Performance Analysis of Machine Learning Algorithms to Predict Mobile Applications' Star Ratings via its User Interface Features

By

Maryam Navaei

May 2022

Director of Thesis: Dr. Nasseh Tabrizi

Major Department: Computer Science

ABSTRACT

The first part of this thesis concludes an overall summary of the publications so far on the applied Machine Learning techniques in different phases of Software Development Life Cycle that includes Requirements Analysis, Design, Implementation, Testing, and Maintenance. We have performed a systematic review of the research studies published from 2015-2021 and revealed that Software Requirements Analysis phase has the least number of papers published; in contrast, Software Testing is the phase with the greatest number of papers published.

The second part of this thesis compares multiple Machine Learning algorithms on predicting mobile application star ratings by its user interface features. User interface features offer great source of information that can be utilized by various Machine Learning algorithms to generate this prediction. To do so, we have developed and selected multiple user interface features extracted from the largest mobile user interface design prediction dataset that is available to public, RICO repository. We initially employed the Machine Learning algorithms to a subset from RICO and then compared our results against the actual dataset using the same algorithms. Furthermore, we calculated Accuracy, Recall

and Precision for each algorithm before and after cross validation, and showcased our results in various charts. The ultimate results demonstrates that our methodology works to predict the star rating of Android mobile applications utilizing the features we extracted from RICO dataset.

Performance Analysis of Machine Learning Algorithms to Predict Mobile Applications' Star Ratings via its User Interface Features

A Thesis

Presented To the Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Software Engineering

by

Maryam Navaei

May, 2022

Performance Analysis of Machine Learning Algorithms to Predict Mobile Applications' Star Ratings via its User Interface Features

By

Maryam Navaei

APPROVED BY:

Director of Thesis

Dr. Nasseh Tabrizi

Committee Member

Dr. Rui Wu

Committee Member/Chair of the Department of Computer Science

Dr. Venkat Gudivada

Dean of the Graduate School

Dr. Paul J. Gemperline

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Humans have been using many types of tools to accomplish various tasks in a simpler way. The creativity of the human brain led to the invention of different machines. These including travelling, industries, and computing. And AI is the one among them [63].

The AI is changing the world by transforming all sections including health care services, education, transport, food, entertainment, and different assembly line and many more. It will impact lives in almost every aspect, including marketing and sales, e-commerce, social media, transportation, financial services, oil and gas, manufacturing, government etc.

ML has arisen as the method of choice for developing helpful software systems for computer vision, speech recognition, robot control, natural language processing and other applications [66]. ML techniques have developed quickly in the perspective of computing with smart mobile phones that warrants the devices to operate in an intelligent mode [64]. Recent developments in ML have motivated extensive interest within the Information Technology on incorporating AI capabilities into software and services. This ambition has enforced organizations to advance their development processes [19].

ML helps developers to build more efficient software systems that improve their performance by experience [64]. The impact of AI and ML techniques is proven in software requirement analysis, software architecture design, software implementation, software maintenance and software testing.

Subsequently in recent years the smartphone industry has made significant evolution in mobile applications as these devices are well known for Internet-of-Things (IoT) and are also called next generation multi-functional smartphones. AI plays a significant role and is uninterruptedly unleashing its power to create intelligent mobile applications and its impact can be seen in personalized recommendations, virtual assistant, mobile business, healthcare services and even in COVID management [64][65]. ML can bridge the gap between user behavior and utilizing this to create a customized service based on user's activities. This is a major progress which enables developers to create ideal applications for users. Predictive analysis helps with processing large amount of data and ultimately generating quantitative predictions according to user's needs. ML also helps developers with adding security and filtering spam emails. Natural language processing and character recognition helps with development of applications that can be used for reading and translation. ML and Data analysis is essential in e-commerce systems, in the improvement of product and service search results etc.

1.1 Research Contribution

In this research work, we initially conducted a systematic literature review about ML techniques applied on each phase of SDLC. The research is done based on publications extracted from IEEE, ACM and Springer during the period of 2015 to 2021. We studied significant number of publications from each database to discuss statistically what phases of SDLC focused more on utilizing ML algorithms and why.

Ultimately, we selected the four most popular ML algorithms according to the numbers extracted from all the phases to generate a performance analysis between those methods separately for each phase of SDLC. Our results are shown in different figures to demonstrate what algorithm outperformed the others in each phase.

1.2 Thesis Structure

The structure of this thesis is as follows. Chapter represents the systematic literature review on ML techniques in Software development life cycle. Chapter 3 provides the related work on the impact of ML algorithms in user interface design prediction for mobile application and discussing how it helps developers to create more appealing user interface to increase customer satisfaction which then leads to higher star rating. Chapter 4 describes the proposed methodology in detail. We then studied to discuss if we can predict the mobile application start rating by its user interface features. Chapter 5 provides the performance analysis of our models using the most extensive public dataset for mobile user interface design prediction, Rico dataset. Last but not least, we conclude the current research, conclusion and discuss future research contribution in Chapter 6.

Chapter 2

Systematic Review

2.1 Introduction

As Software Engineering has become an essential industrial domain and software systems have become increasingly complex due to the performance requirements of their operating environment, the ability to design, develop, maintain, and adapt these systems has surpassed human comprehension alone [29]. Therefore, software systems need to autonomously adapt to changing environments to guarantee expected quality of service [28].

The AI has a long tradition in computer science. Since 1950 in its first three decades, the academy and industry worked diligently to reach to human level machine intelligence. In retrospect that was an overly optimistic expectation [15]. Nevertheless, the need for more automation and intelligence have led to further advances in ML and AI. Researchers are increasingly applying ML and AI to remediate failures and inadequacies in the software systems and the SDLC [3]. ML is a branch of AI that uses data or former development experience to enhance the performance standards of software systems [41][42].

Traditionally, software systems are built deductively, by implementing the rules that administer the system performances as program code. However, with ML techniques, these rules are collected from training data. Conventionally, ML methods can learn a model's parameters automatically using training data and thus it can create models with respectable performance that can satisfy the software system requirements. ML has

achieved great success in challenging real-world AI and data mining scenarios, such as object detection, natural image processing, autonomous car driving, urban scene understanding, automatic machine translation of human speech, and web search/information retrieval, among others [44][45][46].

ML enables computer systems to use data, examples, and experience without human involvement but to replicate some form of human intelligence. ML algorithms are utilized to help computer systems learn rules from data [31]. These technologies will be optimized to be the core components in a variety of software-intensive systems. Recent advances in ML have fostered extensive interest in the Information Technology industry to integrate AI capabilities into software and services. Systems employing ML technologies, have distinctive characteristics moderately distinctive from those of the other software systems. Functionalities of ML based systems heavily depend on the quality of training datasets used to create the predictive models. Changing the training dataset has significant impact on learning results and thus on functional behavior of programs [19] [43].

ML is used to continuously improve system performance and efficiency on a particular task by helping computer systems to learn from experience how to perform the job autonomously. ML plays a significant role in SDLC and has been used in variety of domains [1][30][32]. For example, these domains include defect prediction, requirements elicitation etc. Therefore, ML is becoming one of the most important technologies used in SDLC. It is being applied to poorly understood problem domains where little domain knowledge currently exists for the humans to develop effective methodologies. These scenarios include data mining large databases containing valuable but undiscovered

implicit regularities, and domains where software systems must adapt to changing conditions [3]. ML has become the preferred technique for developing practical Software Systems in Computer Vision, Speech Recognition, Natural Language Processing, Robot Control, Health Systems, Banking, Defence, E-Commerce and many more environments [27][40]. Conventionally, developing software systems necessitates on specifying the requirements in advance to define how the system should behave.

ML techniques are being used to accelerate SDLC and has been used in many areas including behavior extraction, testing, and bug fixing [5][6]. This has led to a new paradigm in technology invention. During the SDLC, the development team must spend significant time discussing how the system must operate to decide which features should be prioritized and which ones need to be eliminated. This Requirement Analysis needs to be followed by Design, Implementation, Testing and Integration, and Maintenance phases [4].

By utilizing ML algorithms, software developers can accelerate the decision-making process based on the previous projects to create data-driven business decisions to ultimately help decrease the development risks and costs. This is invaluable for software companies. Since projects can go over costs and deadlines, coming up with a precise budget and schedule to develop software systems can be an overwhelming task. Therefore, accurate estimation of efforts is highly crucial for Software Project Planning. With the help of ML techniques, the development team can analyze data from past projects to provide a more precise budget estimate and delivery time frame [7].

ML also has a great impact in decreasing the time spent on prototyping as well as the number of expert engineers required to develop the software. In Software Testing, with

the help of ML algorithms, testers can come up with more accurate results to reduce the errors in the system. ML techniques also help accelerate the process of finding defects. Since clean code is crucial for effective software maintenance, large scale code refactoring is unavoidable.

Utilizing ML techniques, developers can review the code and maintain it automatically. Since maintenance is one of the most critical yet expensive phases of SDLC, this results in developing high quality Software Systems. Traditional SDLC can benefit from ML models for rapid prototyping, intelligent programming assistants, automatic analytics and error handling, automatic code refactoring, and precise estimates and strategic decision-making. Given the significant role that ML techniques and AI-based systems can play in the development of Software systems, it is very important for both the Software Engineering and ML communities to research and develop innovative approaches to address the advantages and disadvantages [3][32]. Although ML datasets for SDLC analysis are often poorly documented and maintained and do not have clear creation processes [17] there are many advantages in utilizing ML techniques. Because of the significant role of ML techniques in SDLC and their advantages, we have prepared an overall review to see the impact of data analytics and ML algorithms on each phase of the SDLC. We were interested to identify phases in which ML models are widely being utilized. Our goal was to see why researchers focused more on certain phases than others when it comes to utilizing ML methodologies. We also wanted to see if ML techniques can be applied to less popular phases as well.

To prepare our review, we collected and compiled a database of almost 150 journal articles and conference papers retrieved from ACM, IEEE and Springer digital libraries

within the past five years. We created figures to demonstrate visually and summarize current research trends. Our research uncovers significant differences in the use of ML methodologies in various aspects of Software Engineering and the SLDC (e.g., Requirements Analysis, Design, Implementation, Testing and Maintenance). After a discussion of related work and our methodology, we discuss our findings for each phase of SDLC. We then conclude with a summary of our work and suggestions for further research.

2.2 Systematic Review

In this paper we review current research on applications of ML to different phases of SDLC and SE generally. Because of the popularity of incorporating ML techniques into SDLC, several prior researchers have conducted empirical studies of SE for data science. One study focused on differences between the development of ML systems and non-ML systems in various aspects of SE. That study used interviews to identify pain points from a general tooling standpoint to discover the challenges and obstacles of implementing visual analytic tools as none of the SE phases have a developed set of tools and methods [2][18]. Another study focused on characterizing professional roles and practices regarding data science [28].

ML Software systems are challenging to test and verify. Sometimes the ML model may be incorrect even if the learning algorithm is executed properly due to inaccurate training data. Traditional testing techniques are insufficient for such systems. Therefore, it is crucial for Software Engineers to research and develop advanced ways to report

these challenges [3]. Software Engineers focused on the challenge of verifying accuracy of systems built using ML and AI models and testing those systems.

Multiple previous literature reviews have been written on applying ML algorithms to each phase of SDLC [22][23] [34][35]. We came across a publication that provides a broader context that discusses the relationship of ML techniques and its tools within SDLC stages which is related to our work to some extend however their review was performed for the period of 1991 to 2021 [30]. Nonetheless, our paper utilizes a systematic review methodology to study the impact of ML across the entire SDLC in the most recent publications (2015-2021) as opposed to focusing on a single phase as seen in most publications. Our research also makes an additional contribution by identifying the four most popular ML algorithms in overall SDLC. We studied those four techniques in each phase independently to observe their overall popularity based on each phase of life cycle.

