EFFORT ESTIMATION IN AGILE SOFTWARE DEVELOPMENT USING ENSEMBLE LEARNING MODEL

By

HABIBA SATTAR

NATIONAL UNIVERSITY OF MODERN LANGUAGES

ISLAMABAD

May, 2025

Effort estimation in agile software development using ensemble learning model

By

HABIBA SATTAR

MSSE, National University of Modern Languages, Islamabad, 2025

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

In **Software Engineering**

To

FACULTY OF ENGINEERING & COMPUTING

NATIONAL UNIVERSITY OF MODERN LANGUAGES ISLAMABAD

THESIS AND DEFENSE APPROVAL FORM

The undersigned certify that they have read the following thesis, examined the defense, are satisfied with overall exam performance, and recommend the thesis to the Faculty of Engineering and Computing for acceptance.

Thesis Title: EFFORT ESTIMATION FOR AGILE SOFTWARE DEVELOPMENT USING ENSEMBLE LEARNING MODEL

Submitted by: Habiba Sattar **Registration #:** 53-MS/SE-F21

Master of Science in Software Engineering
Title of the Degree

Software Engineering
Name of Discipline

Dr. Jaweria Kanwal _____
Name of Research Supervisor Signature of Research Supervisor

Dr. Sumaira Nazir _____
Name of Head of Department SE Signature of Head of Department SE

Dr. Noman Malik _____
Name of Dean (FEC) Signature of Dean (FEC)

May, 2025
Date

AUTHOR'S DECLARATION

I Habiba Sattar

Daughter of Sattar Muhammad

Registration # 53 MS/SE/F21

Discipline Software Engineering

Candidate of **Master of Science in Software Engineering (MSSE)** at the National University of Modern Languages do hereby declare that the thesis **Effort Estimation in Agile Software Development Using Ensemble Learning Model** submitted by me in partial fulfillment of MSSE degree, is my original work, and has not been submitted or published earlier. I also solemnly declare that it shall not, in the future, be submitted by me for obtaining any other degree from this or any other university or institution. I also understand that if evidence of plagiarism is found in my thesis/dissertation at any stage, even after the award of a degree, the work may be canceled and the degree revoked.

Signature of Candidate

Habiba Sattar

Name of Candidate

May, 2025

Date

ABSTRACT

Effort Estimation in Agile Software development Using Ensemble Learning Model

In the domain of software development, effort estimating is an essential procedure that entails projecting the size and schedule of a particular project. It becomes necessary to create an estimate before beginning any software project. Obtaining the required approvals and evaluating the project depends on this preliminary assessment. The importance of this procedure cannot be emphasized since a project's success or failure is solely dependent on how precisely and accurately effort is estimated. There are various cost and effort methods and techniques. These techniques have been utilized to construct several effort estimation models that are used in the software development process in the traditional model. This research explores the application of ensemble learning techniques, specifically stacking, to enhance the accuracy of effort estimation in Agile environments. Stacking involves combining multiple diverse base estimators to create a meta-estimator that outperforms individual models. This study includes a crucial step for gathering datasets because old dataset size is small and old. The proposed approach is assessed using real-world Agile project datasets, proving the advantages of the stacking model over agile software development estimation techniques.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

ACKNOWLEDGEMENTS

First of all, I wish to express my gratitude and deep appreciation to Almighty Allah, who made this study possible and successful. This study would not be accomplished unless the honest espousal that was extended from several sources for which I would like to express my sincere thankfulness and gratitude. Yet, there were significant contributors for my attained success and I cannot forget their input, especially my research supervisors, Dr. Jaweria Kanwal, who did not leave any stone unturned to guide me during my research journey.

I shall also acknowledge the extended assistance from the administrations of Department of Software Engineering who supported me all through my research experience and simplified the challenges I faced. For all whom I did not mention but I shall not neglect their significant contribution, thanks for everything.

DEDICATION

This thesis is dedicated to my Teacher and Parents, whose boundless love, encouragement, and sacrifices have been the cornerstone of my academic journey. Their unwavering support and belief in me have provided the foundation for all my accomplishments.

CHAPTER 1

INTRODUCTION

1.1 Overview

The Software plays an essential role in our day-to-day life. It enhances our experience in various fields, such as healthcare, education, fitness and security. For instance, fitness apps play a key role in our well-being by motivating us to stay active and healthy, tracking our workouts and health data.

The software development life cycle contains several stages, such as requirement gathering, design, development, testing, and deployment. Every project needs to be followed carefully through each of these stages in order to succeed. In software development, accurate estimation is part of successful project management. Effort, cost, and time estimation are critical for managing software development projects. Software estimation involves predicting the size of the software product, how much work it will take to develop, setting up a project schedule, and estimating the overall cost of the project. It is widely regarded as the most critical and challenging task in project management, crucial for accurately forecasting expenses.

Accurately estimating necessary resources and schedules is essential for achieving successful software development. [2][3]. It is widely acknowledged that almost 75% of overview of projects exceed their budget, time frame, or both, as indicated by the CHAOS immediate reports that repeatedly illustrate a decline in project success rates [85]. The most vital and intricate challenge in software development is to precisely anticipate the development cost, time, and efforts, enabling project managers, system analysts, and developers to make informed management decisions. Failure to do so can result in complete disaster. It is believed that substantial budget overruns occur solely due to inaccurate estimation

Software Development Effort Estimation (SDEE) is a crucial component of software management, which measures the effort to develop software. It entails calculating the quantity of work requires to complete a project as a whole or a specific task within it. Generating an incredibly accurate estimation is useful for developers when they are working on project. A project's success can be at risk if the estimates are inaccurate or unrealistic [1].

Effort estimation predicts the amount of work required, measured in person-hours or person- days. Project managers can better plan workloads and prevent overworked teams by using this estimation by precisely calculating the amount of work needed for each activity through the use of effort estimate. Team members can allocate duties more fairly and prevent overloading by knowing the estimated person-hours or person-days required [86]. As a result, it is possible to have a balanced workload, the likelihood of burnout is decreased, and team morale and productivity are raised.

Cost estimation calculates the total financial expenditure needed, including both direct (like labor and materials) and indirect costs (like overhead). This is essential for maintaining the project's financial viability and for developing precise budgets [87]. Time estimation predicts the duration needed to complete tasks or the entire project. Precise time estimation is essential as it has an immediate effect on work planning and sequencing, resource distribution, and project effectiveness in general. Time estimates that are unrealistic run the risk of missing deadlines, which can cause delays, higher expenses, and even project failure. Time estimation is therefore crucial for meeting stakeholder expectations and sustaining project momentum. Although every type of estimation focuses on different aspects of project management, they are all necessary for the project's success. In this research we calculate effort in terms of cost and time [87].

In the traditional waterfall approach, estimation is done at the beginning of the project. Teams make detailed plans and predictions about how long the project will take and how much it will cost, based on all the requirements known at that time. This method tries to predict everything upfront and is less flexible to changes once the project is underway. If something changes, it can disrupt the schedule and require re-estimating and adjusting the plans. On the other hand, agile methods estimate effort in a more flexible and ongoing way. Instead of planning everything at once, agile teams estimate the effort for each part of the project (called sprints or

iterations) as they go along. They use user stories, which describe what needs to be done from the user's perspective.

Agile allows for continuous feedback and adjustments, making it easier to handle changes and adapt to new information throughout the project. This approach is more flexible and responsive compared to the rigid planning of traditional methods.

