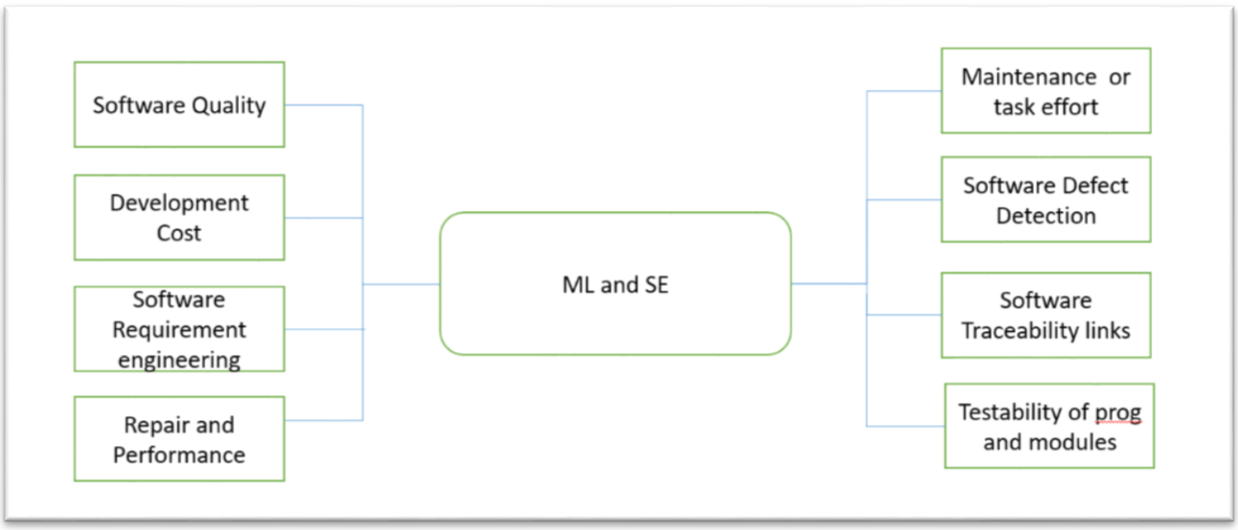
Software Quality
Development Cost
Software Requirement engineering
Repair and Performance
ML and SE
Maintenance or task effort
Software Defect Detection
Software Traceability links
Testability of prog and modules

Fig 1. Machine learning scope in Software Engineering

From our research, we have figured out the current scope of ML in SE is Fig 1. That actually depicts the intersected areas of ML and SE that means the use of ML algorithm in SE domain. Almost the same areas with more intersected sets have been covered in the last survey on this topic [8]. Additional intersected sets can be the cause of different reasons i.e. the dataset for the survey covered multiple databases and with a larger time interval (1987-2001). We cannot completely claim that excluded areas of SE are dead areas or no more into research trends. It might be ignored because of the filter applied for this study i.e. time and selected conferences. Before that research conduction [7], [8] has been considered as a background for this study. The below Fig. 2 illustrates the statistics of different ML methods used in SE.

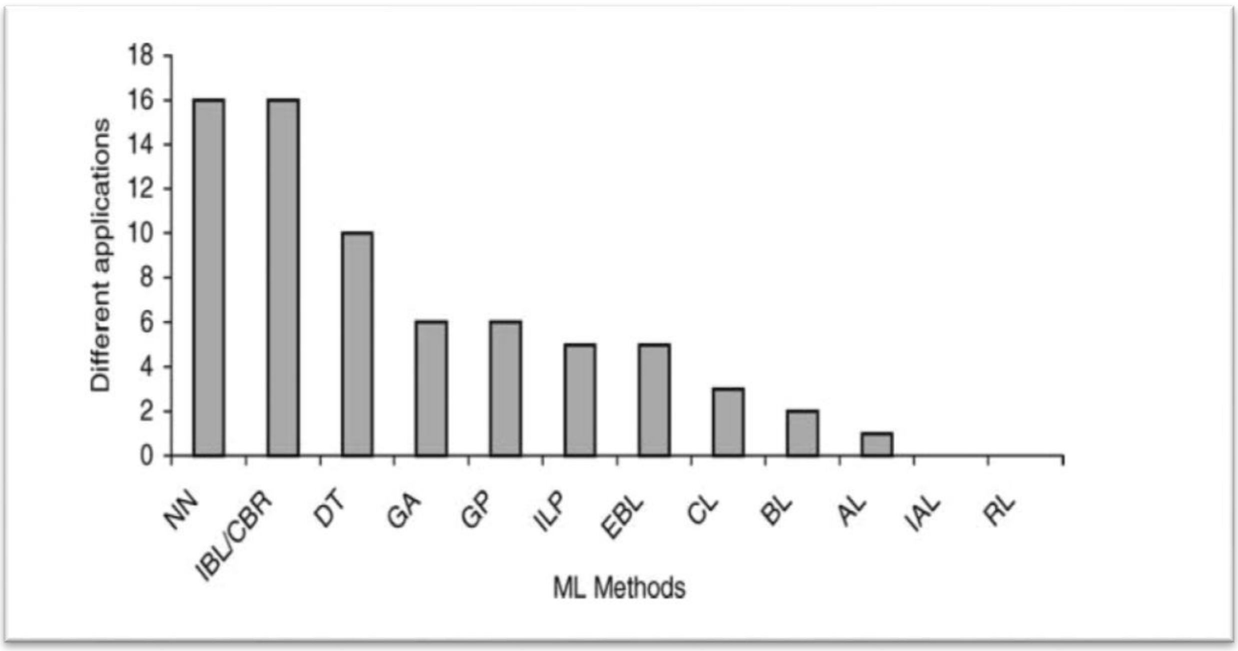

Fig.2 State-of-the-practice from the perspective of ML algorithms [8]

The current research trend from 2011-2017 in the domain of software engineering and machine learning shown in Fig 3.