2.3 Methodology

Initially, we intended to focus on publications for the past decade (2010-2020). Due to overwhelming number of publications, we finally decided to focus our review on the journal articles and conference papers published in the past five years. We realized the application of ML in SDLC considerably increased in the past ten years where it had a slow start from 2010 to 2015 and then significantly boosted from 2015 onward based on the number of publications, because recent advances in ML greatly impacted SDLS as ML techniques can change and update software systems.

We used different keywords and queries to retrieve our results from ACM, IEEE and Springer databases. Some queries gave us our desired results in IEEE and Springer

whereas we had to modify our keywords and queries to retrieve relevant publications from ACM. Additionally, we found out that ACM would return significant number of irrelevant publications when using verbatim queries utilized in the other two libraries. Therefore, we had to apply more constraints when searching in ACM database. For instance, we had to include ‘Artificial Intelligence’ as a keyword to ensure our query retrieved relevant results.

There were situations where our queries returned similar or enhanced work in more than one publication, we treated each of those as a separate publication since each focused on a unique topic. Also, there were situations where one ML technique was utilized in multiple phases, we counted those separately for each phase. The latter scenario led us to study the four trendiest ML techniques in each phase furthermore to have a more focused observation.

2.4 Current Applications of Machine Learning to SDLC

ML plays a significant role throughout the SDLC. In this section we discuss the impact of applying ML in each phase of the SDLC:

- Software Requirements Analysis
- Software Architecture Design
- Software Implementation
- Software Testing
- Software Maintenance

We provide an overview of each phase of the SDLC and discuss current research trends in ML for each phase.

2.4.1 Software Requirements Analysis and Machine Learning

Software Requirements Analysis (also called Requirements Engineering) clarifies the specifications needed for a software system to satisfy the business requirements. High quality software development strongly depends on clearly defined Software Requirements (SR) that explains the needs and expectation of the software in a very detailed form [10]. These requirements must be documented, measurable, testable, traceable and related to the business needs.

Requirements Engineering (RE) plays an important role in the overall process of Software Development and consists of two main phases: Requirements Identification and Prioritization [34]. Requirements Analysis is critical to the success or failure of a Software system. Software Engineers must also perform Requirements Classification during the Analysis phase. Requirements Classification is cumbersome, time consuming, and difficult when performed as a manual process. RE consists of four parts: Requirements Elicitation and Discovery, Requirements Specification and Analysis, Requirements Validation and Requirements Management [9]. However, after utilizing ML techniques, Software Engineers were able to automate the process of grouping the requirements into Functional Requirement (FRs) and Non-Functional Requirements (NFR) [33]. RE is also one of the key factors in Software Quality [27].

In SDLC, requirement analysis is the process of correct requirement gathering, the efficient examination of collected requirements, and clear requirement documentation. As software systems increase in complexity and scale, Requirement Engineering becomes a challenging issue that leads to increased development costs. This has led to engineers

showing additional interest in automatic requirement analysis techniques which can lead to more accurate and rapid analysis of SR and reduction in development costs. ML methodologies are being used for Requirement Prioritization where engineers need to decide which set of requirements to be considered first and in requirement specification. ML algorithms are likely to classify and prioritize the requirements efficiently and how they can be evaluated [34].

Researchers have shown a lot of interest in applying Supervised ML algorithms including Support Vector Machine, Naïve Bayes, Decision Tree, K-Nearest Neighbor and Random Forest when their focus is on one of the following broad categories [11]:

- Detection of linguistic problems in requirements documents and artifacts written in natural language
- Classification of document content
- Requirement traceability
- Effort estimation
- Requirements analysis
- Failures Prediction
- Quality of and detection of business rules.

Techniques used in Software Requirements Specification (SRS) approach are Natural Language Processing (NLP) and Information Retrieval (IR) [10]. ML provides approaches that use big data to enable a ML algorithm to learn from producing outputs, which would be difficult to obtain otherwise [12]. RE plays a key role in the success of a

project. Based on previous research that was conducted on 350 companies to understand project failure rates, 16.2% projects were completed successfully, 52.7% faced challenges and were partially completed. About 31% of the projects were never completed due to poor RE [55].

One main limitation with public ML datasets is that there are a limited number of such datasets available for Requirements Analysis. One of the most used RE databases is the PROMISE repository, which is unbalanced and has only 625 classified requirements written in natural language [9]. Despite limitations of current ML algorithms in recognizing and prioritizing requirements, including scalability, dependency, and complexity [34], what prompts researchers to show continued interest in utilizing ML algorithms in SR classification is the significant role that these techniques play in helping developers document their software with more precision and validation. This makes the software system easier to understand and use [33].

2.4.2 Software Architecture Design and Machine Learning

The Architecture Design phase (also called the preliminary or general design stage) deals with converting the defined requirements from the Analysis phase into an implementable form. Software design has a major impact on software quality. Examples of bad design include anti-patterns, high dependency design, and massive source code files. These make SE tasks more difficult by exacerbating the challenges and expense of fixing software defects [38][39].

This phase can be divided into three categories: Interface Design, Architectural Design and Detailed Design. Interface Design specifies interaction between a system and

its users. In the Architectural Design, the specification of major system components, their responsibilities, relationships, and their interactions are defined. During Detailed Design the specification of internal elements ¬of all major system components along with their properties, relationships and data structures are determined. Evaluation of system architecture is a turning point in the decision-making process. It aims at justifying the extent to which decisions made during Architecture Design satisfy a system's quality requirements. Specifically, when there are functional uncertainties and changing requirements. Architecture Evaluation facilitates the process of identifying and minimizing the design risks that save integration, testing and evolution costs in software systems [35].

Software Design should be clear and understandable to meet all system requirements. Since subsequent phases of SDLC significantly rely on the Architecture Design phase, many companies and researchers show interest in understanding the relationship between Software Design and Maintenance. Researchers have been working on introducing various ML techniques for efficient prediction of Software Design's impact on maintainability of a system [37].

Several ML techniques are being used in this phase on criteria such as predicting the design for a mobile application's user interface (UI), predicting the architecture for safety critical systems, web service anti-pattern detection, etc. For instance, design patterns can add complexity. This leads to an increase in maintenance and evolution efforts. Using ML for design pattern detection decreases the system complexity and results in increased understandability of the software's architecture and design [36][14][16].

Code Smells, Call Dependencies and Lines of Code can be indicators of poor architecture and design [39]. Code Smells are not the errors in implementation; rather they are weaknesses in the design of part of the software system that can either impede the development process or increase the chance of system failure and errors in the future. Detecting Code Smells could lessen the work of developers, resources and cost of development [13].

Since detecting Code Anomalies requires refactoring approach, Support Vector Machine is the most effective ML algorithm used for this matter [11]. Design patterns are a known reverse engineering technique to solve numerous problems in the design phase; they can be mined from source code of similar category software. Data mining from design patterns, which is based on supervised learning and software metrics, is another most popular research area in Design phase. ML techniques are being used to enhance pattern mining by excluding as many false matches as possible. ML algorithms such as Layer Recurrent Neural Network, Random Forest, Support Vector Machine and Boost are mostly used in this process [10][18].

2.4.3 Software Implementation and Machine Learning

The Implementation phase involves construction of the actual software system. Coding the system takes place in this phase and when it is complete, the result will be evaluated against the elicited requirements from the Analysis phase. Software defects are prevalent in software development and might cause several problems for users and developers alike. As a result, research has examined distinct techniques to diminish the impacts of these defects in the source code. One of the most prominent algorithms focuses on defect

prediction using ML methods. This helps developers in managing these defects before they are commenced in the production. [8].

Code refactoring is one of the popular research areas for this phase. Refactoring changes the system in such a way that does not alter the external behavior of the code but will improve its internal structure. Software containing code smells indicates a violation of design and coding practices in SDLC. This issue can turn into a larger problem as the effort to remove the errors will increase exponentially if code smells are left unremedied [20].

ML algorithms can precisely model the refactoring recommendations. Process and ownership metrics both play a significant role in the creation of highly effective models. In addition, models trained with data from heterogeneous projects generalize better and achieve great performance [21]. God Classes, Long Methods, Functional Decomposition and Spaghetti Code are the most frequently identified code smells [56].

Supervised Machine Learning algorithms are known to be the most effective approach in predicting code refactoring, as these will help developers to make faster and more-educated decisions considering what to refactor. These ML techniques evaluated included Support Vector Machines, Naïve Bayes, Decision Trees, Random Forest, Logistic Regression and Neural Networks. Random Forest has emerged as the most accurate approach in predicting software refactoring using Apache, F-Droid and GitHub datasets [58].

2.4.4 Software Testing and Machine Learning

In order to verify and validate Software Systems, engineers need to find a system's faults and defects. Software Testing is thus one of the most important phases in SDLC. The objective of Software Testing is to reveal existing faults, particularly during automated testing [47]. Due to constant changes to the codebase of a system, errors, failures, and defects can happen in system behavior, there should be automated tests to identify these errors before a system is deployed. Improvements in system infrastructure by finding its faults could save up to third of its costs [23].

Testing helps developers ensure the behavior of a system is working as specified. Testing efforts must be predicted in advance to guarantee time efficiency, effort, and cost usage [22]. In general, Software Testing is an inductive inference in which the tester determines the general properties of a software system by carefully studying the behavior of the system on a finite number of test cases [26].

As software systems increase in complexity, it becomes more difficult to detect faults [24]. We can collect numerous kinds of data during software testing process, including execution traces, coverage information for test cases, failure data etc. ML techniques applied during Software Testing identifies the patterns in data that often feature about data generation process and will be used for decision-making. Experts need to provide inputs to transform raw test data into abstract test cases [25].

Test cases are generated to provide test output data. Decision trees or induction rules are widely used in Software Testing especially in fault prediction. Models produced by these techniques are simpler to interpret compared to techniques that are more sophisticated such as neural networks or support vector machines. Support Vector Machine and Random Forest provide best prediction models for these datasets (the most widely used Testing dataset is from NASA) in fault prediction [25][26].

When testing software, there are considerable advantages to be gained from techniques which propose unusual interactions within a system. Techniques such as Random Testing, Fuzzing and Exploratory Testing have a disadvantage, however, in that the outputs of the tests need to be manually checked for correctness which adds additional effort for Software Engineers [48]. Currently testing techniques and software reengineering are critical and most crucial in determining software usability [49][50].

Due to the quantifiable nature of Software Testing, there are many datasets available. Likewise, the number of publications is also significantly higher compared to other phases.

2.4.5 Software Maintenance and Machine Learning

The process of maintaining a software system after it has been delivered, modifying, and updating it to correct defects is called Software Maintenance. This is the final phase of SDLC. Software Maintenance has become an important area in SE related to software quality. Consequently, predicting this characteristic in an accurate and timely manner is a crucial requirement for efficient management during the final phase of SDLC [51].

Software Maintenance incorporates repair, preservation, and continuing optimization of a system. This phase has four different categories: Preventive Maintenance, Corrective Maintenance, Adaptive Maintenance and Perfective Maintenance [57]. As change is inevitable, engineers must develop techniques for evaluating, controlling, and making modification. Software Engineers need to be capable of predicting software maintainability in time, ensuring a constrained and optimum use of the available resources. A large number of ML techniques, including hybrid techniques, ensemble techniques, and meta-heuristic techniques have been explored for Software Maintainability Prediction (SMP) [51][52][53]. Yet researchers continue to seek improvement in this area.