Machine learning, a subfield of artificial intelligence, employs algorithms to examine data and provide forecasts by identifying patterns and correlations within it. Machine learning algorithms are trained on software project data in the context of effort estimation to gain insight from past trends and enhance estimation precision. In the following section we will discuss detail of agile software development.

1.1 Agile Software Development

Agile is latest software development process. The terminology "agile development" refers to a variety of incremental and iterative software development technique. Agile is flexible and lightweight approach to software development unlike traditional software development, agile software development is the plan-driven strategies and embraces change.

Agile emphasizes dividing the project into smaller, more manageable pieces called iterations or sprints. Agile's iterative appr enables ongoing feedback and prompt project modifications in response to emerging requirements or difficulties. Agile teams frequently collaborate closely, holding frequent meetings to discuss progress and resolve problems, such as daily stand-ups. Agile also prioritizes collaboration between the customer and the development team. Customers are involved throughout the process, to ensure that the final software fulfills their needs and expectations, as opposed to waiting till the conclusion of the project to deliver the final result.

The Agile Manifesto highlights the key differences between Agile and plan-driven methodologies, with Agile being more flexible and responsive to change, making it better suited for dynamic project environments. Agile is founded on 12 key principles and values individuals

and interactions, working software, customer collaboration, and adaptability over strict processes, comprehensive documentation, contract negotiation, and following a plan [10] The Agile Manifesto emphasizes flexibility and responsiveness, contrasting with the rigidity of plan-driven methodologies. Agile promotes frequent reviews, retrospectives, and team ownership of deliverables, all of which help to create a transparent and accountable culture.

Agile guarantees a quicker time to market and an earlier realization of commercial value by regularly releasing functional software. Agile values direct communication and collaboration over strict processes and tools, prioritizing working software over extensive documentation. Its favor's close customer collaboration, accommodating changes instead of rigidly adhering to contracts. Agile teams adapt to change rather than strictly following predefined plans, working in short cycles to remain responsive [6]. These principles make Agile well-suited for dynamic and evolving project environments, focusing on delivering concrete results early and continuously.

In contrast to traditional techniques, agile procedures establish a coherent and efficient project management framework by integrating user stories, story points, velocity, and the product backlog. Agile software development user or client necessities are taken in the form of user stories. User stories are short explanations of what a user expects a software product to achieve. These stories assist everyone on the team in quickly and easily understanding what users require from the product [11]. The brief description of user stories is described in chapter 2 section 2.5.

These user stories are then estimated using story points, in agile story points are a unit of measure used to estimate the relative size or complexity of user stories or tasks in a software development project. The story points are used to calculate velocity, a statistic that tracks how much work the team can finish in a sprint. This information is used to improve future planning and guarantee that workload expectations are reasonable. All of these elements are organized within the product backlog, a dynamic, prioritized list that guides the team's focus on delivering high-value features. Through the integration of various methodologies, Agile establishes a flexible process where every phase builds upon the other, culminating in the successful completion of projects that satisfy user objectives. This contrasts with traditional approaches, which may find it difficult to adapt to changing demands.

1.3 Agile Estimation Workflow

Agile project management uses the process of "agile estimation" to estimate the amount of work, time, or money needed to do a job or group of tasks. The main objective is to give teams a reasonable and comparable work estimate so they can efficiently schedule their sprints, releases, and project schedules in general [84]. There are six steps involved in the agile estimation that are explain in below section for example user stories, planning poker, story points, product backlog, velocity, re-estimation [44]. Figure 1.1 shows the Agile estimation workflow, illustrating the process of breaking down user stories and assigning effort using relative sizing techniques. It highlights collaboration between team members to ensure accurate and consistent estimates throughout the sprint planning.

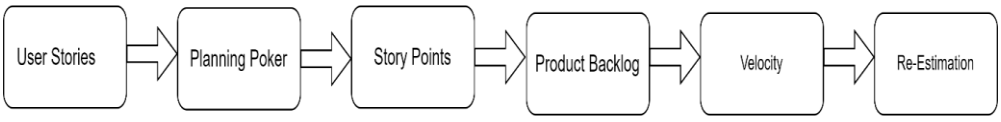

Figure 1.1: Agile Estimation Workflow

In the first step , agile software development user or client necessities are taken in the form of user stories as described in section 1.2. User Stories are the items are estimate using planning poker, serving as the foundation for your estimation process. To estimate the effort required for each user story using planning poker is a collaborative and consensus-building technique. Team members select cards with estimated values to collectively allocate story points to user stories. The estimates are revealed simultaneously, prompted discussion if they vary significantly. This process keeps the team in sync with the necessary tasks and guarantees precise, consensus-driven estimations. This method ensures accurate, consensus-driven estimates and aligns the team on the work required. The estimates generated in planning poker become story points [12].

In Agile, story points are a unit of measure used to estimate the relative size or complexity of user stories or tasks in a software development project. Unlike traditional time-based estimates (e.g., hours or days), story points are a more abstract and relative measure that takes into account various factors, including complexity, effort, risk, and uncertainty. They help

prioritize work based on its level of challenge. Story points guide the prioritization of user stories in the product backlog, with higher points indicating more complex tasks [13]. In agile software development, velocity is a metric used to measure the amount of work a development team can accomplish within a specified time frame, typically a sprint. The product backlog is a dynamic list of all user stories and tasks, ordered by priority. It serves as the to-do list for the project. User stories from the product backlog are taken into sprints, forming the basis for measuring progress through velocity [14]. By tracking the quantity of work finished in each sprint, velocity assists the team in determining its capacity and organizing their upcoming assignments.

In Agile software development, there are two types of velocities: initial and final velocity. Initial velocity is measured when the project begins. However, as the project progresses, various factors such as frictional forces (Negative influences in project productivity) and dynamic forces (Unpredictable forces) can affect it and final velocity is then calculated using these considerations to provide a more accurate measure of the team's performance over time, [40] the brief description of these forces described in chapter 2 sections 2.5. Velocity informs the team's ability to handle work in upcoming sprints and influences the process of re-evaluating and adjusting estimates [15]. Re-estimation involves adjusting story points based on new information, feedback, or changes in understanding, keeping estimates accurate. The results of re-estimation impact future user story estimation, creating a continuous feedback loop for improvement.

In essence, the agile estimation workflow begins with identifying user stories, estimating them with planning poker, converting estimates into story points, prioritizing the product backlog, measuring progress through Velocity, and refining estimates through re-estimation. These steps are interconnected, forming a cycle of continuous improvement in project planning and execution.The agile technique has been widely adopted in the software development industry in recent years. The main causes of project failure are inaccurate and lacking requirements. Estimating software involves forecasting the software product's size, required development efforts, project timelines, and projecting the final project cost.

Accurate cost estimation in project management is the most important and difficult assignment [4]. The agile technique asks team members to use their effort and level of difficulty

for the scrum rather than asking management to estimate the length. Technology teams use effort estimation to estimate how long it will take to develop a project or product, how many people they will require, and how much it might cost. The software development process depends on effort estimating because it enables teams to make sure a product is created and delivered on schedule.

When estimating effort, we determine the order in which the tasks need to be finished in order to complete the project. Next, we determine how long each task will take in person-hours or days. We get an estimate for the total activity by adding up the efforts of these separate tasks. Software is invisible, intangible, and complex, making it difficult to comprehend and predict its cost. Consequently, each estimation method employs a unique set of characteristics to estimate the software's cost.