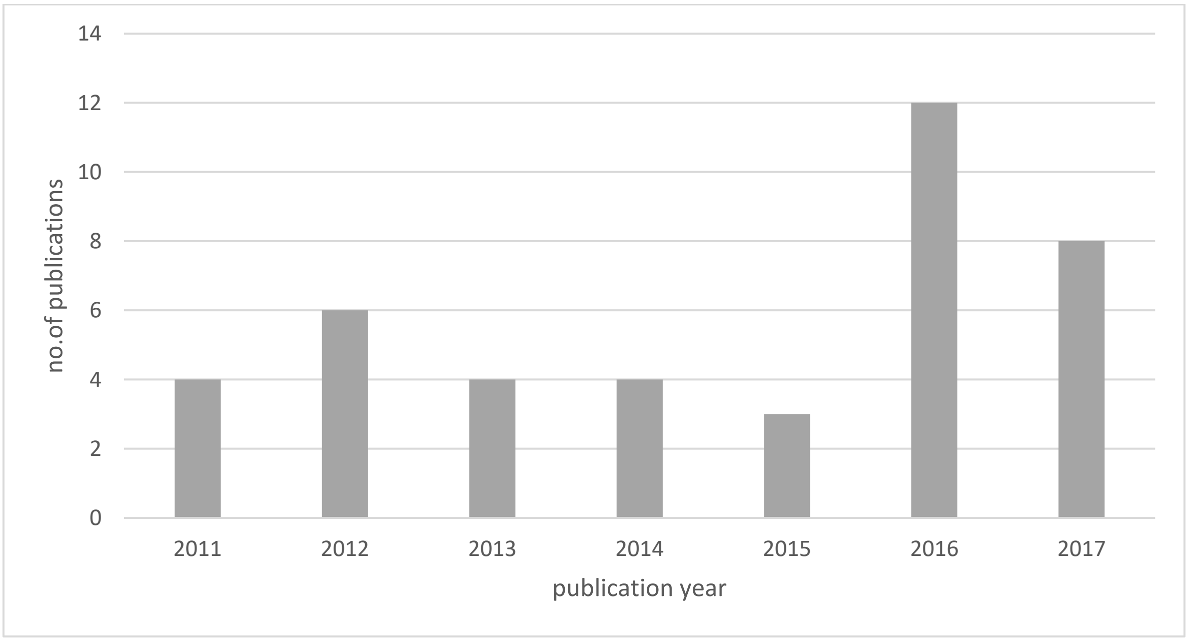

Fig 3 Research trend in ML & SE

2. Literature Review:

2.1 Software Traceability:

In software development life cycle (SDLC), each phase has its own artifacts. All are having different sets of the documents and formats. For the effective development and management, there must be traceability links between the artifacts. Through machine learning via using classification traceability link recovery is being done automated [2]. The presented approach automatically classified all potential links as true or false based on the features i.e. IR ranking, Query Quality and Document Statistics. For the correct classification of valid links that effected by the class imbalance, Synthetic Minority Oversampling Technique (SMOTE) and majority under sampling used for balancing the data. For the experimental setup Naive Bayes, KNN, J48, and random forest implemented in weka tool. Random Forest gave the most promising result between all of them in SMOTE and under sampling while considering both TPR and FPR.

For the traceability link between sections in documents and class entities discussed in [1]. Information retrieval techniques vector space model (VSM), regular expression (RE), key phrases (KP) and clustering is used to extract the links between sections in documents and class entities. This combination helped to overcome the limitation of IR. KP using gave the facility to IR for generation of all possible links. The true links at high cut points were increased by integration of RE. Mostly documents have a hierarchal structure. Documents division usually done by heading. Some sections have multiple headings and out of those, some has further extensions of heading i.e. sub heading. That means document structure can be replaced with parent, sibling, and child relationship. By taking advantage of this structure, clustering is used to reduce the fault links at low cut. With this technique, result improvements were noticeable as compared to the existing system. The basic techniques VSM finds the links between the class entities and documents sections. Two categories defined for finding the class and names in the documents; first is with one word name e.g. print, main etc. and second is with compound words. For the document's summary from the comments of the code, Key Phrase (KP) technique is used to extract the keywords. This keyword mining helped to know the alternative words to class name and identify what tasks the class has fulfilled. To overcome the contradiction of the expected results and real extracted results, key phrases were added to the VSM queries. In the last, k mean clustering algorithm with some modification was used for the clustering of the documents.

For finding the tractability between the software artifacts by using semantic of documents and domain knowledge was developed in [6]. The solution designed to automate the capture of domain knowledge and the artifacts' textual semantics with the explicit goal of improving accuracy of the trace link generation task. For each artifact (i.e. each regulation, requirement, or source code file etc.), each word is replaced by its associated vector representation learned in the word embedding training phase and then sequentially fed into the RNN. First, it learns a set of word embedding for the domain using an unsupervised learning approach trained over a large set of domain documents. The first part of the proposed system was finding the words embedding that was done through applying unsupervised learning. Domain documents used as an input for the training set. The first phase generated the high dimensional word based on the co-occurrence and distributional semantics. The second phase defined by the training of RNN that build semantic similarities between the documents. The Bidirectional Recurrent Gated Unit built semantic associations between artifacts. As a resultant, it delivered higher MAP scores than either VSM or LSI. However, this system did not include all type of documents e.g. source code that was left as a future work.