A significant amount of time sometimes about 60-70% in SDLC total cost is dedicated to maintenance purposes [51]. As software systems are getting more complex and increasing in size, maintenance has become increasingly difficult. Therefore, software maintainability has turned into a serious challenge that companies are dealing with. Software maintainability is directly connected with the financial implementation and project success [54].

Software Maintenance is classified based on three viewpoints: intention-based classification, activity-based classification, and evidence- based classification. Software Maintenance involves bug fixing, upgrading, and enhancing the software program.

Many ML and AI techniques have been used in Software Maintenance. Yet, there are not sufficient datasets available as input data. Among available datasets, their size is not large enough to give accurate results. More work is needed in this area. Fuzzy logic is the mostly frequently used ML technique in Software Maintenance predictive models

[55]. Other techniques include ML algorithms such as Soft Computing, Fuzzy Networks, Evolutionary Algorithm, Deep Learning and ID3 [59].

2.5 Chapter Conclusion

We include publications from 2015 to 2021 in this review. The figure below summarizes how many journal articles and conference papers are published in ACM, IEEE and Springer for the period of 2015 to 2021 that focus on ML techniques applied in each phase of SDLC.

Table 1 lists the keywords we used to retrieve these publications and ultimately generate the figures to show the overall results. For some of the phases, we had to modify our search queries as it was a quite challenge to find more related and accurate results. Whereas some phases like Testing, had a very large number of publications.

Table 2-1 Searched Keywords for Paper Retrieval

Keywords (Searched Queries)
Software engineering AND Machine learning
Software Development Life Cycle AND Machine Learning
Machine Learning AND Software System Requirements Analysis
Machine Learning AND Software Design
Machine Learning AND Software Implementation
Machine Learning AND Software Testing
Machine Learning AND Software Maintenance
Impact of Machine Learning in Software Maintenance
Impact of Machine Learning in Software Architecture
Impact of Machine Learning in Software Requirements
Software Requirements Analysis AND Random Forest Algorithm
Software Requirements Analysis AND Support Vector Machine
Software Requirements Analysis AND Naïve Bayes
Software Requirements Analysis AND Decision Trees
Software Design Phase AND Random Forest Algorithm
Software Design Phase AND Support Vector Machine
Software Design Phase Analysis AND Naïve Bayes
Software Implementation AND Random Forest Algorithm
Software Implementation AND Support Vector Machine
Software Implementation AND Naïve Bayes

Software Testing AND Random Forest Algorithm
Software Testing AND Support Vector Machine
Software Testing Analysis AND Naïve Bayes
Software Maintenance AND Random Forest Algorithm
Software Maintenance AND Support Vector Machine
Software Maintenance Analysis AND Naïve Bayes
Software Maintenance AND Decision Trees

As shown in Figure 1, Software Requirements Analysis has the least number of papers published in this topic; in contrast, Software Testing has the largest number of publications. The reason is partially because there are many datasets available for Software Testing due to its quantifiable nature as well as the ease of obtaining data for this phase.

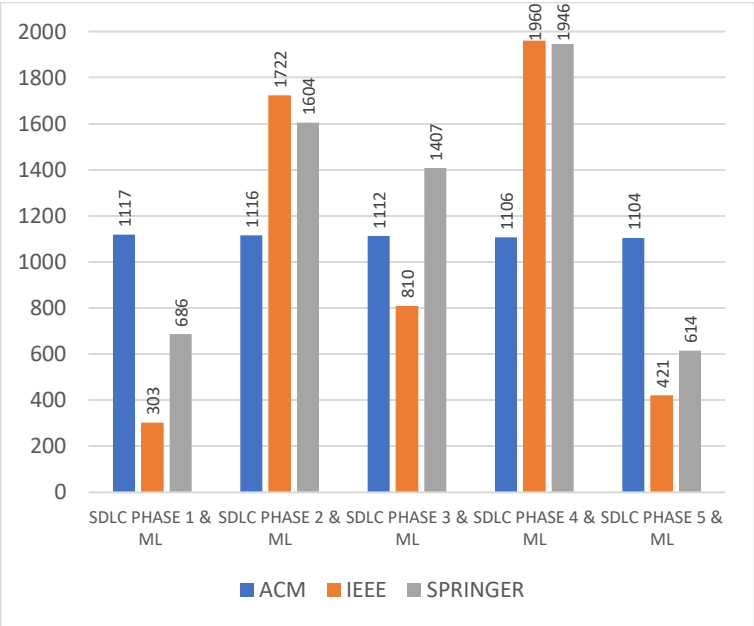

Figure 2-2: ML Applied in SDLC – Papers published from 2015 to 2021.

Hence it is much easier to utilize ML techniques in Software Testing than in Requirements Analysis.

In Figure 2, we narrowed our findings by selecting the four most used ML algorithms in all phases of SDLC (Random Forest, Support Vector Machine, Naïve Bayes and Decision Trees). Ultimately, we analyzed all the relevant publications done in ACM, IEEE and Springer in the same timeframe to see which ML methodologies are the most popular in particular phases.

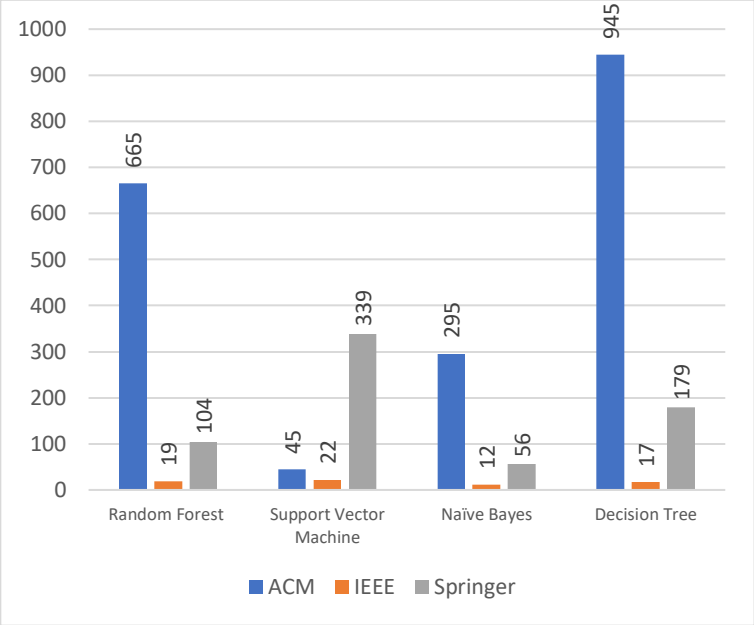

Figure 2-3: SDLC Software Requirements Analysis - ML Applied in SDLC – Papers published from 2015 to 2021.

Based on the results shown in Figure 2, Decision Trees are most popular algorithms among these four elected techniques. One reason is because Decision Trees are quick to create and help with eliminating dead ends. This decreases the probability of errors made in this phase that can affect subsequent phases as well. This technique can

greatly simplify the SR gathering process and saves engineers significant time and budget resources.

In the process of SR, due to their exclusive characteristics, the appropriate data sets are extremely dimensional, sparse, and are mostly the outcome of ambiguous expressions and, consequently, show problematic challenges for data processing techniques [62]. there are many repositories with duplicate code, which constitutes data inconsistency [61]. Hence, more work needs to be done on creating more diverse and appropriate datasets for this phase.

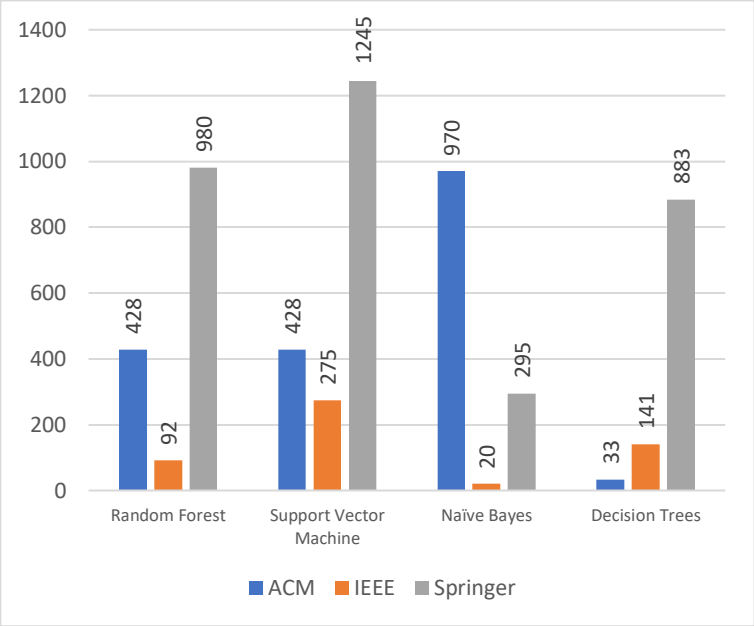

Figure 2-4: SDLC Software Architecture Design - ML Applied in SDLC – Papers published from 2015 to 2021.

Support Vector Machine (SVM) is one of the most popular ML techniques used in SDLC. It is a supervised learning algorithm used for both classification and regression problems. This technique is very accurate and robust in detecting design anomalies such

as code smells also known as Bad Smells, Design Flaws, Anti Patterns and Code Anomaly [60].

ML demands massive datasets to train on and these should be unbiased and have great quality which sometimes requires new data to be generated. We suggest applying ML algorithms more in Software architecture design as little work has been done and needs to be explored more [62].

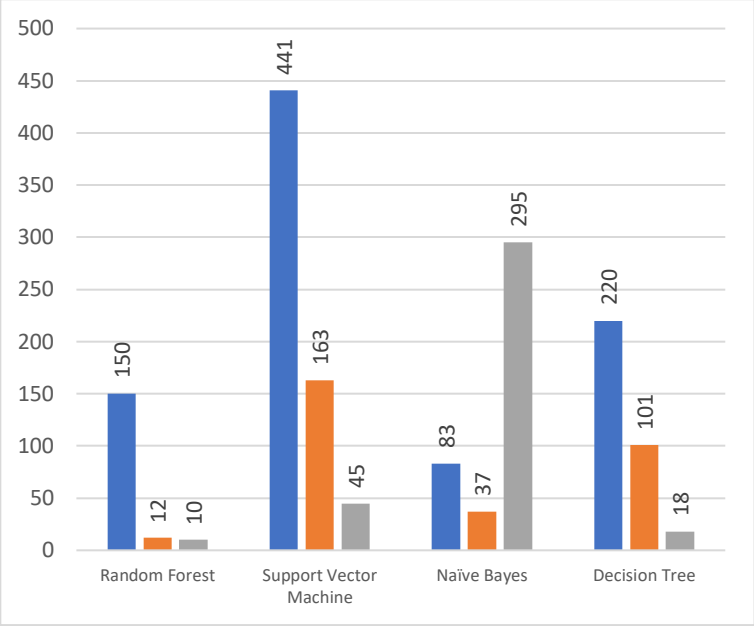

Figure 2-5: SDLC Phase Implementation - ML Applied in SDLC – Papers published from 2015 to 2021.

Again, Support Vector Machine is most frequent algorithm used in the third phase of Software Development Life Cycle known as Software Implementation. This technique is very popular in Design and Development of software applications.