1.4 Effort Estimation through Machine learning

For estimating effort, a variety of models are available, including non-algorithmic, algorithmic, and expert judgment techniques. Expert judgment is based on past projects with comparable parameters; if the parameters of the completed projects are almost the same as the present project, estimating the effort is easy [5]. Parametric techniques are used by algorithmic models, which make use of fixed-form formulas parameterized by past data. SLIM [7], Albrecht's Function Points [8], and COCOMO [6] are a few prominent algorithmic techniques. The shortcomings of algorithmic models have given rise to non-algorithmic models, which make use of machine learning and soft computing. Even though these models are available, new ones are always being created to get estimates that are closer to reality [9].

Machine learning approaches significantly improve effort estimation in agile software development by leveraging historical data to provide accurate and adaptable predictions. Machine learning models can analyze vast amounts of data to identify patterns and make predictions that traditional methods might miss These methods analyze complex project variables like team composition and task intricacies, enhancing estimation precision over traditional approaches. They also facilitate continuous improvement through iterative learning, ensuring estimates keep it up relevant as projects evolve. Additionally, machine learning

promotes transparency and consistency in estimation practices, fostering trust and informed decision-making among stakeholders [36]. Overall, these advancements optimize resource allocation and contribute to successful project outcomes in dynamic agile environments.

Ensemble learning in machine learning involves combining multiple models to improve prediction performance. In effort estimation, methods like stacking use a variety of models such as decision trees or regression models together to generate more precise estimates [45]. This strategy reduces biases and variability inherent in individual models, resulting in reliable predictions that consider diverse project factors.

Consequently, effort estimation processes software project data and forecasts resource needs using machine learning. In turn, ensemble learning improves this procedure by combining various models, guaranteeing more accurate forecasts for efficient project management and planning.

1.5 Problem Statement

Software effort estimation is a significant part of software development, as the success of project depends upon accurate estimation. Accuracy of effort estimation becomes challenging because of the project size, complexity and uncertainty. Agile software development with its iterative and adaptive nature, introduces unique dynamics that make accurate estimation of software projects a challenging task. "Traditional effort estimation approaches [88] may not be reliable for accurate estimates, especially in an agile development environment. In recent years, machine learning techniques have gained prominence in software engineering, offering new opportunities for improving estimation practices. The growing prominence of machine learning techniques in software engineering, there is limited research [16][17] or exploration into the application of ensemble models specifically for effort estimation in Agile software development projects. Ensemble models, known for their ability to combine the strengths of multiple algorithms, could offer greater accuracy in predicting effort. Their capacity to adjust to intricate patterns in Agile projects emphasizes the necessity of more empirical research and validation.

1.6 Significance of the study

Accurate software effort estimation and agile development are essential for figuring out the size and capability of the development team needed to finish a project successfully. Accurate estimates guarantee efficient resource allocation and timely completion of tasks, which directly impacts the quality and timeliness of the finished output. The estimating process is essential to the success of Agile projects since mistakes in these estimates have the potential to cause major project failures. While overestimating can result in wasteful resource use and longer project durations, underestimating can lead to missed deadlines, higher costs, and overworked team members. The project's outcome may be adversely affected by either scenario, which emphasizes how crucial it is to strike a balance through precise estimation. Precise approximations avert these problems, guaranteeing that projects stay on schedule and within budget.

Accurate estimates improve project predictability, which is necessary for Agile development's efficient sprint planning and on-time delivery. Teams are better equipped to set realistic objectives, make consistent progress, and interact with stakeholders when they have faith in their estimations. This consistency helps to guarantee that expectations are met for the project and fosters stakeholder trust.In the end, precise effort assessment enables more informed decision-making at every stage of the project. Teams can make well-informed decisions regarding scope and priorities, effectively manage risks, and guarantee resource utilization. Agile teams can improve their chances of delivering high-quality work on schedule and achieving more successful project outcomes by improving estimation accuracy.

1.7 Research Objective

The following is the research objective:

OBJ: To propose a model for effort estimation in agile software development using ensemble learning.

1.8 Research Questions

The following questions has been considered in order to achieve these study goals:

RQ: How to control the unnecessary packet flooding to reduce the energy consumption in the network initialization phase?

1.9 Research Contribution

What Precise predictions are essential for efficient planning, resource distribution, and on-time delivery in agile organizations. This study's primary objective was to use ensemble learning to increase accuracy in the context of Agile software development. In order to do this, we were aware of the constraints imposed by the modest size and lack of public accessibility to the current datasets. We made a concentrated effort to obtain more extensive and varied datasets directly from different software firms in order to address this problem. Our strategy concentrated on combining several machine learning models using the stacking method, utilizing their unique capabilities to create a more reliable and accurate prediction model. Our study's findings demonstrated a significant increase in estimation accuracy, proving the value of ensemble learning strategies in agile software development. By improving forecast reliability in Agile projects, this work helps to further the ongoing endeavor to improve project outcomes, resource management, and planning.

1.10 Summary

The chapter introduces the challenges in software development effort estimation, outlines the agile estimation workflow, identifies the problem, highlights the significance of accurate estimation, research objectives and research questions for further exploration.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

This chapter presents an overview of effort estimation models applicable to both traditional and agile software development methodologies. A wide range of estimation techniques has been introduced, including both conventional algorithmic models and modern learning-based approaches. As software projects grow increasingly complex, accurate effort estimation becomes a crucial element of successful project planning and execution. Inaccurate estimations often lead to project delays, cost overruns, and potential failures, underscoring the importance of reliable estimation methods in the software industry. The rapid evolution of software technologies has further emphasized the need for effective estimation practices. Various researchers have proposed diverse strategies to improve prediction accuracy, broadly classified into algorithmic, non-algorithmic, expert-driven, and machine learning-based methods. A dedicated section of this chapter reviews existing literature on effort estimation in agile software development. The literature review process involves examining and synthesizing previously published academic work relevant to the topic, providing insight into the structure, findings, and contributions of prior research. The focus here is on scholarly work related to "Effort Estimation in Agile," highlighting the range of methodologies proposed for work estimation in agile environments.

2.2 Effort Estimation in agile software development

The manager determines a team member's workload capacity under the waterfall methodology by estimating the amount of time needed for each task and allocating work based on the team member's total time available. On the other hand, the Agile methodology

approaches the assessment of a team member's capacity in a very different way. First of all, it emphasizes teamwork by giving tasks to the entire group rather than to a single person. Second, it deviates significantly from the waterfall approach by not evaluating work in terms of time in order to maintain the self-organization that is crucial to the methodology's success [18]. Team members estimate their work using effort and complexity levels in Agile procedures. Teams are not required to estimate their work using a particular approach by the Agile Methodology. It does, however, discourage using time-based evaluation and instead promote the use of more abstract metrics to gauge effort. The Fibonacci sequence, t-shirt sizes, numerical sizing, and even dog breeds are examples of common estimating techniques.

It is important that the group comes to a consensus on the selected scale so that each member of the group feels at ease with its values. The team gets together during the Sprint Forecasting meeting to determine how much work will be needed on the tales in the backlog [19] These assessments are essential to the Product Owner's ability to efficiently prioritize backlog items and project releases according to the team's velocity. Establishing a safe and unbiased environment encourages more realistic and collaborative estimation outcomes. This makes a fair assessment of the jobs' difficulty necessary. This approach fosters trust within the team and supports more sustainable sprint planning over time. Thus, in order to avoid pressuring the team to underestimate their effort and take on extra work, it is recommended that the Product Owner refrain from watching the estimating process.

Precautions should be taken to limit any effect on the estimation process, even when the team estimates between themselves. It is advised that each team member reveal their evaluations at the same time. The most widely recognized techniques for evaluating ASD are:

- Expert Opinion
- Analog
- Disaggregation

Each of the above methods can be used independently, but for better results, they should be combined.