No.	Problem Area	Description	Solving Technique
1.	Traceability Link between the documents/ artifacts [1]	Information retrieval techniques vector space model (VSM), regular expression (RE), key phrases(KP) and clustering used to extract the links between sections in documents and class entities.	Clustering
2.	Traceability Link Automation [2]	The existing techniques still needed human input for manually inspection and decision for identification of true or false links. In this work, automation is being done with help of ML.	Random Forrest
3.	Traceability Link between the software artifacts [6]	Finds the tractability links between the software artifacts by using semantic of documents and domain knowledge.	Recurrent Neural Network (RNN)

2.2 Software Requirement Engineering:

Software requirement engineering is a key component in software engineering. The quality of product depends on the understanding of the requirements. As time passed software development models gets a different vision and style that can help to accommodate the user needs. Mostly, companies are focusing on agile models. Developers and teams have more close interactions with the users. In the discussion of the customers and developers, developers noted down the requirements and on the base of that, generate the user stories. In the past, more emphasize was on hand written stories. To generate the summaries from the user stories, information is retrieved automatically through ML [4], an online corpus was developed. In this recorded data i.e. discussion of user and developer can be uploaded and by using the trained classifier, user useful information can be extracted to generate the executive summary. Two different classifiers SVM with RBF kernel and logistic regression used to implement this concept. The notable factor was the use of the SMOTE to overcome the supervised algorithms problem. Supervised algorithm made prediction on behalf of everything that lied in the large class and ignored the small classes. Also for the conversion instead of the sentence author used turns i.e. the unit of speech. The methodology starts by observing, taking notes and recording audios from nine different meetings between developers and the customers. The meetings are then transcribed with identification of different speakers. Each conversation was then annotated manually by at least 3 authors. It was found that in the 3176 turns, 5.5% had function information, 2.9% had rationale information, and only 0.5% turns contained role information. The less role information is because of the fact that people already know their role in the project, and is the least discussed during the meeting. It is also found that 10.2% turns belong to extractive summaries of their respective conversations. The approach then consists of developing machine learning classifier to detect if the turns in speech have any information related to function and rationale. It also focused to build classifiers to create extractive summaries of the conversations. Predictions models are built based on support vector machine and logistic regression techniques. Training is performed on all conversations except one, and then test is performed on the remaining conversation to create a cross validation fold. The models resulted in 54% precision for detecting conversation containing function data, and 25% precision containing rationale data. Whereas, for extractive summaries, the precision obtained is 70.8%.

Requirements have a key impact in each phase of the software development process. The output from the requirement gathering is SRS documents. When some points and requirements mentioned in the document does not include the full details. It caused ambiguity and it gave a chance to software architect to make his own assumptions. This sometimes can lead software designer in a wrong direction. That effects software cost and time. The simple solution for finishing this ambiguity can be asking help from the business analyst. However, he cannot help me effectively because of his limited technical knowledge. To overcome this problem with the help of ML, probing questions can be given to business analyst regarding the architecture design. Therefore, he can ask all the questions and provide full details of requirements. The main focus was on the functional requirements. The group of five Architecturally Significant Functional Requirements (ASFRs) areas defined: Audit trail, Batch processing, Business process state alerts, reports, and workflow. To detect the ASFRs by different ML algorithm tested that included Naive Bayes, J48 Decision Tree, SSVM, Nearest Neighbor and AdaBoot. The Naive Bayes observed the most promising results. The main issue was in this prediction that data i.e. functional and nonfunctional requirements were not equally distributed. Random sub sampling used to solve that problem. After detection of the ASFRS from the data, multi labeling technique helped to predict the belonging to the defined five categories. Fit helped to follow the probing question (PQ) flow. With this use of PQ, flow business analyst can follow the flow of questions in order to elicit architectural significant info. Furthermore, VSM used for the associating requirements with specified PQ and computed the similarities between requirements and PQ [5].

For verification of software requirement model new framework is proposed in [3]. In autonomous and embedded systems software development formal verification used for the quality and safety. Still, in the specification process, human input is required that causes the errors and inconsistencies. The information obtained by the current tool used to revise the model. For making this process automated different techniques for evolution existed. The proposed framework used the New Symbolic Model Verification (NuSVM) tool for getting the verified properties in temporal language. Neural Networks used for the adaptation purpose. In this framework, symbolic reasoning and robust ML is used for the adaptation and evolutions of the models. The learning is performed to integrate the sources of knowledge, the initial description of system translated into a logic program (a) by using SCTL, sets of examples of the desired behavior of the system (b), or by general sequences of events that the model should satisfy (c). After the learning process, the engine represents the knowledge learned through a list of state transitions (d), which are associated with numerical values providing a measure of their importance in the system. The information gathering for a system improved by the consideration of error handling and noise tolerance features.

No.	Problem Area	Description	Solving Technique
1	Requirement Engineering [4]	Generate the summaries from the user stories information retrieved automatically through ML.	Classification
2	Requirement Engineering [5]	Probing question and detection of the architectural significant functional requirements	Naive Bayes

3	Requirement Engineering[3]	Proposed a framework for adapting and evolving Software requirement model.	Recurrent Neural Network (RNN)

2.3 Software Defect Detection:

Software defect or bug detection plays a major role in software engineering. These models provide the list of the bugs in software and help for the effective allocation of resources for the testing of software. If the software has a small size, it is easy to test it. Dealing with the larger size software it is hard to test and software will have defects / bugs. It is hard to make sure how much defects you have corrected and how much is left. It was estimated in 2002 that 80% of the total project spend on the defects correction [9]. For making it convenient, different prediction models proposed in the literature based on Machine learning. All these models predict using software metrics e.g. line of code (LOC), MaCabe Cyclomatic Complexity (CC), Hallstead and C & K metrics etc. The researcher has used the NASA dataset (PDI) for predicting the data by applying classification and regressions models. This area has three mainly focused subdivision i.e. software defect prediction model (in the same project), cross project defect detection, and effort aware just in time (JIT) detection. We have classified our selected papers into groups according to the problem statements discussed.