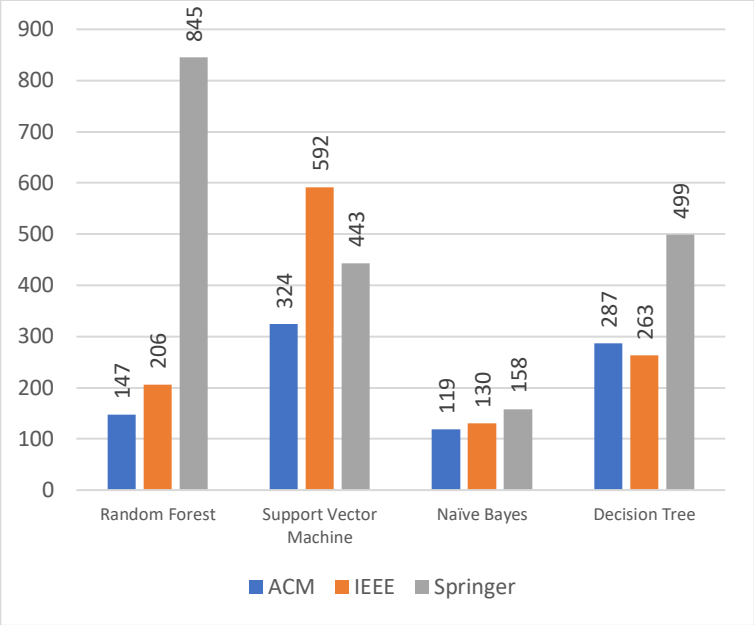

Figure 2-6: SDLC Phase Testing- ML Applied in SDLC – Papers published from 2015 to 2021.

As shown in Figure 5, both Random Forest and Support Vector Machine are popular techniques in Software Testing phase. However, SVM has slightly more publications. One reason for this is SVM is the better classifier to eliminate infeasible test cases saving time and costs in projects which is a huge achievement as testers spend more time and their resources on testing mobile and hybrid applications. ML helps testers to better understand user's needs and respond faster to the everchanging expectations by improving automation testing, reduced UI based testing, assisting in API testing, and improving accuracy.

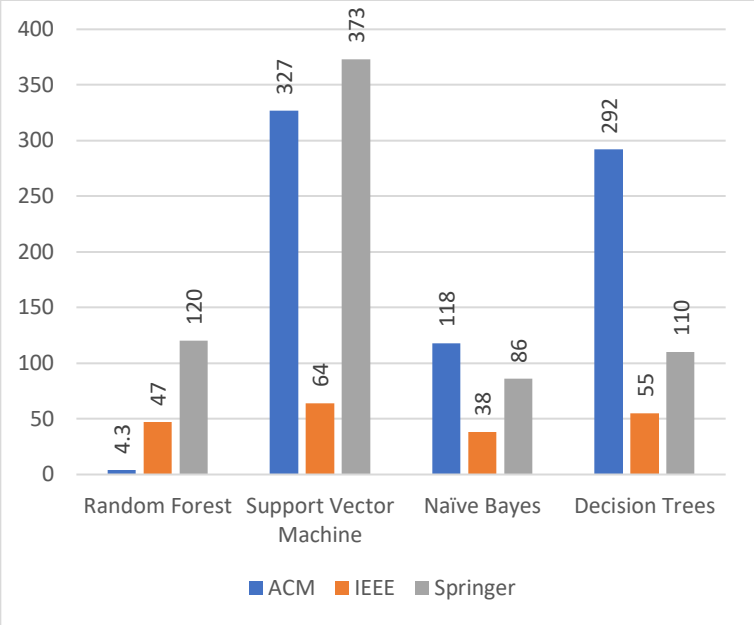

Figure 2-7: SDLC Maintenance - ML Applied in SDLC – Papers published from 2015 to 2021.

Data Scientists and Software Engineers have shown the most interest in applying Support Vector Machine in last phase of SDLC than other techniques. However, Decision Trees have high usage as well. Classification and refactoring approaches play a significant role when maintaining software systems. Available datasets are not sufficient to give conclusive results which provide information regarding the input data of the system [54] and more work need to be done to create new datasets to determine the accuracy of the precision and decreasing the complexity.

2.6 Research Direction

Machine Learning is becoming one of the most important technologies used in Software Development Life Cycle. ML approaches are being used for inadequately implied problem domains where little knowledge exists for humans to develop effective algorithms. ML has different types: Supervised Learning, Semi-Supervised Learning, Unsupervised Learning

and Reinforcement Learning. Our study shows Support Vector Machine (SVM) is one of the most popular supervised ML algorithms in current SE research. It is being applied on all the phases of SDLC as it can be used for both classification and regression problems. While SVM is not considered the best performing algorithm in all cases, it remains among the most highly used ML methods. Reinforcement Learning is among the least mentioned ML algorithms in SDLC. Although it has been used in Software requirements analysis for understanding the relationships between requirements at different levels of abstractions a few times, it is an interesting topic that we can study on more in terms of simplifying SE. Software Testing is utilizing ML techniques more than any other SDLC phase since there are a lot of datasets available to researchers.

Our future work aims at addressing the challenges outlined in Software Requirements Analysis, Architecture Patterns, Software Maintenance and ultimately creation of datasets for those phases. Also, future works could look at data mining, predictive design and modeling for different software applications including mobile applications.

Chapter 3

Related Works

The AI techniques have influenced smart mobile phones in many ways, one of its major impacts is in mobile applications. Due to rapid changing environments, these applications need to adopt and behave accordingly. Data Science improves mobile application development process, it is not only for assessing the data, but it runs more comprehensively leading to intelligent mobile applications that can predict and make decisions based on user's needs [64]. Its impact is categorized in the following:

- Analysis of User Experience

  Due to rapid growth of mobile application industry, developers need to know more about how users utilize their applications. By means of data science, they can conduct a detailed analysis of user experience of all features.

- Real-time Data Availability

  As client's needs change all the time and so quickly, they expect the companies to offer best solutions. Companies use data science to speed up the development process also to keep up with the continuous changes of end user's needs.

- Customized Marketing Campaigns

  Companies are now able to assess buying patterns, demographic data, and user's behavior for changing the marketing plan per their requirements utilizing data science techniques to increase application revenue, push adaptation and enhance satisfaction.

- Client Requirements

  Organizations use data science to understand the interaction and reaction of the users from various locations, background, education, and age groups to develop quality applications that meet user's requirements.

- Sales Conversions

  Data science plays an important role to provide developers with analyzing the data to ensure they can provide an enhanced solution that make it easier for them to include mobile application monetization models in it.

- Social Media Analytics

  Data science provides valuable information to organizations such as complaints, client reviews, dynamic media etc. knowing these helps developers to uncover enhanced ways to sell their products by knowing the end users better and their way of interacting with the application.

- Connected Devices

  IoT devices help collect data from users and evaluate them for actionable awareness using data science techniques which is incorporated in them. That will help developers build user friendly, result oriented and impactful applications.

Data science plays a critical role in mobile application development process, and due to its simple performance and advanced features, it has become a crucial part of mobile application industry. For instance, data Science in UX designing has the power to provide better user experience with interactive design, automated responses, and better personalization. A satisfied user is more likely to return and recommend the application to others. The positive reviews will motivate developers to further enhance the usability

of the product to build applications with good value and win the competitors. User experience should be in such a way to keep the clients. To generate positive user reactions, you must fulfill the basic usability principles of interface design:

- Learnability

  How easily first-time users can complete basic tasks?

- Efficiency

  How quickly users perform basic tasks once they've learned the design.

- Memorability

  The ability of users to remember how to use the system.

- Errors

  The amount and severity of errors users make and how easily they can correct them.

- Satisfaction

  How pleasant the experience of using the design was?

As mentioned previously, one of the aspects that plays a significant role in the success of mobile application is its User Interface (UI) design. UI is the process of improving the appearance and the interactivity of the application. It focuses on the creation and placement of all the items on the screen such as buttons, icons, or any visual elements that users interact with. There are some key principles involved in designing UI to ensure it is efficient and easy to use:

- Content Prioritization
- Make Navigation Perceptive
- Touchscreen Target Sizes

- Provide User Control
- Readable Text Content
- Make Interface Elements Clearly Observable
- Hand Position Controls
- Minimize Data Input
- Create a Smooth Experience
- Test Your Design

Now that we know the key principles of an effective design, we are going to discuss how data science impacts the design prediction in various ways. With the great help of data science, you may achieve a better balance between the application's data usage levels and the amount of data stored locally on the user's device. It detects areas for improving application speed, responsiveness, and overall user experience. By employing data science, developers may determine exactly how people are interacting with the UI, Evaluate the efficacy of the application's architecture, identify user tendencies and habits, focus QA testing efforts on the devices/OS versions that are most common amongst the user and finally implement shortcuts and architectural revisions to ensure the UI supports how people are operating the application's features and functions.

Chapter 4

Methodology

In this chapter, we provided the proposed methodology for extracting and incorporating UI features to predict mobile application's star rating. We selected our data from Rico design dataset which is the largest repository of mobile applications designs to date, created to support five classes of data-driven applications including design search, UI layout generation, UI code generation, user interaction modeling, and user perception prediction. The Rico dataset contains 73K UIs design mined from more than 9.7K free Android apps of 27 different categories during runtime via human powered and programmatic exploration [67][68].

We selected two subsets from existing 7 subcategories based on what we aimed to accomplish; first subset is UI screenshots and view hierarchies that contains over 66K unique UI screens. For each UI, Rico presents a screen shot in the format of PNG file and a detailed view hierarchy JSON file. The Hierarchy of JSON file starts with activity_name that contain the name of the application package and the name of the activity the UI belongs to. All UI elements are accessible by traversing hierarchy at the root node: ["activity_name"]["root"].

The second subset we selected is Play Store Metadata, which contains data about applications in the dataset including its category, play store rating, number of ratings, and the number of downloads stored in CSV file.

After carefully going through the UI screenshots, JSON files, and CSV file, we decided to extract the following features:

- Application Category
- Number of Ratings
- Scrollable Horizontal/Vertical
- Clickable Items
- Visible/Invisible Items
- Visible to User
- Focused Items
- Not Pressed Items
- Application Star Rating (Target Feature)

The common feature between the two subset is application package name, which is activity_name in first subset and package_name in CSV file. The First three features were obtained from CSV and sorted out based on the application category. Extracting the reminders of features from JSON required additional work as we had to initially split the first part of activity_name which is a path extracted from JSON files. Rico's JSON files are heavily nested, for parsing each file, we incorporated JSONPath Online Evaluator to facilitate parsing process and then applied the path to each distinct feature variable in Python to store the number of such elements in each application. We went through this process because we had to have as a minimum one feature name in common between both subsets to eventually merge them. For those features with Boolean values, we created a function to loop in the list for that feature and to increment the counter of that variable if the value was equal to True. Ultimately, the function returns the final value when the execution of loop halts.

The feature extracted from each subset were stored into two Python dictionaries. The last step of working with Rico dataset, we merged both dictionaries by appending them, finally removed brackets, commas, and all extra annotations. and saved the ultimate file as CSV.

Our performance analysis consists of comparing the results of Precision, Recall, and Accuracy before and after cross validation among four ML algorithms; Logistic Regression, Decision Tree, Random Forest, K-nearest Neighbor which are some of the most popular ML algorithms. Accuracy is the fraction of predictions our model got right which is calculated by dividing the number of correct predictions by the total number of samples. Precision measures the model trustiness in classifying positive samples, Recall measures how many positive samples were correctly classified by the model.

TP = True Positive

TN = True Negative

FP = False Positive

FN = False Negative

1. Accuracy = (TP + TN) / (TP + TN + FP + FN)

2. $$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

3. $$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

4. Machine Learning Models Performance Analysis

4.1 Logistic Regression

Regression models are predictive analysis based on theory and assumptions, and benefit from human interference and subject knowledge for model design. In LR, we calculate the probability that the realization of the output variable belongs to the appropriate category [69]. LR can be called a Linear Regression model, However the LR uses a more complex cost function, this cost function can be defined as the Sigmoid function or also known as the Logistic function instead of a linear function.