2.3 Effort Estimation Methods

There are different effort estimation methods that are listed below:

- Traditional methods
  - Algorithmic based method
  - Non-Algorithmic method
- Learning based method

Figure 2.1 shows traditional software estimation methods, which include both algorithmic approaches like COCOMO and non-algorithmic techniques such as expert judgment. These methods rely on historical data, mathematical models, or human expertise to estimate cost, effort, and project duration.

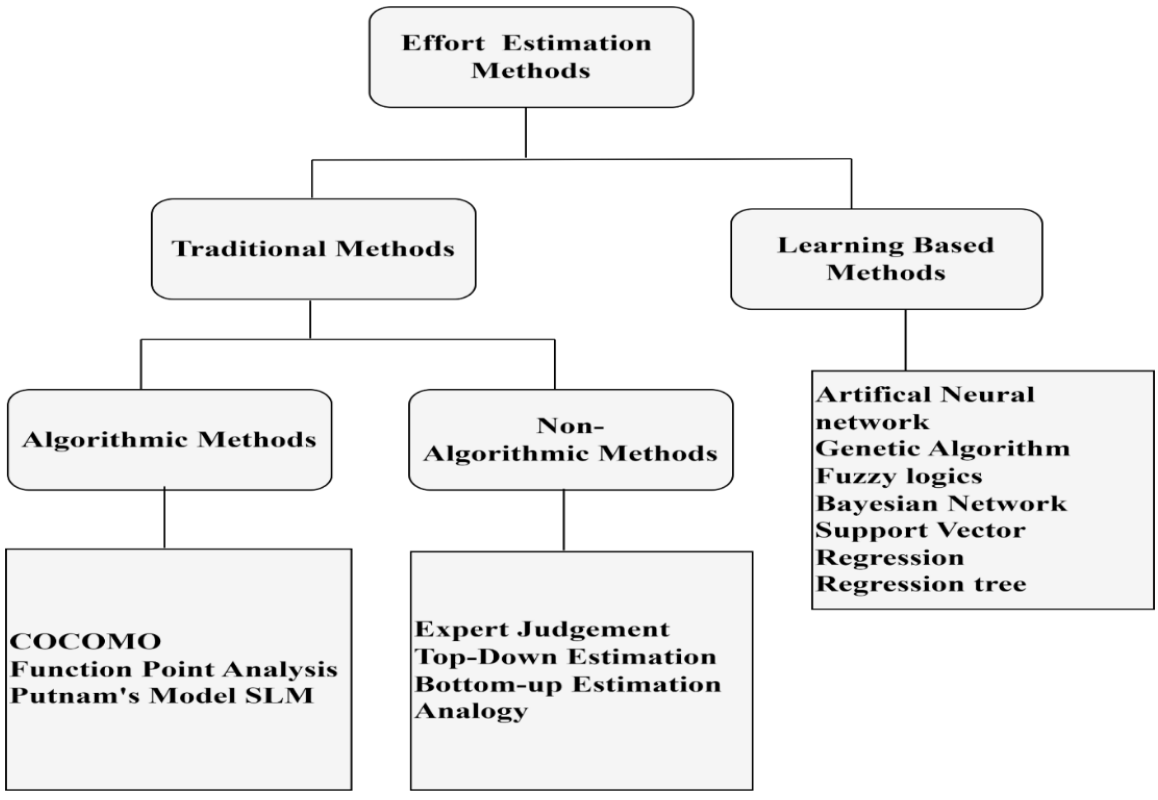

Figure 2.1: Effort Estimation Methods

2.4 Algorithmic Method

This approach for estimating efforts relies on mathematical equations that utilize cost factors related to product, machine, and personnel. These factors and their adjustment parameters are derived from previous projects. This method requires tailoring data and input parameters to align with the specific development environment. Although this method is deemed more precise than other methods like estimation by analogy and expert opinion, it can be challenging to quantify all cost factors, and some are disregarded in certain software projects [20]. One major drawback of this method is the inconsistency of estimates, as studies have indicated variances of up to 85-610% between predicted and actual estimates While system calibration to the particular development environment might improve accuracy, these techniques still result in 50–100% mistakes, and calibration is regarded as an extra burden [21][22]. This approach has led to the development of several models, such as function point analysis (FPA), Putnam models, and COCOMO models. Algorithmic methods include the following:

- Putnam model /Slim model
- Functional point analysis
- COCOMO Mode
- Communication Range: write briefly about it

2.4.1 Putnam Model

A useful model known as the Putnam model was introduced by L. Putnam in 1970 [7]. The Rayleigh Curve Function is used by this model to calculate the time and effort required to develop a software project. Based on this paradigm, Putnam's company created a proprietary suite called SLIM.

The equation 2.1 illustrates that software size is directly proportional to the cube root of effort and to the four-thirds power of development time, highlighting the non-linear relationship among these variables as identified in models like Putnam’s. This model's equation is:

$$S = E \times (Effort)^{1/3} td^{4/3} \qquad (2.1)$$

In the given equation, "td" stands for delivery time, "E" for environmental considerations, "S" for product size expressed in ESLOC (Estimated Source Lines of Code), and "Effort" for the total development effort expressed in person-years in the given equation. where:

'td' refers to the delivery time.

'E' represents the environmental factor.

'S' denotes the size of the product in ESLOC.

'Effort' represents the overall development effort in person-years. As shown in Equation 2.2, the total effort required for a project is modeled as a cubic function of development time, where D_0 is a constant reflecting team dynamics or staffing rate. This equation emphasizes that even small increases in development time can lead to significantly higher effort, based on non-linear project scaling models. Another important equation in this model:

$$Effort = D_0 \times t^3 \qquad (2.2)$$

Standalone software systems and other combinations fall in between these two extremes. The personnel build-up parameter, denoted by "$D0$" in this case, might range from 8 (for a new system with numerous interfaces) to 27 (for the reconstruction of an old system). A new software system usually takes more time and work to develop. During project planning, these differences have a big impact on scheduling and resource allocation tactics. Project managers can set more reasonable deadlines and expectations if they know where a system lies on this spectrum. On the other hand, because the code and logic of an existing software system have already been built, reconstructing it takes less time and effort. Systems with standalone software and various combinations lie in the middle of these two extremes.

2.4.2 Functional Point Analysis

As a result, he devised a new method to determine the size of software projects by linking it to the software's functionality. Albrecht believed that the measure of software size should be language- independent and relevant to software users. He aimed to quantify the software's functionality rather than relying on lines of code (LOC) as a measure [23].

- Inputs of the application
- Outputs of the application
- Provision for querying
- Internal data storage
- External Interfaces

These attributes were platform-independent and applicable to most software applications. They were tangible and visible to clients. Albrecht presented his research findings in a paper at an IBM conference in 1979, introducing function point analysis (FPA) as an advanced estimation method to determine the dimensions of software applications, including size, effort, productivity, and defect density.

Function point (FP) serves as the unit of measurement for determining the size of software applications. The software's functionality is categorized and identified based on the five attributes mentioned earlier in FPA. These functions are then evaluated for complexity (low, moderate, or high), and FP values are assigned accordingly. The total sum of FPs is further adjusted using 14 technical attributes. The cost of project development, in terms of hours or money, for a single unit is calculated based on data from previous projects [24].

Research explores various aspects of news content patterns and coverage, including photos [14], images, and videos across multiple news sources [25]. Methods exist to assess the impact of news content on digital social media platforms. The goal is to measure software functionality rather than relying solely on lines of code (LOC) as a metric.