Cross project validation is detecting bug not into the similar project but across different projects. Usually, for most of the projects, historical data is not available. For this, it is important to have a way that can use already existing data and help to detect bugs. Different supervised algorithms have been used to solve this problem but it did not give satisfactory results. The main reason for these results were the ML algorithms with the concept of the similar test data. For ML algorithms test data derived from the same feature space. That condition does not hold by the cross project prediction. For this problem solving spectral based unsupervised classification and transfer learning used [10] [11].Transfer learning is used to predict the bugs in a cross project. The main idea for transfer learning is when adding more data to the existing model, building a new model with old and new data is expensive. In the paper [10] extracted the same knowledge from the projects and transferred knowledge used to build the prediction model. Additionally, data filtration applied on the data before applying the ML. Transfer learning state of the art learning approach, transfer component analysis (TCA) used with the extension named as TCA+. The additional feature was the automated selection for the normalization option according to the target and source project. Relink, AEEEM data set used for the experiments as a source and target projects. The other example unsupervised algorithm used to solve the same cross project prediction [11]. Mostly unsupervised classifier did not compete the supervisor classified but the addition of spectral clustering showed a brilliant result. These results are not only good but also compatible with the supervised classifier results. Unsupervised classifier predicts the defects without access to the training dataset. Spectral clustering works on the connectivity that calculates on the base of similarities. It partition dataset based on the connectivity between data. Furthermore, clustering performed on the graph. This graph node showed software entity e.g. file or class and edges contained the connection values. The weight measured between two ends of software metric. A dataset of total 26 projects from AEEM, NASA, and PROMISE used. The other key point was the proof of the strong connection of the buggy files and weak connection of buggy and clean file. CLA/CLAMI [13] also worked for the cross project unlabeled data bug predictor. The main focus was to reduce human effort and making an automated tool for unlabeled data set and predicts the bugs. It has four phases named as clustering, labelling instances in cluster, metric and instance selection. For finding the AUC curve on supervised learning model and CLA model logistic regression was used.

Many researchers have focused on the area of bug reports and solved different related problems with ML algorithms [12] [14] [15] [17]. Bug reports contained the collected information about the bugs encountered using the software. Developers used the information stated in the report for the diagnosis and removal of software bugs. The Bug report is the documented file of the predicted bugs by tool, users, and tester etc. When it comes to removing the bug if the file is not defined and clearly mentioned e.g. if the scenario is mentioned without mentioning then exact the file, the developer has to make a logical connection between the files for making it correct. If the project is large enough, it is hard to identify manually. Mostly information retrieval technique (IR) has been used to extract the texts from the code and bug reports. The bug report has natural language text and code has been written under technical rules. Because of this, IR faced the limitation of the lexical mismatch. The proposed solution for this problem used advanced vector space model (rVSM) and deep neural network and showed better results than the existing state of the art systems. [12]. RVSM used to calculate the similarities of textual features. DNN used for computation of the relevancy between the features. It contained mainly four types of features identifiers, comments, a textual description of API, API classes etc. Bug fixing history added up with help of recency score as a metadata feature. The reason was more bugs existence in the most recent files as compared to the old file. rVSM, DNN, metadata feature again passed to another DNN that learned the weights for these three to identify the bug. The final score computed for each bug with source files and ranked. To avoid the redundancy of information and reduction of feature auto encoder was used. The other problem in the bug report is the duplication of the same bug. For this problem, Dupfinder tool [14] implemented an unsupervised VSM model. This tool is an extension of already existing web based bug tracker tool Bugzilla. Dupfinder accepts new bug report, last created bug reports i.e. recent report. For the new report text preprocessing used for the creation of vector space model. Three IR preprocessing steps included tokenization, stop removal, and stemming. In tokenization, all special characters and symbols got removed. Stop removal step removes the common occurring word e.g. “I”, “You” etc. Stemming reduces all same form root words. Afterwards, similarity measured between the new and recent bug reports. In the last step, the list of relevant bug reports was returned by the server. In [15], the same problem i.e. bug duplication was addressed and solved by extending BM25F i.e. used for short query of text retrieval. Additionally, other metrics related to software and report used in building up the training set. These other features e.g. version and priority of the report, components identified with bugs were taken from the Bugzilla. The proposed model improved the state of the art by 10-27 and 17-23 in recall and mean average precision respectively. It was tested on the large bug report dataset from the open source projects including some of the Mozilla sub projects, Eclipse and OpenOffice. For the retrieval of the bug report tokenization, steaming and stop work removal preprocessing was applied on both existing and new bug reports. The bug repository treated as a bucket that is hash map like data structure. Whenever a new report came, system calculated the similarities between the new report and each bucket and returned the K master reports. The first contribution from the paper was the extension of BM25F. In that case, query is a new bug report contained the summary and long description of the bug report. The main two components i.e. inverse document frequency (IDF) and local importance measure (TFD) defined the BM25F.IDF is overall weighting scheme in the documents The extension was done by considering the term frequencies in queries. For prediction, two round gradient applied.

For software defect detection, Dictionary learning (DL) has been used [18]. Cost-Sensitive discriminative dictionary learning (CDDL) algorithm was used to predict the defects in software designed using supervised cost sensitive dictionary learning. Dictionary learning method learned from the training samples where the given signal was well presented for processing. Supervised dictionary learning reduced the number of dictionary atoms and complexity of sparse coding. The algorithm of CDDL steps are:

- Initialize the dictionary and sub-dictionary. The common used Shared DL methods replaced by the learning sub dictionaries using primary component analysis (PCA). It solved the dimension reduction and class imbalance problem.
- In the second step of CDDL algorithm, it updated the sparse coding coefficient matrix X by fixing dictionary D.
- In step three, D updated by fixing X.
- D returned as an output in step 4 and iterates step2 until maximum iteration number limit reached or adjacent iteration are close.
- The last step was prediction with sparse representation based classification (SRC).