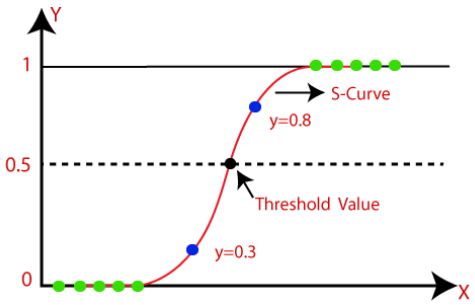

Figure 4-1 Logistic Regression graph

4. $f(x) = \frac{1}{1+e^{-(x)}}$

Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

4.2 Decision Tree

Decision trees are built via an algorithmic approach that classifies ways to split a data set based on various conditions. It is one of the most extensively used methods for Supervised Learning. Decision Trees are a non-parametric Supervised Learning method used for classification and regression tasks. Information gain is used to decide which feature to split on at each step when building the tree, from the highest information gain we can select the attribute.

[70][71]. ID3 is a Supervised Learning algorithm for Decision Trees. It is explicitly taught from a series of training examples from several classes. It predicts the class of an item based on the theory that it formulates. ID3 classifies features that distinguish one class of examples from others. ID3 requires that all features be known in advance and that each feature is well. Given a two group classification problem with each object a n-dimensional vector A= (a1,a2,a3…an), to measure the uncertainty of the two classification outcomes, group x_1 or x_2, the entropy function, can be defined as [72]:

$$5.\quad H(a)=\sum A[-P(x1|A)\log 2P(x1|A) - P(x2|A)\log 2P(x2|A)]$$

4.3 Random Forest

Random Forest (RF) classifier is an ensemble classifier that produces multiple decision trees, using a randomly selected subset of training samples and variables. It is very

popular due to the accuracy of its classification. The trees are created by drawing a subset of training samples through replacement (a bagging approach). This means that the same sample can be selected several times, while others may not be selected at all [73].

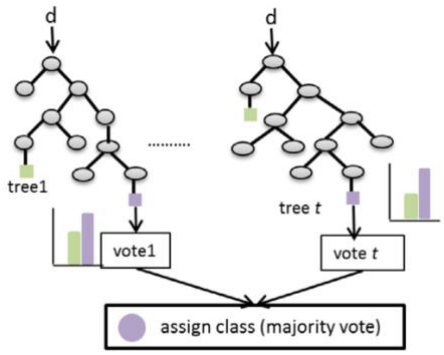

Figure 4-2 Random Forest

4.4 K-Nearest Neighbor

k-nearest neighbor algorithm is used to estimate and impute missing data. k-nearest neighbor method can predict both qualitative attributes (the most frequent value among k-nearest neighbors) and quantitative attributes (the mean among k-nearest neighbors). The drawback of this method is that, whenever the k-nearest neighbor looks for the most similar instances, the algorithm searches through the complete data set [71].

Chapter 5

Experimental Results

In this section, the results obtained from the ML models are analyzed and discussed. This experiment involved training the models on the training set. The same set used for 10-fold cross validation and testing them on the held out 20% unseen Rico data. To measure each model's performance, Accuracy, Recall and Precision were generated.

5.1 Model Comparison and Evaluation

5.1.1 Logistic Regression

As you can see in figure 5.1, Accuracy improved after cross validation as it provides a way to improve the estimated performance of a ML model. Initially our model was overfitted because of the number of features involved, accuracy in from overfitted model decreased after cross validation. Hence, we had to perform feature selection (Forward selection) to exclude unnecessary and redundant features, reducing the amount of data needed for learning, improving algorithms' predictive accuracy, and increasing the constructed models' unambiguousness [74]. Forward selection is an iterative method in which we start with having no feature in the model. For each round, we add the feature which best enhances our model till an addition of a new variable does not improve the performance of the model anymore.

There are no significant changes in Accuracy and Recall values after cross validation, as shown in the figure below, Precision increased once we cross validated the set.

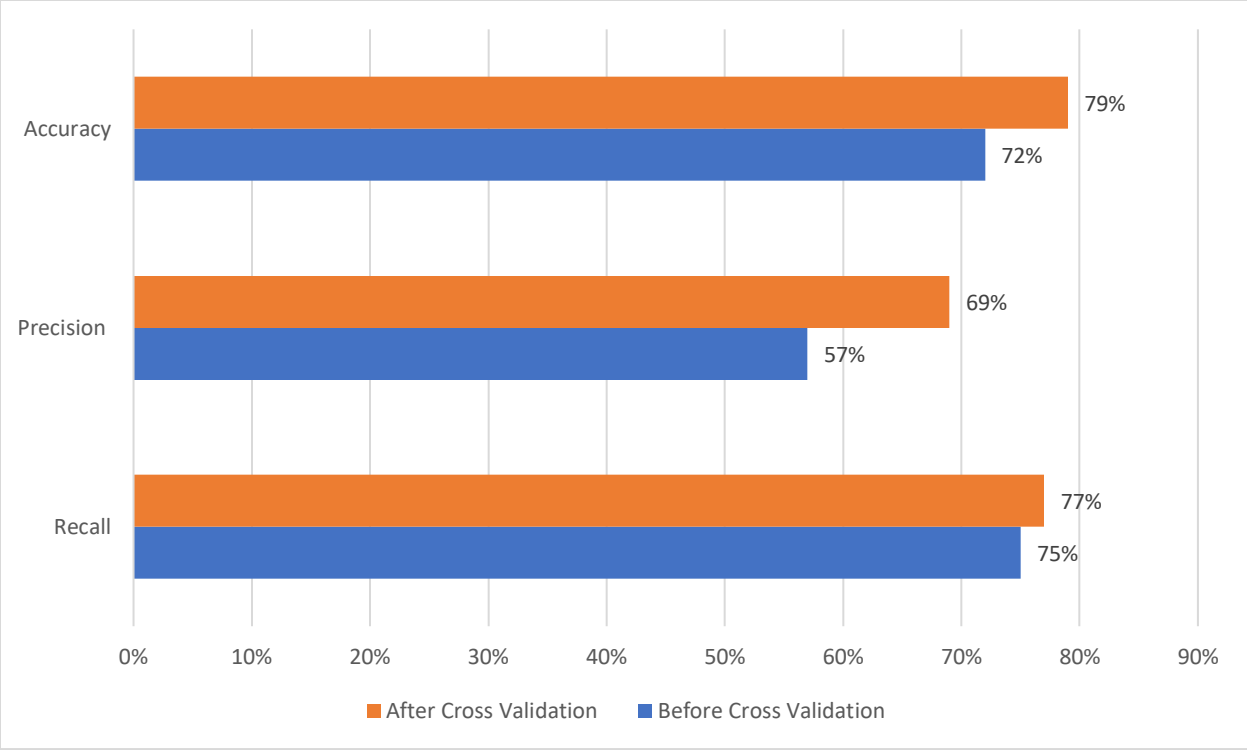

Figure 5-1 Comparison of Accuracy, Precision and Recall in Logistic Regression

Cross validation almost always led to lower estimated errors as it uses some data that are different from the set. The Logistic Regression model performs better, though we may still want to use a Decision Tree for its interpretability. Logistic regression for classification works best when the classes are well separated in the feature space.

We calculated feature rankings for our model in the figure below, as we expected application category plays a significant role in our model. Feature selection is an important step in model tuning as it reduces dimensionality in a dataset which improves the speed and performance of a model. The negative numbers mean that our model is not getting good use of this feature.

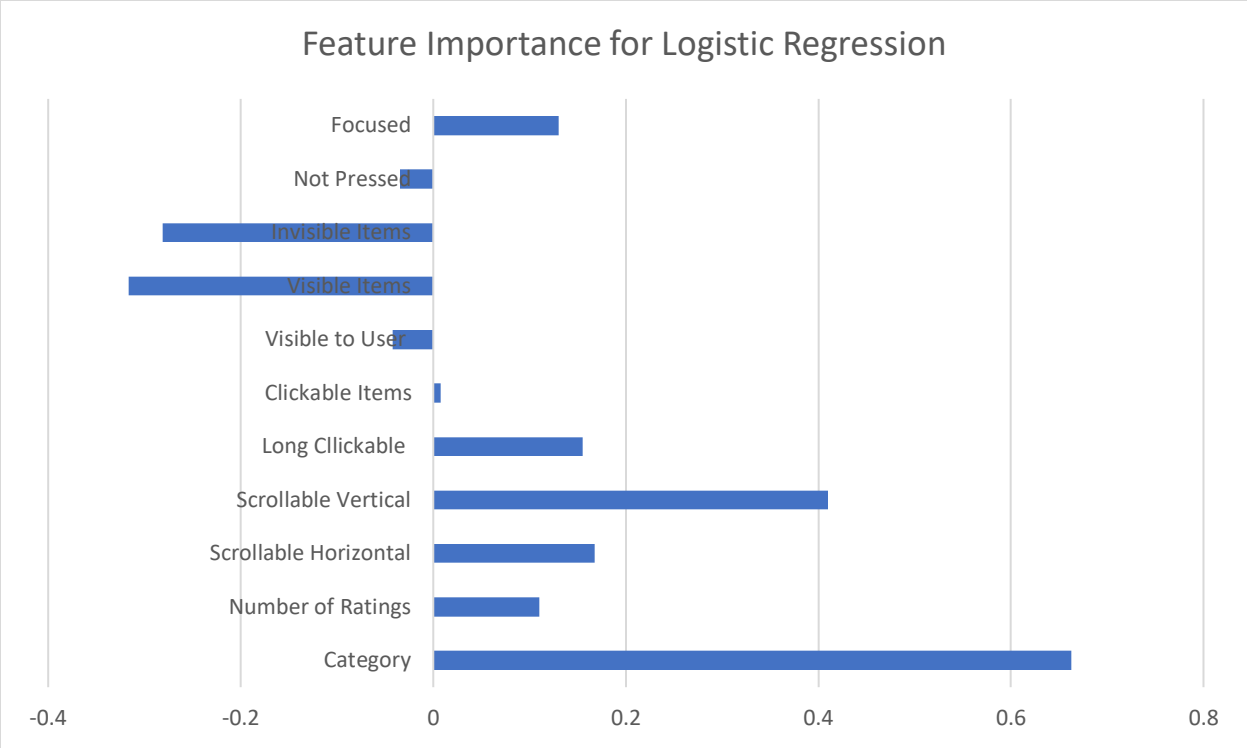

Figure 5-2 Feature importance chart for Logistic Regression

5.1.2 Decision Tree

You can see that the accuracy and precision of the Logistic Regression model is higher, and the recalls of the two models are about the same. Decision Tree algorithm is the Gini Impurity. However, entropy is implemented, and we can select which one to use when creating the DecisionTreeClassifier object. Decision Trees are very susceptible to random characteristics in the training dataset. We say that Decision Trees have high variance since if you randomly change the training dataset, you may end up with a very different looking tree. To reduce the variance, we can use bagging technique [75]. Accuracy and Precision decreased after cross validation as you may see in the figure below. Recall almost remained the same.