2.4.3 COCOMO Model

Software project size can be estimated using the mathematical model COCOMO, which was created by Barry Boehm [26]. Based on an analysis of 63 software development projects that made use of procedural languages and the Waterfall methodology, it was first implemented at TRW Aerospace in the 1970s. COCOMO II, which could estimate contemporary software development projects and procedures, was developed in the 1990s as a result of improvements made to COCOMO [27].

To put it simply, COCOMO uses formulas to figure out how much a software development project will cost. After being modified to take into account the unique features of the current project, parameters are taken from previous project data. Three types of the original COCOMO model exist: basic, intermediate, and detailed COCOMO.

Software Development Modes

COCOMO categorizes software development projects into three different modes based on their complexity. While these modes utilize the same cost estimation relationship, they generate different cost estimates for projects of the same size.

Organic Mode: The development team is well aware of and conversant with the issue at hand when working in the organic manner. A small development team can handle such tasks with ease because there is a wealth of data from past endeavors.

Semi-detached Mode: The development team in this mode is made up of a mix of staff members with and without expertise. The group's knowledge of the ongoing initiative is somewhat restricted.

Embedded Mode: The embedded mode encompasses software systems that are more complex and tightly integrated with hardware. Developing such projects requires a higher level of creativity and expertise.

COCOMO-II: This represents the latest version of Boehm's well-known COCOMO model, initially introduced in 2000. While the original COCOMO model has shown its effectiveness in conventional software engineering, it no longer fits the evolving landscape of software development practices. The primary objective behind the development of COCOMO II was to adapt to the modern software engineering techniques in use today. COCOMO-II is available in three primary variations:

Application Composition Model: This model is designed to calculate the time and effort required for projects developed using modern RAD GUI builder tools. It's particularly useful for estimating resources needed for projects that involve the composition of applications.

Prototype Model: This model comes into play during the initial phases of a project when the project's full architecture has not yet been finalized. It helps in estimating project costs and timelines before making final architectural decisions. This model relies on five scale variables, seven cost components, and function points (or lines of code when applicable), although it may be limited by a lack of sufficient early-stage data.

Post-Architecture Model: As the project progresses and the upper-level design is completed, the COCOMO 2 model is used, offering the highest level of detail. This model incorporates new equations, updated line-counting methods, and additional cost drivers to provide a more accurate estimation of the resources required at this stage. As Equation 2.3 illustrates, the effort required to complete a software development project is modeled as a function of the system’s size raised to an empirically derived exponent, and further adjusted by the cumulative impact of 17 distinct cost drivers. This formulation, originating from the COCOMO II model, allows for nuanced effort estimation that accounts for both scale-related and context-specific factors, thus improving prediction accuracy over simpler estimation techniques.

$$Effort = (PersonMonth) = A \times (Size)^{E} \times \prod_{i=1}^{17} E\,M_i \qquad (2.3)$$

In equation 'A' is a calibration factor. The organization's prior project data tends to be used to adjust it. The scale factor "E" is dependent on five factors. Team cohesiveness,

development flexibility, architecture and risk management, process presence, and process maturity are all included in this list.

2.5 Non-Algorithmic Methods

In a non-algorithmic model, past project experiences can be used, which is comparable to underestimating a project [88]. Non-algorithmic methods include the following:

- Expert Opinion Method
- Delphi Technique
- Analogy
- Top down
- Bottom Up

2.5.1 Expert Opinion Method

The accuracy of this approach relies heavily on historical data from completed projects and the experience of professionals within a relevant development environment. However, research conducted by VIGDER & KARK [28] has indicated that many cost estimators tend to avoid consulting previous projects because it's often challenging for experts to perceive how such knowledge can enhance estimate accuracy. When multiple expert opinions are considered, their estimates are typically combined using a weighted average.

2.5.2 Delphi Technique

As per the insights of professionals, the Delphi technique stands out as one of the most renowned methods. It draws its name, "Oracle," from the ancient Greek prophet. In the Delphi

technique, expert opinions are collected and harmonized through a sequence of iterative dialogues, meetings, and surveys aimed at achieving a collective agreement within a group.

Originally, RAND introduced this method in 1950 to predict the outcomes of combat scenarios [29]. However, it's worth noting that it might have connections to various other fields as well. The Delphi Technique initially lacked group discussions, but the Wideband Delphi method addressed this gap by incorporating group interactions, enhanced engagement, and communication during evaluation rounds [26]. It proves particularly valuable when empirical data is scarce, and estimations heavily rely on expert judgment. The effort estimation process in Delphi consists of the following phases:

- Each expert receives product specification forms
- Each expert receives product specification forms
- The project manager convenes a group meeting where specialists deliberate on the project details
- Experts complete the provided forms
- A summary of the estimations is compiled and shared
- To address areas with significant variations in expert opinions, a group discussion is arranged
- Specialists revisit the forms for further input and refinement

2.5.3 Analogy

Shepperd has introduced another valuable and pragmatic approach for estimating software projects, referred to as "Estimation by analogy" in [28]. With this method, the initial step involves identifying and evaluating all past projects that closely resemble the current one [31].

This technique involves assessing the characteristics of the planned project before drawing comparisons with previously completed projects. Instead of selecting projects that are completely identical to the current one in every aspect [32], the cost of the current project is

determined by examining the cost of a comparable project that has already been successfully executed.

2.5.4 Top down

Nonetheless, it's essential to acknowledge that top-down estimation tends to be less precise when compared to other cost estimation techniques, primarily because of the inherent uncertainty in project scope [33]. The notable shortcomings of this approach include the potential omission of lower-level system components and a lack of distinction for lower-level specialized issues. Consequently, top-down estimation is often not used as the final cost estimate but rather as a tool for project selection. Once a project is selected, other cost estimation techniques are typically employed to provide a more comprehensive assessment of the project's overall expenses.

2.5.5 Bottom Up

This approach involves assessing the cost of individual components and then aggregating these costs to arrive at the overall estimate for the system [34]. To initiate a bottom-up estimation, it's essential to first break down the entire software product into various smaller work products or components. This can be achieved, for instance, by employing a work breakdown structure.

2.6 Basic Terminologies for effort estimation in agile software development

In this section, basic terminologies for effort estimation in agile software are explained below.

2.6.1 User Story Size

A user story is a succinct, plain-spoken statement of the system's business requirements that is supplied by the end user in agile software development. System requirements are defined at a high level, covering the "who," "what," and "why" questions. These user stories are usually brief enough to fit on a small notecard. In Agile development, a user story's size plays a critical role in the work estimation process [11]. This size is based on several criteria and has a substantial impact on the effort needed for a particular user narrative.