During the software defect prediction errors have two type; first defected modules classified as non-defective and second is non-defective defined as defective. These errors lead to cost issues i.e. increase and risk. Increase in cost is by identifying non-defected as defected module and risk involved by identifying defective modules as non-defective. CDDL has a penalty cost function i.e. punishment increment when a defective module predicted wrong. In this study, NASA dataset was used and out of total 40 metrics, 20 common metrics were selected. For evaluation of the performance of the proposed model recall, false positive was calculated on the dataset. Random division with 1:1 applied for getting the training dataset for all the compared methods. To avoid the prediction performance random division done 20 times and average results considered as predicted results. In the comparison of the CDDL approach SVM, Compressed C4.5, DT (CC4.5) weighted Naïve Bayes, coding based ensemble learning and cost sensitive boosting neural network. CDDL had the highest false positive values and F-measure in comparison to other algorithms.

For the most important problem that exists in the static bug finder is soundness. That came up with unwanted false and non-precise results. The reason is the uniform unsoundness and soundness of the existing tools. To overcome this problem selectively unsoundness as solution discussed to reduce the false negative rates. [25] Two main instances used for the analysis i.e. interval analysis for finding buffer overflow and taint analysis for format string vulnerabilities. The set of programs that named as codebase and set of features used as a dataset. Features holdthe programming properties i.e. syntactic or semantic with binary or numeric values. For classification, One Class Support Vector machine (O-SVM) unsupervised classifier used, that learns from the positive examples. That is why before that all codebase has been filtered for the positive learning dataset that contains the harmless component. Afterwards, the features of these components were given as input to the classifier. Classifiers consider the instances, harmless instances analyzed unsoundly. If the instance identified as an anomaly i.e. harmful then instance (e.g. loop) analyzed soundly.

In 2003 NASA published a study on analysis of defected data [21]. This gave the comparison of the automatic and manual analysis of the defect data. The main challenge in the categorization of the reports that have the testing data of underdeveloped craft. These reports were loosely structured and hard to analyze. For this problem, orthogonal defect classification (ODC) was applied. This tool is used for the structuring of the defect reports. ODC has two categories named as activities and target. New defect tagged in the activities in progress and triggering event that identifies the problem after correction of defects analyst kept record related to defect type and its target of the fix. Manual ODC consist of five steps including initial categorization, final categorization, clustering, explanation, and action plan. Initial and final categorizations are manual process that changes loosely text in to ODC headings. This time consumption process will be solved (automatic) by modification of JPL problem reporting tool and IV & V defect tracking tool to include the ODC categories. After data entry methods has been changed, it will no more require to categorize the defect reports. In clustering distribution was uneven and focus should be on small number of high frequency problems. The large number of anomaly was recorded during sending commands and receiving data from spacecraft. The large number of anomaly from the receipt from the space to ground cells were because of many technical complications. After creation of cluster, it is audited by generating the explanation of each cluster explanation. This process is also manual as it involved so much business user discussion about the significance of each cluster. Action plans reflect the change in current practice if the practitioner changes it may be ignore. It stated that regression was not appropriate for the discrete data that's why association rule was implemented. Out of five, three just report same hierarchy the interesting one is the third i.e. a major clustering of defects around flight operations information development and procedures. This association exactly the kind of the result that seek from the defect analysis since it guides what intuitions is to revise and where the best focus future effort. It is useful to know at least something via automated methods in the case of no experts available or data is too complex. For the experiment, TAR2 was used. This learning assumes each class has a numeric value showing its worth to user. In ODC logs each defect were scored according to its criticality. Treatment is a constraint for the most interesting rations of classes e.g. what is the constraint for the highest/ lowest criticality errors. It is also best-viewed comparison as a baseline distribution and class distribution. It is also found for the lowest critical anomalies E, mostly values are unknown or none. The conclusion of the paper was that manual analysis by expert is better than the machine learning. Automatic ML can only access the symbol in the training set. However, dataset with large dimensions can be processed through ML. The two scenarios where ML is a good option; first is unavailability of the experts and second is audit check of TAR application on each incremental change of dataset. This activity will be cheap because automatic ML is faster to check the data quality.

The other new area in bug prediction is JIT (just-in-time) defect prediction [29]. It used a number of change metrics to predict the probability of changes to be defect-inducing changes. It is helping developers not by only narrow down the scope of the code that caused bugs. Additionally, it is predicting on check-in time when they still have a fresh memory regarding the changes they made. In traditional defect predictions after a module is predicted as defect-prone, it may be difficult to find the specific developer, who is most familiar with the code, to inspect the module to find the latent defects. As the predictions made on check in time, it is hard to make predictions with the supervised learner. The interesting part of this study was the use of unsupervised learning a method for the effort aware JIT defect predictions. For software defect prediction as state of the art method is learning from label~~l~~ed data. In this approach, the historical log of known defects is learned by a data miner. This approach requires waiting until a historical log of defects is available; i.e. until after the code has been used for a while. The proposed systems build an unsupervised model that ranks changes in descending order according to the reciprocal of their corresponding raw metric values. The smaller changes were focused more because more defects can be found by faster looking the smaller modules. For each change metric M, the corresponding model is R(c) = 1/M(c) where c represents a change and R is the predicted risk value. Bugzilla (BUG), Columba (COL), Eclipse JDT (JDT), Eclipse Platform (PLA), Mozilla (MOZ), and PostgreSQL (POS) dataset used for the experiments. Experimental results showed that many simple unsupervised models are similar to or even better than the best-supervised model in terms of ACC. After that, another study [28] came arguing the results and fact establishing used of unsupervised learning in [29]. The new study made opposite conclusion and stated the clear advantages of using supervised over unsupervised.

An interesting study [22] performed to identify the reasons for not using bug tools. For this study semi structured interview was conducted with twenty software developers. The research questions focus was to know about reasons of using and not using of static bug finder tool, the current tools fitting in their workflow and the last to know the improvements they want in the tools. Current static analysis tools may not give enough information for developers to assess what to do about the warnings produced and very seldom offer a fix to what it claims is an issue. It concluded that false positive and developer high input causes lack of interest to use these tools. If static analysis tools offered quick fixes, giving a potential solution and applying it to the problem may help developers assess warnings more quickly and ultimately save time and effort. At the same time, quick fixes do not appear to be a universally applicable mechanism to help developers resolve static analysis warnings because many static analysis warnings do not have a small set of solutions. Instead, interactive quick fixes that enable easy access to refactoring and code modification tools may be able to semi-automatically help developers resolve static analysis warnings. If a tool is to be customizable, it should be customizable in a way that is simple and useful to the developer.