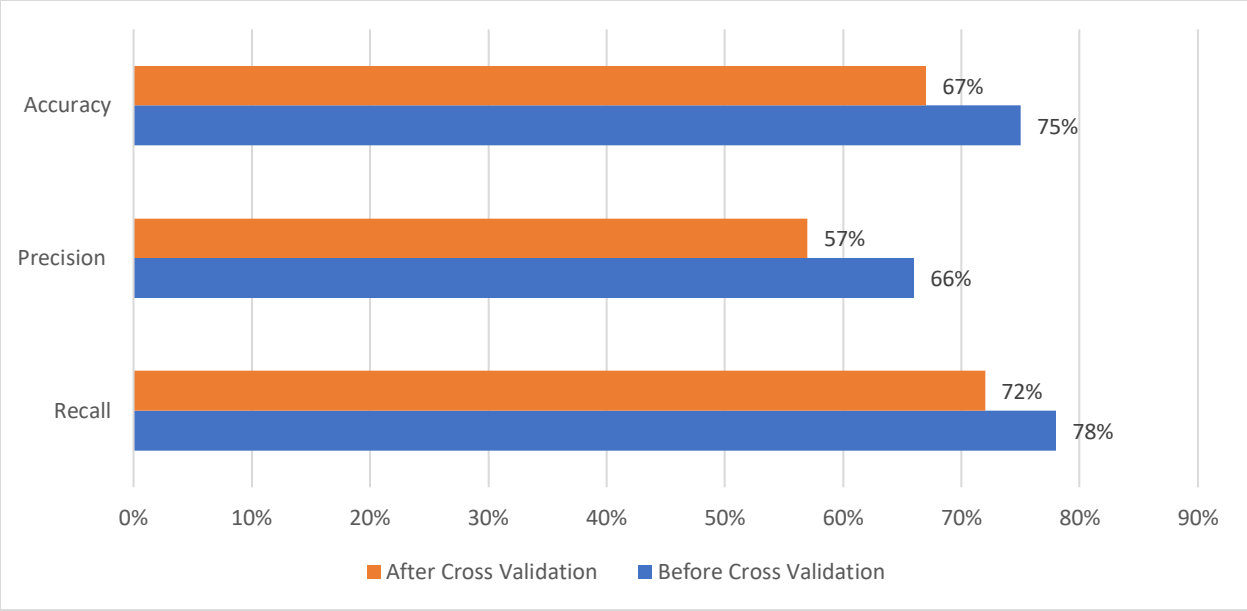

Figure 5-3 Comparison of Accuracy, Precision and in Decision Tree

Decision trees tends to overfit in comparison to other algorithms, which provide lower accuracy. they are greedy and deterministic they don't normally provide the best result. That is why random forest and gradient boosting appeared and they are extremely good. They substitute this drawback of decision trees.

In our model, visible items, focused, long clickable, and category were of highest importance features respectively as you may see in the following figure. Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The higher the values the more important the feature [78].

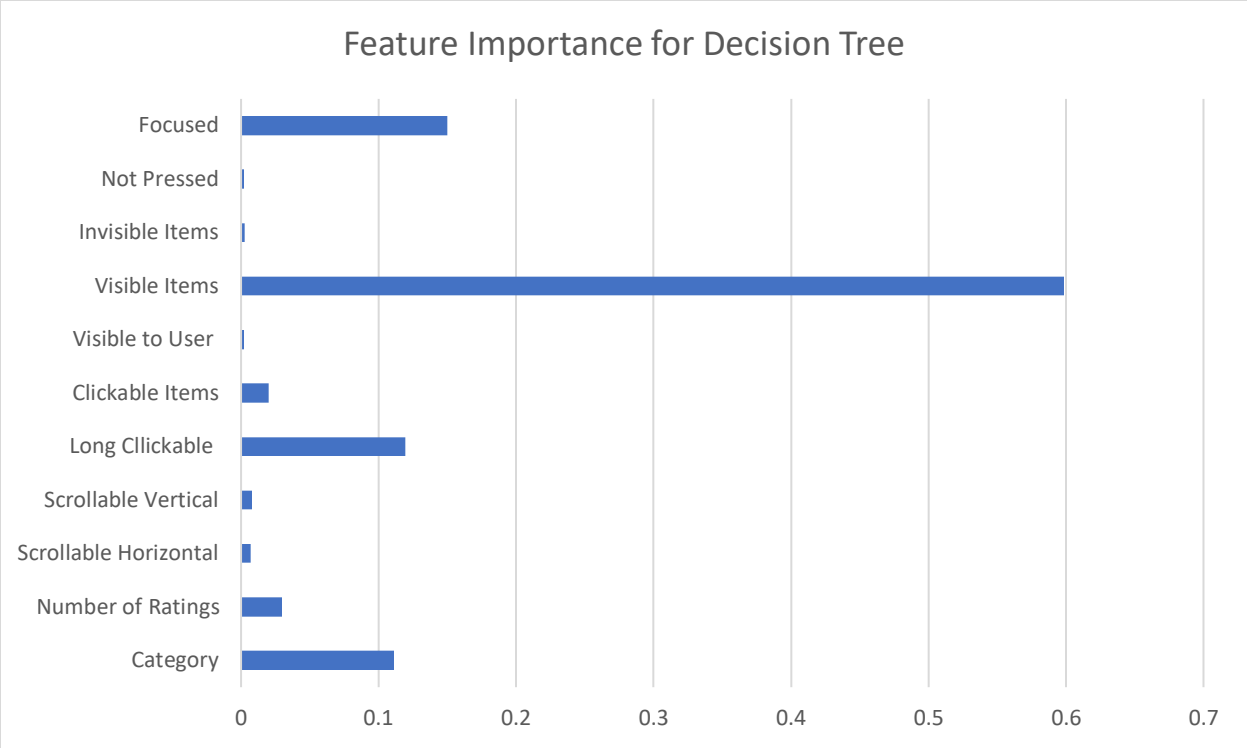

Figure 5-4 Comparison of Accuracy, Precision and Recall in Decision Tree

5.1.2 Random Forest

For our Random Forest model, we have added the random state parameter here so that it will do the same split every time we run the code. Without the random state, we'd expect different datapoints in the training and testing sets each time we run the code which can make it harder to test the code. the accuracy for the decision tree model is 67%, much worse than that for the random forest. With Random Forest, overfitting is generally not an issue. We used two tuning parameters n_estimators (the number of trees) and max_features (the number of features to consider at each split which is the square root of number of features (p)). Random Forest also does better than Logistic Regression [76].

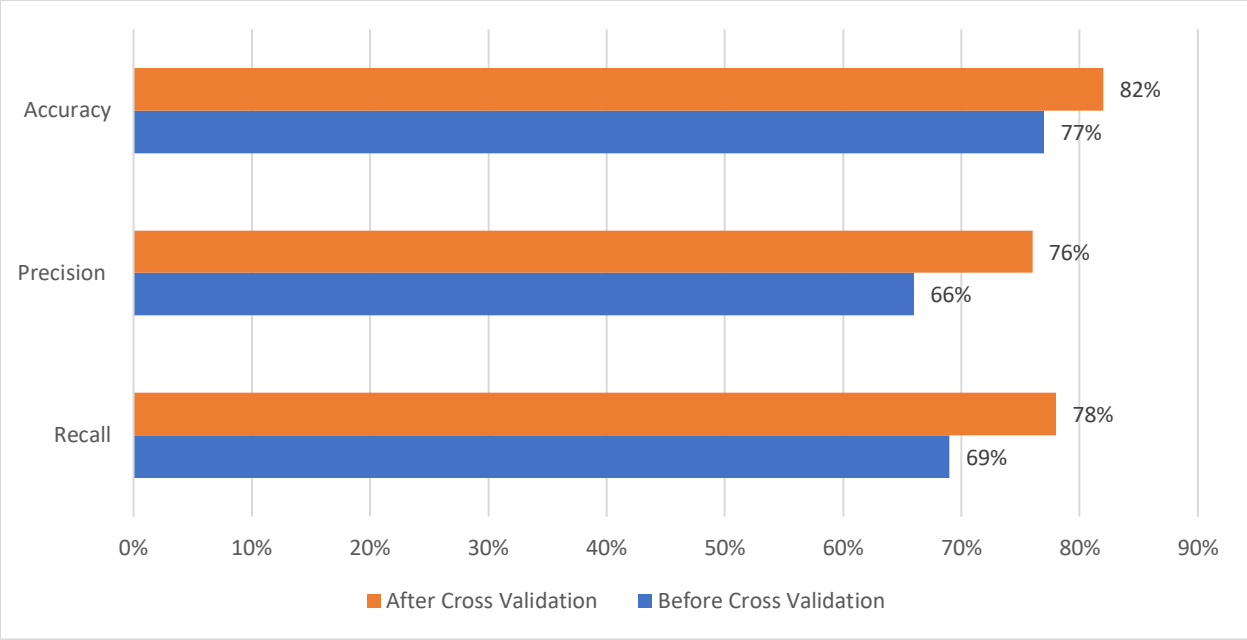

Figure 5-5 Comparison of Accuracy, Precision and Recall in Random Forest

One of the reasons that Random Forest gives great results is because it performs implicit feature selection because it splits nodes on the most important variables, but other ML models do not [77]. A classification algorithm consisting of many decision trees combined to get a more accurate result as compared to a single tree.

The result of feature rankings for our model is shown in the figure below. Scrollable items (Horizontally and vertically) are the features with highest rank. Clickable items on the other hand, has the least importance among other features.

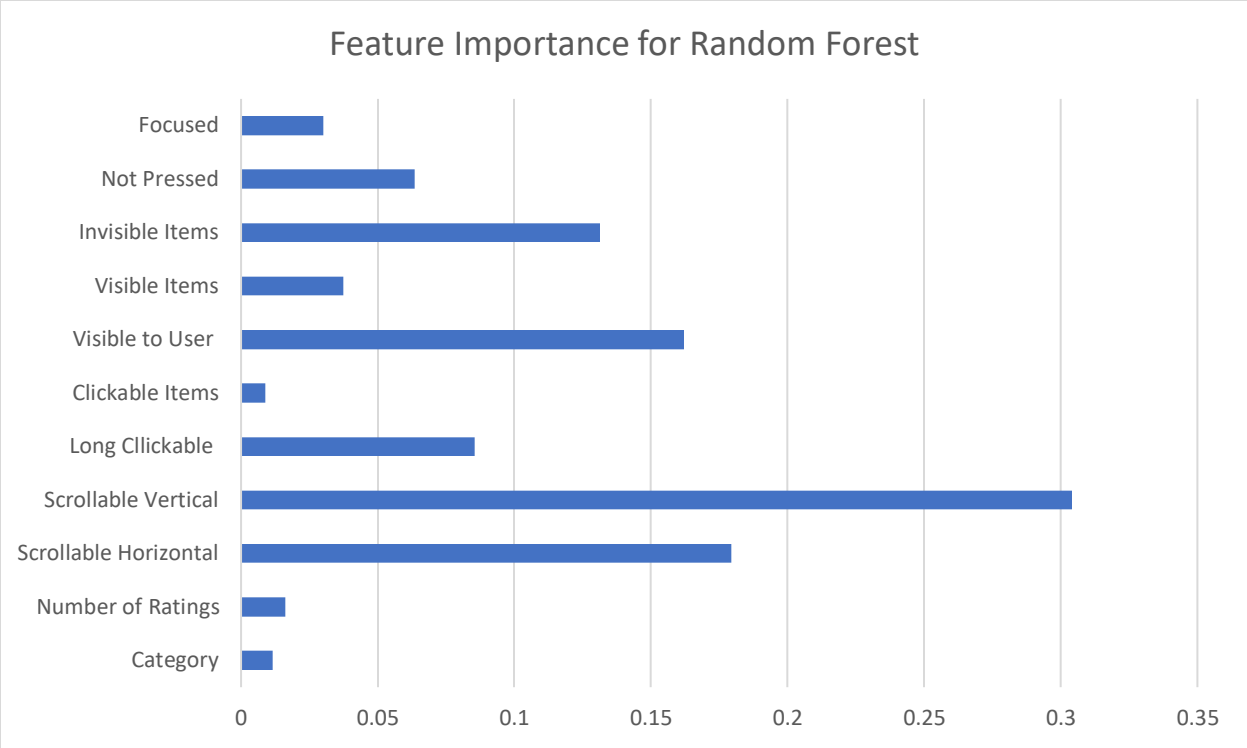

Figure 5-6 Feature importance for Random Forest

5.1.4 K-Nearest Neighbor

K-Nearest Neighbor (KNN) algorithm is one of the ML methods which has solid consistency, finding cases by calculating the proximity between new cases and old cases based on weights adjustment. KNN produced the best accuracy results among all the other algorithms we used previously. As it's shown in the figure below, Recall significantly improved after cross validation, but we did not experience a significant change in Precision value.