- Atomic Large
- Non-Atomic Medium
- Non-Atomic Large
- Atomic Medium
- Small

2.6.2 User Story Complexity

Complexities in user stories add uncertainty to the estimation process. In Agile Software Development (ASD), requirements are collected using user stories. Although effort estimation techniques primarily depend on user stories, many methods often neglect the specific characteristics of individual user stories. Each Story Complexity Factor is assigned a weight (1 to 5), and the complexity of the User Story is then calculated using the following formula [44]. Equation 2.4 shows that the total story complexity is computed as the weighted sum of twelve contributing factors, each representing a different dimension of complexity within a user story. The weights reflect the relative significance of each factor, allowing for a more granular and customized assessment of complexity in agile software projects[44].

$$Story\ Comy(\mathrm{SC}) = \sum_{k=1}^{12}(SC * Weight) \qquad (2.4)$$

User Story Effort calculation: The effort needed for a specific story is figured out using two factors size and complexity and it's calculated using a simple formula. As shown in

Equation 2.5, the Effort Score (ES) is defined directly as the complexity value of a task or user story. This simplified relationship is useful in agile estimation practices, where complexity serves as a proxy for the effort required, enabling faster and more intuitive planning during sprint and backlog sessions[44].

$$ES = complexity \quad (2.5)$$

Efforts for the entire project will be summarized in all distinct stores. As illustrated in Equation 2.6, the cumulative effort score is obtained by summing the individual effort scores of all k user stories or tasks. This total provides a measure of the overall workload or complexity for a sprint, iteration, or project phase, supporting more accurate planning and resource allocation in agile project management.

$$\sum_{i=1}^{k}(ES)i \quad (2.6)$$

In this case, the effort for a single story is denoted by ES, while the effort for the entire project is denoted by E. A story point, which represents the quantity of work finished in a certain amount of time, serves as the unit of effort.

2.6.3 Friction Forces

Newton's First Law states that any force hindering the movement of an object due to contact with other bodies is termed friction. Friction forces, according to this law, negatively influence project productivity by diminishing team velocity. While project managers or developers can mitigate these forces, complete elimination is not possible. Figure 2.2 depicts the various friction forces that can hinder progress in Agile projects, such as miscommunication, unclear requirements, and resistance to change, which affect team efficiency and project outcomes [44].

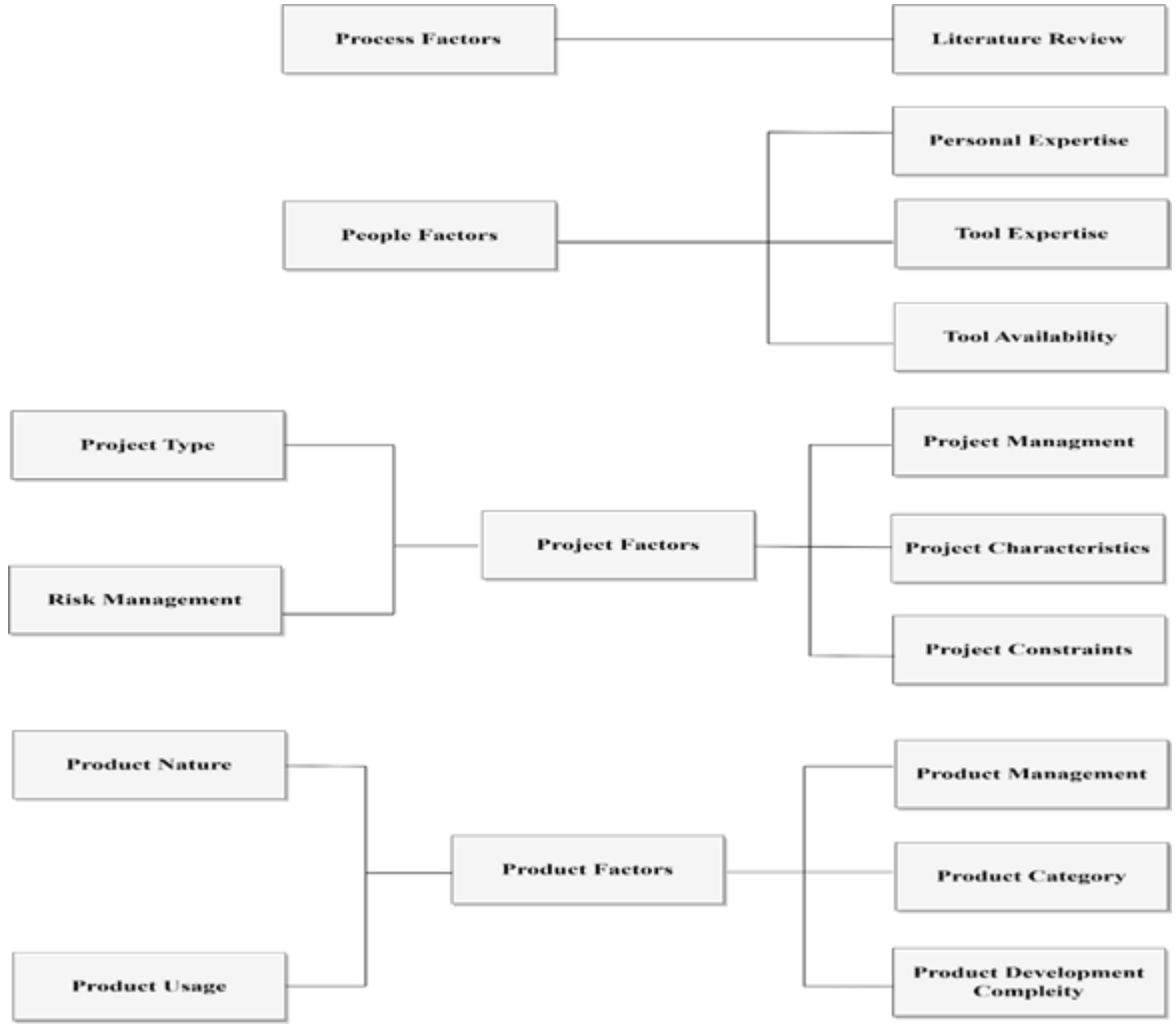

Figure 2.2: Friction forces for agile projects

In simpler terms, friction forces impede project progress, and although efforts can be made to reduce their impact, complete removal is not achievable. In software projects, friction forces are assigned weights ranging from a range of 1 to 3, where 3 represents the maximum intensity and 1 signifies moderate intensity[44]. As presented in Equation 2.7, the total functional requirements (FR) are calculated as the sum of 14 individual functional factor scores. Each factor represents a specific aspect of the system's functional needs (e.g., inputs, outputs, interfaces), and their aggregation provides a quantitative measure of the system's overall functional complexity. Friction forces are calculated as:

$$FR = \sum_{i=1}^{14}(FF)i \qquad (2.7)$$

Then, by adding up all of the friction forces, the total friction is determined. The entire sum of weights for all friction forces is obtained by dividing this sum by 42, with the maximum value of 3 for each force. This result is then multiplied by 3 to yield a score of 3, indicating the highest level of friction.

In simpler terms, the intensity of friction in a project is determined by assigning weights to individual forces, summing them up, and normalizing the result to obtain a friction score.

2.6.4 Dynamic Forces

Unexpected and unpredictable events usually result from dynamic or variable forces. They may cause a project to stall, momentarily lowering momentum. They usually have a short-term influence, even though they can have noticeable impacts. Dynamic or variable forces, to put it simply, are unforeseen circumstances that momentarily halt a project and impede its advancement [44].

As shown in Equation 2.8, the (DF) is determined by summing the values of nine contributing value factors, each representing a specific influence on the system's complexity or performance. Dynamic force DF calculated as

$$DF = \sum_{i=1}^{9}(VF)i \qquad (2.8)$$

Dynamic forces include expected team change, introduction to new tool, Vendor defect, Personal issues expected relocation, expected ambiguity in detail, expected delay in stakeholder response and team member responsibilities outside the project as shown in figure 2.3.