The most important and key essence of ML is data. Mostly researchers have used the dataset for defect detection included NASA dataset. NASA dataset needs preprocessing to make the data more meaningful and comparable [24]. For example, a line of code (LOC) can be zero or one depending upon the programming language and definition of LOC but it cannot be 1.1. No referential integrity rules were checked to identify these kinds of errors. In addition, there were some features in the dataset with repetitions or with no impact i.e. constant or zero values on predicting the dataset. The proposed preprocessing algorithm checked implausible value i.e. referential integrity rules and feature repetition, its impact, and deletion of these values from the dataset. The ultimate goal was making dataset more meaningful and comparable. This study with the same issue on data quality issue was carried out in [25] with additional rules. Furthermore, it was emphasized in the study that for future research data must be cleaned. NASA should clean the data by using [25] and their own extended research.

No.	Problem Area	Description	Solving Technique
1	Cross Project bug Prediction [10]	Extracted the same knowledge from the projects and transferred knowledge used to build the prediction model	Transfer Learning

2	Cross Project bug Prediction [11]	To represent terms and fragment by mining the source code repository and linking patterns	Spectral based unsupervised learning
3	Cross Project bug Prediction [13]	For the cross project unlabeled data bug predictor CLAMI focusing on reduction of Human input.	Logistic Regression, clustering
4	Bug Report [12]	To represent terms and fragment by mining the source code repository and linking patterns	Deep Neural Network
5	Bug Report [15]	To overcome duplication of the same bug in report	Deep Learning
6	Bug Report [14]	To overcome duplication of the same bug in report	Unsupervised VSM model
7	Bug Report[17]	To overcome the manual identification of missing information from the bug report and improves the quality of bug report by describing the missing information.	Support Vector Machine.
8	Defect Detection [18]	Finding the defects in the software.	Dictionary Learning
9	Defect/Fault prediction[33]	Prediction of faults made on cluster or related classes rather than the entire systems. The proposed model built on using the properties of the classes in each cluster.	Clustering

2.4 Software Testing:

Load testing is performed for determining load of a system under stress. A large amount of data and counters are logged to capture performance of the system under load. However, the current practices are time consuming, error prone and generates terabytes of performance related data. The paper [20] constructs performance signatures from the counters. Analyzing signatures and comparing them with baseline signatures rather than entire date improves efficiency and provides a manageable set of data for root cause analysis. The proposed approach provides the approach for the automatic detection performance in load testing with minimal domain knowledge. Two dimension reduction algorithm are proposed; random sampling and clustering (K-means) and Principal Component Analysis (PCA). In random sampling, some counters selected randomly out of the all performance-recorded data to make a small signature. The performance counter threshold i.e. twenty chosen by the experienced practitioners. The clustering approach takes advantage of the fact that many counters measure common performance characteristic. Using K-means clustering approach, it divides n counters into k clusters. Control charts are used which compare the control limits of baseline load test with the load test to determine the percentage deviation. Two more advanced approaches Principal Component Analysis (PCA) and WRAPPER also used. PCA as unsupervised approach reduces the original data by projecting the dataset to a lower dimensionality space. The supervised approach WRAPPER presented in which performance counters data from baseline test needs to be labelled as passed or failed by an analyst. A search algorithm is used to optimize selection of attributes with respect to accuracy of prediction. A case study was performed to determine how effective the proposed approaches are in determining performance deviations in the load tests. Resource exhaustion, system overload, configuration errors and procedural errors selected as a fault injection. Seven different experiments performed and repeated ten times to ensure the consistency in the results. The result showed that among the unsupervised approaches, PCA always performed better. However, the supervised approach WRAPPER dominates PCA in terms of precision, recall and F-measure.

For determining of inadequate performance behaviors, performance testing plays a vital role in identifying various factors such as longer execution times and lower throughputs etc. Generally, engineers select inputs likely to trigger bottlenecks and test it. Mostly the process is manual and time-consuming. A big challenge is to automate this process of generating inputs likely to trigger bottlenecks and to analyze the traces to understand the cause of exposed bottlenecks. Two different scenarios are considered in [23]; a single-version scenario in which bottlenecks are identified in a single software version, and a two-version scenario in which changes of code causing performance regression are identified between two consecutive releases. In the single-version scenario, the traces are classified into two categories good and bad. Good traces have longer execution time and are more suited to expose bottlenecks and vice versa. FOREPOST and FOREPOST-RAND (rely on machine learning), are proposed to extract rules mapping performance behaviors to inputs. Depending upon traces, ML extracts rules for describing performance behaviors with corresponding inputs. Once there is no new rule extracted, profiling process is terminated. While FOREPOST finds some bottlenecks, its limitations to extract rules miss some bottlenecks. The reason is that FOREPOST focusses on local hot paths but fails to explore whole application under testing comprehensively. For this reason, GA-Prof is proposed, which uses genetic algorithms to find inputs that are more likely to expose bottlenecks. Starting with the initial inputs, profiling information is analyzed, and the input with longer execution times create inputs for the next generation. For two-version scenario, PerfImpact is proposed which uses GA to select test input data which worsen performance behavior in newer version as compared to the previous version. First inputs selected randomly and sent to both versions of an application under testing. The PerfImpact calculates the difference between execution times of same input for both versions and creates new inputs for next generation based on inputs causing the larger difference.