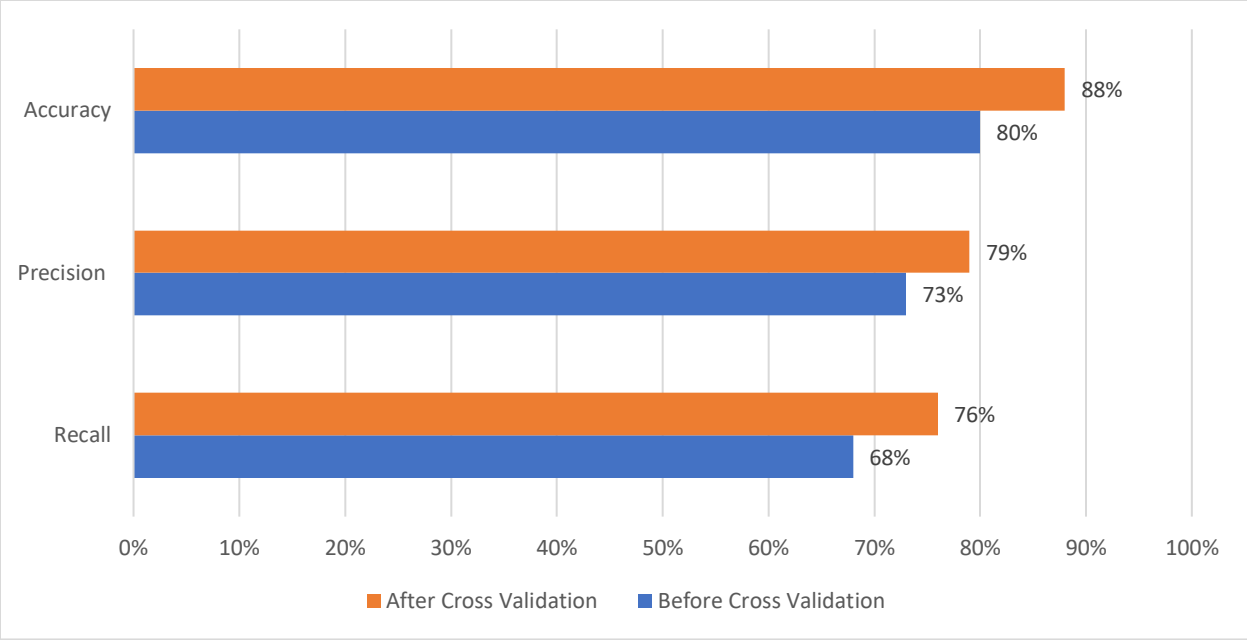

Figure 5-7 Comparison of Accuracy, Precision, Recall in KNN

Our feature importance ranking for the KNN model shows great results as you can see in the following figure. Almost all features have very close importance value with number of ratings having the utmost impact and category the least impact.

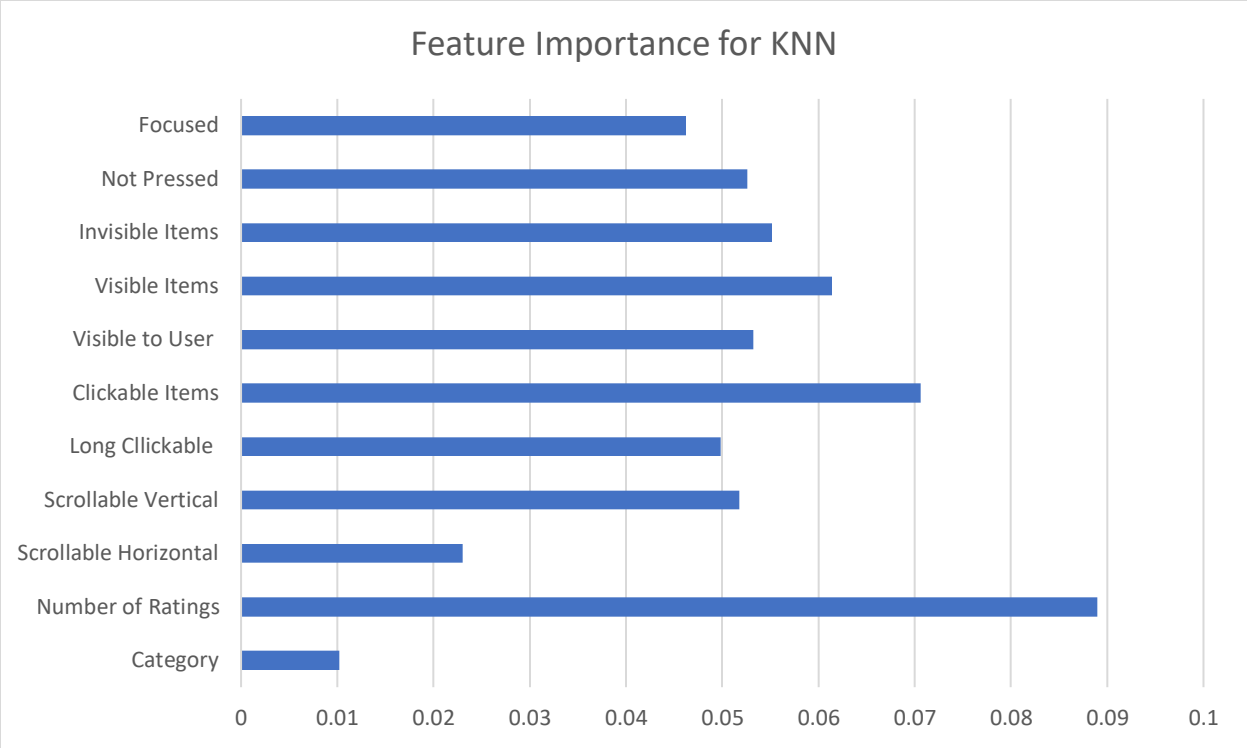

Figure 5-8 Feature importance for KNN

We summed up the computed accuracy for each model before and after cross validation in the figure below.

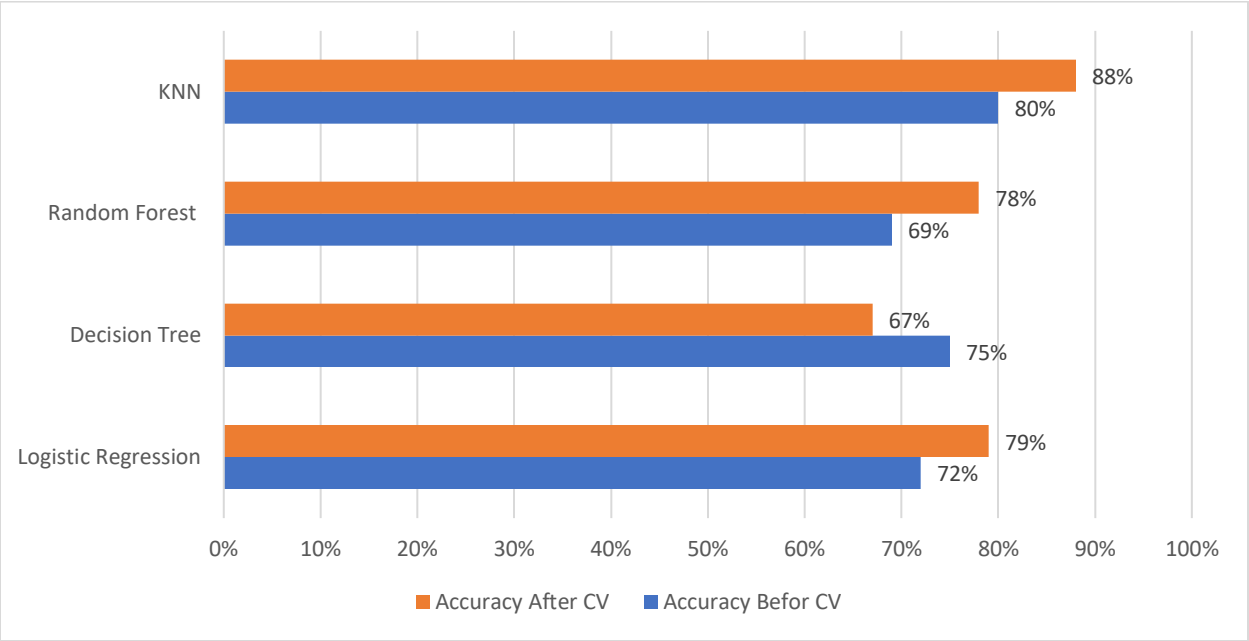

Figure 5-9 Comparison of Accuracy before and after cross validation

Chapter 6

Conclusion and Future Work

Due to the continuous increase in mobile application, customer review analysis and star rating prediction are attracting increasingly more attention. Most modern mobile applications are GUI dependent and focus on attractive user interfaces and spontaneous user experiences to attract client. A successful application provides an amazing UI and UX. Therefor UI features have direct impacts on the user's satisfaction of an application. However, there are very few works that have reviewed the current state of the user interface features analysis area of research. As it is such, we systematically defined important UI features, towards information extraction and analysis. We first identified popular ML models that are employed on client's reviews. we discussed the many application areas that are incorporating customer review analysis.

In terms of future research directions, we are going to research feature selections based on different application's categories. For instance, the most important UI features for a map application will be different than a game application. Ultimately, we will increase or decrease the number of features based on application's categories to compare our results. We also plan to increase the number of models we used and incorporate confusion matrices to analyze results from the cross-validation stage to improve the predictive power of our models.

Bibliography

[1] Mohamed Alloghani, Dhiya Al-Jumeily, Thar Baker, Abir Hussain, Jamila Mustafina, Ahmed J. Aljaaf (2020) - An Intelligent Journey to Machine Learning Applications in Component-Based Software Engineering - Springer.

[2] M. Kim, T. Zimmermann, R. DeLine and A. Begel (2018) - Data scientists in software teams: State of the art and challenges – IEEE.

[3] Foutse Khomh, Bram Adams, Jinghui Cheng, Marios Fokaefs, Giuliano Antoniol (2018) - Software Engineering for Machine-Learning Applications: The Road Ahead – IEEE.

[4] Murat Cetiner, Ozgur Koray Sahingoz (2020) - A Comparative Analysis for Machine Learning based Software Defect Prediction Systems – IEEE.

[5] Karl Meinke, Amel Bennaceur (2018) - Machine Learning for Software Engineering: Models, Methods, and Applications – IEEE.

[6] Maria DE-Arteaga, William Herlands and Daniel B. Neill, Artur Dubrawski (2018) - Machine Learning for the Developing World – ACM.

[7] Vehbi Yurdakurbann, Nadia Erdogan (2018) - Comparison of machine learning methods for software project effort estimation – IEEE.

[8] Geanderson Esteves, Eduardo Figueiredo, Adriano Veloso, Markos Viggiato, Nivio (2020) - Understanding Machine Learning Software Defect Predictions – Springer.

[9] Tahira Iqbal, Parisa Elahidoost, Levi Lúcio (2018) - A Bird's Eye View on Requirements Engineering and Machine Learning – IEEE.