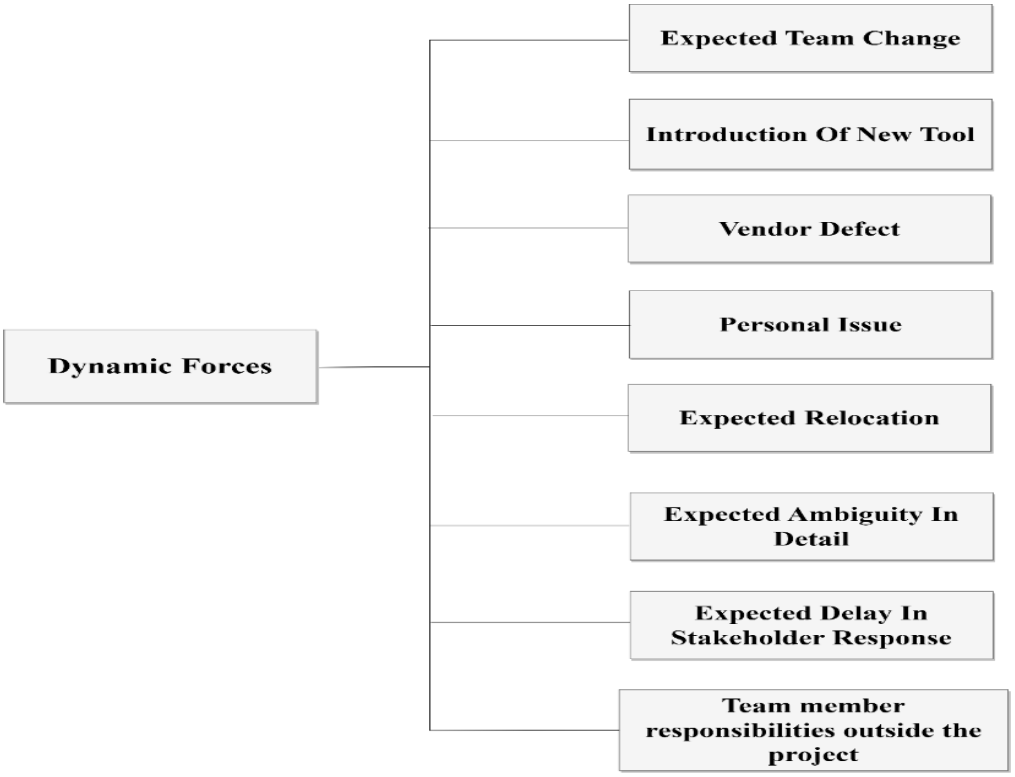

Figure 2.3: Dynamic Forces for Agile Projects

2.6.5 Deacceleration

Deacceleration is the measure of how velocity decreases over time. In my scenario, it is calculated by multiplying Dynamic Forces (DF) by Friction Force (FR). In simpler terms, the slowdown in velocity is determined by the combined impact of Dynamic Forces and Friction Force. The final team velocity (V) is achieved by optimizing Vi, considering the decelerating effects of both Friction Forces and Dynamic Forces. In simpler terms, the team's ultimate speed is determined by fine-tuning Vi, considering the forces that slow down progress such as friction and dynamic factors [44]. As illustrated in Equation 2.9, the deacceleration factor is calculated as the reciprocal of the product of functional requirements and the degree of influence. This value reflects how increased system functionality and complexity can contribute to reduced development efficiency. As shown in Equation 2.10, the final velocity V is calculated by raising the initial velocity to the power of the deacceleration factor The final velocity is calculated as:

$$Deacceleration = \frac{1}{FR*DF} \qquad (2.9)$$

As shown in Equation 2.10, the final velocity V is calculated by raising the initial velocity to the power of the deacceleration factor the final velocity is calculated as:

$$V = (Vi)^D \qquad (2.10)$$

2.6.6 Team Velocity

Team velocity is determined by dividing the Total Unit of Effort by the sprint size. In simpler terms, it's a measure of how much work a team can complete in a single sprint, calculated by looking at the effort expended and the duration of the sprint[40]. As illustrated in Equation 2.11, team velocity is determined by dividing the total completed effort by the duration of the sprint. This metric serves as a key performance indicator in agile project management, enabling teams to forecast future work capacity, assess progress, and plan sprint backlogs more effectively. A stable or improving velocity over time reflects team maturity and process optimization.

$$Team\ Velocity = \frac{\text{Unit of Effort Completed}}{Length\ of\ Sprint} \qquad (2.11)$$

2.6.7 Completion Time

Completion Time (T) is calculated as the total time required to finish the entire project. In simpler terms, it represents the overall duration needed to complete the project from start to finish. As shown in Equation 2.12, the estimated time to complete a task or project is calculated by dividing the total effort by the team's velocity. This provides a straightforward way to forecast completion time, assuming a constant team performance throughout the duration of the

work. As illustrated in Equation 2.13, time is estimated by summing the effort scores for all tasks and dividing the result by the deaccelerated velocity. This refined formula takes into account both the varying effort per item and the reduced team performance due to complexity or system influences, resulting in a more realistic estimation of development time.

$$T = \frac{E}{V} days \qquad (2.12)$$

$$T = \frac{\sum_{i=1}^{n} (ES)i}{(Vi)^{D}} days \qquad (2.13)$$

To convert the time unit (T) from days to months, you divide it by the number of working days per week (WD). In simpler terms, the duration will be expressed in months instead of days, by dividing the total time by the number of working days per week.

2.7 Learning Based Methods

Learning-based approaches rely not on mathematical equations but on analytical comparisons and interpretations of past completed tasks [35]. These methods require access to knowledge about previous projects that share similarities with the one being estimated. Historical datasets play a pivotal role in the estimation process. Machine learning techniques are under the category of learning-based approaches and are further divided into supervised and unsupervised learning models. The following sections will have more information.

2.7.1 Supervised Learning

In supervised learning, the machine undergoes learning under supervision, utilizing a model capable of making predictions based on labeled data. Labeled data signifies that the anticipated output is already known [36]. There are two primary categories within supervised learning.

1. Regression

2. Classification

Regression: In supervised learning, regression is a type of algorithm used for predicting continuous numerical values. In regression, a model is trained using input-output pairings from a dataset in which the output variable is either a real or continuous value. Regression analysis aims to identify the relationship between the input variables and the continuous output so that the model may predict results for previously unseen data [37]. Regression, to put it simply, helps one understand how input variables interact and help predict continuous outputs. A basic example is linear regression, which models the relationship between variables as a linear equation. Other regression algorithms, such as support vector regression or polynomial regression, may also be used, depending on how complex the data relationships are.

Classification: Algorithms used to assign instances to specified classes or categories or forecast discrete categorical labels are referred to as classification in supervised learning. Training a model on a labeled dataset with an output variable that indicates a class or category is the aim of classification. The class of novel, unseen cases is then predicted by using this trained model. In order to predict the class membership of future samples, the algorithm analyzes patterns and correlations found in the input data throughout the classification process. Classification tasks are frequently used in image recognition (classifying objects to specified categories), sentiment analysis (classifying language as positive, negative, or neutral), and spam detection (classifying emails as spam or non-spam) [37]. Numerous classification algorithms are available, including logistic regression, decision trees, and support vector machines.

2.7.2 Unsupervised Learning

When an algorithm is given unlabeled data and instructed to find patterns, correlations, or structures on its own, the process is known as unsupervised learning. In contrast to supervised learning, the algorithm receives no predefined labels or target outputs to guide it. Rather, the program looks for hidden patterns or clusters by investigating the data's inherent structure. Unsupervised learning tasks are often categorized into two main types:

1. Clustering

2. Association

Clustering: In unsupervised learning, clustering is a technique that involves grouping similar data points together based on certain features or characteristics. Unlike supervised learning, where the algorithm is provided with labeled data and a specific target to predict, clustering algorithms work with unlabeled data and seek to uncover inherent patterns or structures within the dataset [38].