There are many test cases at the software and hardware levels of testing. These tests are frequently executed as regression tests. Only a few faults cause a large number of failures. Testing all the cases is too expensive and time consuming. It is not certain that the first identified fault is contributing more in order to enhance the failures or what other mainly faults causing the problems. The study [30] deals with the problem of reducing the failure analysis time. They proposed to cluster failing TCs with respect to the fault that caused them, select one representative for each cluster, investigate only the representatives, find the underlying faults and resolve them, then re-run the tests. The industrial partner BMW dataset used in experiments. First, a test suite was run and each test case execution profiled was extracted. In the second step, agglomerative hierarchical clustering used to build a tree of failing tests based on the similarity of execution traces. Each single failure at first is in its own cluster. Applying the distance metric similarities between clusters is measured. A new cluster built up after merging of the two similar clusters. After that tree cut down into clusters and fault localization techniques used to decide on the best number of clusters. The spectrum-based, or coverage-based, techniques used to rank suspicious entities. The center of the cluster calculated from the k-nearest neighbors (KNN). These KNNs selected as representatives of the corresponding cluster.

This study[31] followed the same problem of regression testing problem. Test prioritization helped to optimize the number of running test cases. After ranking, the test only like hood test run is done for revealing the failures. The proposed solution used ML for the test prioritization. Learning to rank is a sub domain of ML. It deals with the construction of ranking models for Information reterivial(IR). The main objective is to train models that rank unseen lists. Five features were given to the learning model that includes. Java code coverage, text path similarity, text content similarity, failure history, and test age. The features integrated through a linear model trained using SVMmap.

No.	Problem Area	Description	Solving Technique
1.	Load Testing [20]	Proposed automatic detection performance in load testing with minimal domain knowledge.	Clustering
2.	Performance Testing [23]	To automate process of generating inputs likely to trigger bottlenecks and to analyze the traces to understand the cause of exposed bottlenecks.	RIPPER and Genetic Algo
3.	Testing[30]	Reduce the failure analysis time and identify the main faults that cause failures.	Hierarchical Clustering
1.	Test Prioritization[31]	Test prioritization helped to optimized the number of running test cases .Test prioritization model learning with ML.	SVMmap
2.	Combinatorial testing [32]	Automated generation of the input classification and dependencies for the combinational testing.	Clustering
3.	Test case suites [34]	The proposed approache approach aids in functional analysis of test suites and divide them into correctly manner.	Clustering

2.5 Software Quality:

Training samples and data sharing from other organizations are often required by engineers for project estimation and predictions. However, apart from publically available datasets, other datasets are very rare, majorly because of privacy concerns. If the dataset is published disclosing sensitive attributes, adversaries can access sensitive attributes in the data.
There are already existing privacy-preserving algorithms, but the quality of data confiscated decline inconsistent data. Swapping based method replace the values, but the performance is unstable. Clustering based methods replace values from another value within the cluster. The research paper [19] focuses on maintaining the utility of the privatized date and how effectively the privacy is preserved. The approach is to divide the data into several sub classes by using Interval Covering based Subclass Division (ICSD) division strategy and then obfuscate it by using proposed Manifold Learning based Bi-directional Data Obfuscation (MLBDO) algorithm. The major focus of the paper is to protect the privacy of samples in each subclass while keeping the data utility simultaneously. The testing is performed on seven different datasets including NASA93, Kitchemann, Kemerer, Coc81, China, Albrecht, and Maxwell. The evaluation parameters are Increase Privacy Ration (IPR) for a measure of privacy and Median Magnitude of Relative Error (MdMRE) for a measure of utility. The results are also compared with four other works including k-anonymity, swapping, clustering with MLBDO and ICSD combined with MORPH. It is proved that the privacy and utility parameters i.e. IPR and MdMRE of proposed methodology (ICSD and MLBDO) perform better than the compared works.

3. Discussion:

It is noticeable in the preliminary review that the defect/fault-related research dominates the usage of ML techniques to solve the problem. Most of the research used ML to classify and predict defect by using classification algorithms. Based on the research context, the input for the classification process is not only from the source code itself but also involve other information such as bugs report, developer/reviewer profile, test cases etc. Furthermore, MI is a black box and it is hard to explain the result. It is challenging for researchers, and more importantly practitioners, to have any clear idea and logic of choosing the technique for defect prediction and how well they are likely to perform as mentioned in [27]. There is also another debate on whether classifiers used in the researches are transferable to new settings or they are context specific. The other debate was using supervised classifier vs unsupervised classifier. There can be a better way in regard to automating the requirements engineering. The main constraint in this research area is data. The current researchers have also mentioned that due to the privacy, data can’t be shared. The other research gap is focusing on the fault localization as compared to fault prediction. It will help to save time and efforts.Software testing more specifically in regression testing ML can facilitate more. A new direction to this research can be given by adding more metrics for determination of inputs. Also identifying and finding a more automated way to aid the test traces. It will help to overcome the large number of test cases to the selected test cases.

As data is one of the valuable items in ML, a lot of researchers used publicly open data set such as provided by NASA and PROMISE, eclipse. Although public data may offer replication and standardization, this kind of data is only limited to certain domains. Furthermore, sometimes an explanation of the data is needed to enhance the results.

4. Threat to validity:

Since Machine learning is an emerging domain for problem solving in SE, it is expected to find a lot of research regarding the use of ML in SE. However, this study does not illustrate the completed research review. There is not much research found in this review that might cause by:

- The search result from main three SE conferences. There is need to focus on the ML application focused researchers for getting the view of the search. Additionally, more conferences and journals can be added for the extension of the research.
- The chosen keyword is based on the SE areas and ML algorithms.
- This search was done manually with consideration of the keywords, reading abstract. Several papers do not have these keywords in their title or abstract but actually, use ML techniques. Because of the manual searching and absence of the keywords, some papers could be overlooked.