[10] Raul Navarro-Almanza, Reyes Juarez-Ramirez, Guillermo Licea (2017) - Towards Supporting Software Engineering Using Deep Learning: A Case of Software Requirements Classification – IEEE.

[11] M. Gramajo, L. Ballejos, M. Ale (2020) - Seizing Requirements Engineering Issues through Supervised Learning Techniques – IEEE.

[12] Jennifer Horkoff (2019) - Non-Functional Requirements for Machine Learning: Challenges and New Directions – IEEE.

[13] Amandeep Kaur, Sushma Jain, Shivani Goel (2017) - A Support Vector Machine Based Approach for Code Smell Detection – IEEE.

[14] Ashish Kumar Dwivedi, Anand Tirkey, Santanu Kumar Rath (2016) - Applying software metrics for the mining of design pattern – IEEE.

[15] Andreas Holzinger, Peter Kieseberg, Edgar Weippl, A Min Tjoa (2018) - Current Advances, Trends and Challenges of Machine Learning and Knowledge Extraction: From Machine Learning to Explainable AI – Springer.

[16] Ashish Kumar Dwivedi, Anand Tirkey, Ransingh Biswajit Ray, Santanu Kumar Rath (2016) - Software design pattern recognition using machine learning techniques – IEEE.

[17] Ben Hutchinson, Andrew Smart, Alex Hanna, Emily Denton (2021) - Towards Accountability for Machine Learning Datasets: Practices from Software Engineering and Infrastructure – ACM.

[18] Gorkem Giray (2021) - A software engineering perspective on engineering machine learning systems: State of the art and challenges - Science Direct.

[19] Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, Thomas Zimmermann (2017) - Software Engineering for Machine-Learning: A Case Study – IEEE.

[20] Himanshu Gupta; Lov Kumar; Lalita Bhanu Murthy Neti (2019) - An Empirical Framework for Code Smell Prediction using Extreme Learning Machine – IEEE.

[21] Mauricio Aniche, Erick Maziero, Rafael Durelli, Vinicius H. S. Durelli (2020) - The Effectiveness of Supervised Machine Learning Algorithms in Predicting Software Refactoring – IEEE.

[22] Meiliana; Syaeful Karim; Harco Leslie Hendric Spits Warnars; Ford Lumban Gaol; Edi Abdurachman; Benfano Soewito (2017) - Software metrics for fault prediction

using machine learning approaches: A literature review with PROMISE repository dataset – IEEE.

[23] Rui Lima, António Miguel Rosado da Cruz, Jorge Ribeiro (2020) - Artificial Intelligence Applied to Software Testing: A Literature Review – IEEE.

[24] J. Jenny Li, Andreas Ulrich, Xiaoying Bai, Antonia Bertolino (2020) - Advances in test automation for software with special focus on artificial intelligence and machine learning – Springer.

[25] Krunal Bhavsar, Samir Gopalan, Vrutik Shah (2019) - Machine Learning: A Software Process Reengineering in Software Development Organization – Research Gate.

[26] Hong Zhu (2018) - Software Testing as a Problem of Machine Learning: Towards a Foundation on Computational Learning Theory – IEEE.

[27] Sebastiano Panichella, Marcela Ruiz (2020) - Requirements-Collector: Automating Requirements Specification from Elicitation Sessions and User Feedback – IEEE.

[28] Zhiyuan Wan, Xin Xia, David Lo, Gail C. Murphy (2019) - How does Machine Learning Change Software Development Practices? – IEEE.

[29] Abdessalam Elhabbash, Maria Salama, Rami Bahsoon, Peter Tino (2019) - Self-awareness in Software Engineering: A Systematic Literature Review

– ACM.

[30] Saad Shafiq, Atif Mashkoor, Christoph Mayr-Dorn, Alexander Egyed (2021) - A Literature Review of Using Machine Learning in Software Development Life Cycle Stages – IEEE.

[31] Matthew England (2018) - Machine Learning for Mathematical Software – Springer.

[32] Hamza Abubakar, M. S. Obaidat, Aaryan Gupta, Pronaya Bhattacharya, Sundeep Tanwar (2020) - Interplay of Machine Learning and Software Engineering for Quality Estimations – IEEE.

[33] Gaith Quba, Hadeel Al Qaisi, Ahmad Althunibat, Shadi AlZu'bi (2021) - Software Requirements Classification using Machine Learning algorithm's – IEEE.

[34] Pratvina Talele, Rashmi Phalnikar (2021) - Classification and Prioritisation of Software Requirements using Machine Learning – A Systematic Review – IEEE.

[35] Dalia Sobhy, Rami Bahsoon, Leandro Minku, Rick Kazman (2021) - Evaluation of Software Architectures under Uncertainty: A Systematic Literature Review – ACM.

[36] Mohammad Y. Mhawish, Manjari Gupta (2019) - Software Metrics and tree-based machine learning algorithms for distinguishing and detecting similar structure design patterns – Springer.

[37] Sara Elmidaoui, Laila Cheikhi, Ali Idri, Alain Abran (2020) - Machine Learning Techniques for Software Maintainability Prediction: Accuracy Analysis – Springer.

[38] Xiaofang Zhang, Tida Zhou, Can Zhu (2017) - An Empirical Study of the Impact of Bad Designs on Defect Proneness – IEEE.

[39] S. McIntosh, Y. Kamei (2017) - Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction – IEEE.

[40] Siddharth Bhatore, Y. Raghu Reddy, Lalit Mohan Sanagavarapu, Svl Sarat Chandra (2021) - Software Patterns to Identify Credit Risk Patterns – IEEE.

[41] L Zhang, J Tan, D Han et al. (2017) - From machine learning to deep learning: progress in machine intelligence for rational drug discovery – IEEE.

[42] Yafei Zhu, Lingyun Chen, Hong Zhou, Wanli Feng, Quanyin Zhu (2018) - Design and Implementation of WeChat Robot Based on Machine Learning – IEEE.

[43] Shin Nakajima (2018) - Quality Assurance of Machine Learning Software – IEEE.

[44] Zhiqiang Gong, Ping Zhong, Weidong Hu (2019) - Diversity in Machine Learning – IEEE.

[45] C. Chen, A. Seff, A. Kornhauser and J. Xiao (2015) - DeepDriving: Learning affordance for direct perception in autonomous driving - IEEE.

[46] H. Peng, B. Li, H. Ling, W. Hu, W. Xiong and S. J. Maybank (2017) - Salient object detection via structured matrix decomposition - IEEE.

[47] Morteza Zakeri Nasrabadi, Saeed Parsa (2021) – Learning to Predict Software Testability - IEEE.

[48] Marc Roper (2019) - Using Machine Learning to Classify Test Outcomes – IEEE.

[49] Manu Banga, Abhay Bansal, Archana Singh (2019) - Implementation of Machine Learning Techniques in Software Reliability: A framework – IEEE.

[50] Kim Herzig and Nachiappan Nagappan (2015) - empirically detecting false test alarms using association rules – IEEE.

[51] Shikha Gupta, Anuradha Chug (2021) - An Optimized Extreme Learning Machine Algorithm for Improving Software Maintainability Prediction – IEEE.

[52] H. Alsolai (2018) - Predicting Software Maintainability in Object-Oriented Systems Using Ensemble Techniques – IEEE.

[53] N. Baskar and C. Chandrasekar (2018) - An Evolving Neuro-PSO-based Software Maintainability Prediction – IEEE.

[54] Sudan Jha, Raghvendra Kumar , Le Hoang son, Mohamed Abdel-Basset , Ishaani Priyadarshini, Rohit Sharma, Hoang Viet Long (2019) - Deep Learning Approach for Software Maintainability Metrics Prediction – IEEE.

[55] Mohd. Haleem, Md. Faizan Farooqui, Md. Faisal (2021) - Cognitive impact validation of requirement uncertainty in software project development – Science Direct.

[56] Thanis Paiva, Amanda Damasceno, Eduardo Figueiredo, Claudio Sant'Anna (2017) - On the evaluation of code smells and detection tools – Springer.

[57] Shahida Sulaiman (2005) - Viewing Software Artifacts for Different Software Maintenance Categories Using Graph Representations – Research Gate.

[58] Priyadarshni Suresh Sagar, Eman Abdulah AlOmar, Mohamed Wiem Mkaouer, Ali Ouni, Christian Newman (2021) - Comparing Commit Messages and Source Code Metrics for the Prediction Refactoring Activities – Research Gate.

[59] Yang Xin, Lingshyang Kong, Zhi Liu, Yuling Chen, Yanmiao Li, Hongliang Zhu, Mingcheng Gao, HAaixia Hou, Chunhua Wang (2018) - Machine Learning and Deep Learning Methods for Cybersecurity – IEEE.

[60] Xueqin Zhang, Chunhua Gu, Jiajun Lin (2006) - Support Vector Machines for Anomaly Detection – Research Gate.

[61] Miltiadis Allamanis (2018) - The adverse effects of code duplication in machine learning models of code – Research Gate.

[62] Olimar Teixeira Borges, Julia Colleoni Couto, Duncan Ruiz, Rafael Prikllad nicki (2021) – Challeneges in using Machine Learning to Support Software Engineering – Research Gate.

[63] Batta Mahesh (2018) - Machine Learning Algorithms - A Review – Research Gate

[64] Iqbal H. Sarker, Mohammed Moshiul Hoque, Md. Kafil Uddin, Tawfeeq Alsansoory (2021) - Mobile Data Science and Intelligent Apps: Concepts, AI-Based Modeling and Research Directions – Springer

[65] Lina Zhou, Shimei Pan, Jianwu Wang, Athanasios V. Vasilakos (2017) - Machine learning on big data: Opportunities and challenges – Science Direct

[66] Zhiyuan Wan, Xin Xia, David Lo, Gail C. Murphy (2021) – How does Machine Learning change Software Development Practices? – IEEE

[67] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Danniel Afergan, Yang Li, Jeffret Nichols, Ranjitha Kumar (2017) – Rico: A Mobile App Dataset for Building Data-Driven Design Applications - ACM

[68] Thomas F. Liu, Mark Craft, Jason Situ, Ersin Yumer, Radomir Mech, Ranjitha Kumar (2018) – Learning Design Semantics for Mobile Apps – ACM

[69] Evangelia Christodoulou, Jie Ma, Gary S. Collins, Ewout W. Steyerberg, Jan Y. Verbakel, Ben Van Calster (2019) - A systematic review shows no performance benefit

of machine learning over logistic regression for clinical prediction models – Science Direct

[70] Madan Somvanshi, Pranjali Chavan, Shital Tambade, S. V. Shinde (2016) - A review of machine learning techniques using decision tree and support vector machine – IEEE

[71] Sachin Chavankar, Sudhirkumar Sawarkar (2015) - Decision Tree: Review of Techniques for Missing Values at Training, Testing and Compatibility – IEEE

[72] Arundhati Navada, Aamir Nizam Ansari, Siddharth Patil, Balwant A. Sonkamble (2011) - Overview of use of decision tree algorithms in machine learning – IEEE

[73] Mariana Belgiu, Lucian Dragut (2016) – Random Forest in Remote Sensing: A Review of Applications and Future Directions – Elsevier

[74] H. Liu, E.R. Dougherty, J.G Dy, K. Torkkola, E. Peng, C.Ding, F.Long, M.Berens, L.Parsons, L. yu, Z. Zhao, G. Forman (2005) – Evolving Feature Selection – IEEE.

[75] https://towardsdatascience.com/decision-tree-in-machine-learning-e380942a4c96

[76] https://towardsdatascience.com/financial-data-analysis-51e7275d0ae

[77] https://towardsdatascience.com/improving-random-forest-in-python-part-1-893916666cd

[78] https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3