5. References:

1. Xiaofan Chen, John Hosking, John Grundy. “A Combination Approach for Enhancing Automated Traceability”. In ICSE, 2011.
2. Chris Mills. ”Automating Traceability Link Recovery through Classification”.In ESEC/FSE, 2017.
3. Rafael V. Borges, Artur d’Avila Garcez, Luis C. Lamb, Bashar Nuseibeh.”Learning to Adapt Requirements Specifications of Evolving Systems”.In ICSE, 2011.
4. Paige Rodeghero, Siyuan Jiang, Ameer Armaly, and Collin McMillan. “Detecting User Story Information in Developer-Client Conversations to Generate Extractive Summaries”. In ICSE, 2017.
5. Preethu Rose Anish, Balaji Balasubramaniam, Abhishek Sainani, Jane Cleland-Huang, Maya Daneva, Roel J. Wieringa, Smita Ghaisas. “Probing for Requirements Knowledge to Stimulate Architectural Thinking”. In ICSE 2016.
6. Jin Guo, Jinghui Cheng and Jane Cleland-Huang. “Semantically Enhanced Software Traceability Using Deep Learning Techniques”. In ICSE17
7. Tim Menzies. “Practical Machine Learning for Software Engineering and Knowledge Engineering”. Handbook of Software Engineering and Knowledge Engineering Volume 1, 2001.
8. Du Zhang, Jefferey J.P.Tsai. “Machine Learning and Software engineering”. Software Quality Journal, 11, 87–119, 2003.

9. Tassey. “The economic impacts of inadequate infrastructure for software testing. Technical Report Planning Report 02-3”, National Institute of Standards and Technology, May 2002.
10. Jaechang Nam, Sinno Jialin Pan , and Sunghun Kim. “Transfer Defect Leanring”. In ICSE 2013.
11. Feng Zhang, Quan Zheng, Ying Zou, Ahmed E. Hassan. “Cross-project Defect Prediction Using a Connectivity-based Unsupervised Classifier”. In ICSE 2016.
12. An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. “Combining Deep Learning with Information Retrieval to Localize Buggy Files for Bug Reports”. In ICSE 2015.
13. Jaechang Nam and Sunghun Kim. “CLAMI: Defect Prediction on Unlabeled Datasets”. In ASE 2015.
14. Ferdian Thung, Pavneet Singh Kochhar, and David Lo. “DupFinder: Integrated Tool Support for Duplicate Bug Report Detection”.In ASE’14, 2014.
15. Chengnian Sun , David Lo , Siau-Cheng Khoo , Jing Jiang. “Towards More Accurate Retrieval of Duplicate Bug Reports”. In ASE’11, 2011.
16. Kihong Heo, Hakjoo Oh, Kwangkeun Yi “Machine Learning Guided-Selectively Unsound Static Ananlysis”. 2017 IEEE/ACM 39th International Conference on Software Engineering(ICSE).
17. Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, Vincent Ng. “Detecting Missing Information in Bug Descriptions”. In FSE 17.
18. Xiao-Yuan Jing, Shi Ying, Zhi-Wu Zhang, Shan-Shan Wu, Jin Liu, “Dictionary Learning Based Software Defect Prediction”. In ICSE 2014.
19. Fumin Qi, Xiao Yuan Jing, Xiaoke Zhu, Fei Wu, Li Cheng. “Privacy Preservation via Interval Covering Based Subclass Division and Manifold Learning Based Bi-directional Obfuscation for Effort Estimation”. In ASE ’16.
20. Haroon Malik, Hadi Hemmati, Ahmed E.Hasaan. “Automatic Detection of Performance Deviations in the Load Testing of Large Scale System”. In ICSE 2013.
21. Tim Menzies, Robyn Lutz, Carmen Mikulski. “Better analysis of Defect Data at NASA.” SEKE2003, The Fifteenth International Conference on Software Engineering and Knowledge Engineering.
22. Brittany Johnson, Yoonki Song, Emerson Murphy-Hill,Robert Bowdidge. “Why Don’t Software Developers Use Static Analysis Tools to Find Bugs?”. In ICSE 2013.
23. Qi Luo. “Automatic Performance Testing using Input-Sensitive Profiling”. In FSE 2016.
24. Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. “Data Quality: Some Comments on the NASA Software Defect Datasets”. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 39, NO. 9, SEPTEMBER 2013.
25. Ming Wen, Rongxin Wu, Shing-Chi Cheung. “Locus: Locating Bugs from Software Changes”.in ASE 16.
26. Jean Petric,David Bowes,Tracy Hall,Bruce Christianson,Nathan Baddoo. “The Jinx on the NASA Software Defect Data Sets”. EASE ’16, June 01-03, 2016, Limerick, Ireland.
27. Martin Shepperd, David Bowes, and Tracy Hall. “Researcher Bias: The Use of Machine Learning in Software Defect Prediction”. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 40, NO. 6, JUNE 2014.
28. Wei Fu, Tim Menzies “Revisiting Unsupervised Learning for Defect Prediction”, In FSE 2017.

29. Yibiao Yang, Yuming Zhou1, Jinping Liu, Yangyang Zhao, Hongmin Lu, Lei Xu, Baowen Xu, and Hareton Leung. "Effort-Aware Just-in-Time Defect Prediction: Simple Unsupervised Models Could Be Better Than Supervised Models". In FSE 2016.
30. Mojdeh Golagha, Alexander Pretschner,Dominik Fisch, Roman Nagy. "Reducing Failure Analysis Time: An Industrial Evaluation". In ICSE 2017.
31. Benjamin Busjaeger, Tao Xie "Learning for Test Prioritization: An Industrial Case Study".In FSE 2016.
32. Cu Duy Nguyen, Paolo Tonella. "Automated Inference of Classifications and Dependencies for Combinatorial Testing". In ASE 2013.

33. Giuseppe Scanniello, Carmine Gravino, Andrian Marcus, Tim Menzies. "Class Level Fault Prediction using Software Clustering." In ASE 2013.
34. Marcel Zalmanovici, Orna Raz, Rachel Tzoref-Brill. "Cluster-Based Test Suite Functional Analysis". In FSE 16.