Association: Association in unsupervised learning refers to the task of discovering interesting relationships or patterns in data without explicit labels. The primary goal of association analysis is to identify associations or correlations among variables within a dataset. This is particularly useful for finding hidden connections between different features or items [38].

Various techniques fall under the umbrella of learning-based strategies, including artificial neural networks (ANN), fuzzy logic models, case-based reasoning, evolutionary computation, and combinational models, among others. Artificial neural networks (ANNs) are particularly valuable for understanding the intricate relationships between cost drivers and effort in effort estimation [35]. ANNs possess the capability to learn from historical data; they are trained using a dataset to provide suitable results. They are very good at managing uncertainty in software projects because of their capacity to generalize from intricate, nonlinear patterns. Additionally, when more pertinent data becomes available over time, ANNs can continuously enhance their predictions. These systems employ soft computing techniques, including feed-forward.

Nonetheless, these soft computing techniques are typically employed alongside existing algorithmic methods to enhance the model, and this introduces a similar limitation as that seen in traditional algorithmic approaches. Interestingly, it has been noted that many learning-based techniques tend to yield more precise estimations when compared to conventional algorithmic methods. Multilayer perceptron’s, sigmoid functions, and back-propagation algorithms, to forecast effort.

2.8 Machine Learning Techniques

2.8.1 Support Vector Regression

Discuss Physical Layer Issues found in literature. Support Vector Regression (SVR) is a machine learning technique employed for regression tasks. Its objective is to maximize the margin between the line or curve and the nearest data points (support vectors), thereby identifying the best-fitting model for the data. SVR excels in handling outliers, missing values, and data quality issues.

If categorical variables are present, encode them (e.g., assign numerical values to labels such as "small," "medium," and "large"). Relationships between input features and output variables that are non-linear are modeled. This is made possible by the application of kernel functions, which enable SVR to identify intricate patterns in the data. When it comes to outliers, SVR is less susceptible than certain other regression techniques. Regularization parameters in SVR, like the cost parameter C, aid in preventing overfitting. The regularization parameter C and kernel parameters are examples of hyperparameters that greatly influence SVR performance. It can be difficult to find the ideal set of hyperparameters and may take a lot of tweaking. Even though SVR often handles outliers effectively, noisy data may cause it to perform poorly [39][40]. As presented in Equation 2.14, this linear equation represents a simple regression model where the predicted output Y is computed as a weighted sum of the input x, adjusted by a bias term b. This fundamental form is widely used in machine learning and statistical modeling to establish relationships between variables and to make predictions based on observed data. The equation of hyperplane as follows:

$$Y = wx + b \qquad (2.14)$$

As shown in Equation 2.15, this is a standard linear equation where the product of the input and its weight is adjusted by a bias term to produce a result a. As shown in Equation 2.16, the equation represents a linear relationship where the sum of the weighted input equals the negative of a value a. This form is often encountered in scenarios where the outcome represents a cost, loss, or deficit. It may also appear in classification models or regression problems where

directionality (positive or negative influence) of the result is significant. Then equation of descion boundary

$$wx + b = +a \qquad (2.15)$$

$$wx + b = -a \qquad (2.16)$$

SVR calculates the separation between the hyperplane, or boundary line, and the best line that fits within that threshold value. Support vectors in SVR are the important data points on either side that are closest to this hyperplane. The boundary line in SVR must be defined using these Support Vectors.

The figure 2.4 shows how SVM finds the best separating line (hyperplane) between two groups by using the closest points, called support vectors, to create the widest possible margin.

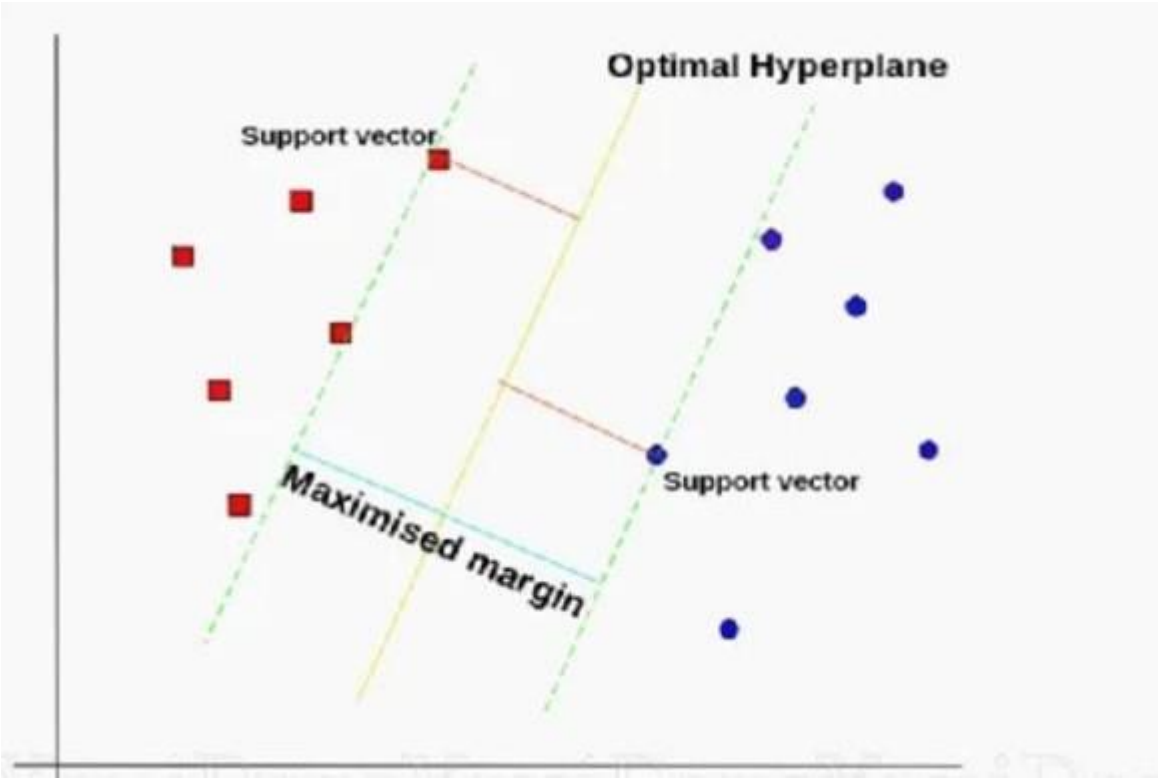

Figure 2.4: Support Vector Regressor

2.8.2 Linear Regression

A continuous result variable (dependent variable) can be predicted using linear regression, a statistical modeling approach, by taking into account one or more predictor factors (independent variables). It makes the assumption that the variables have a linear connection, meaning that variations in the predictor variables cause the result variable to change continuously. If the four basic criteria of linearity, independence, homoscedasticity, and normality are satisfied, linear regression is a simple and fundamental method that yields insights [41]. However, it might have trouble with problems like multicollinearity from highly correlated predictors and complex, nonlinear interactions between variables.

The dependent variable, or output, is displayed on the Y- axis in linear regression, while the independent variable, or predictor, is represented on the X-axis. The graph shows how these variables are linearly correlated, with the blue line showing the closest- fitting straight line. With the goal of minimizing the distance between the line and the data points, the best-fit line in linear regression is determined using the traditional slope-intercept form. As illustrated in Equation 2.17, this is the equation for a simple linear regression model where the output is predicted based on a linear combination of the input. The parameter represents the model's intercept, while β is the coefficient that defines how changes in the input affect the output.

$$Y_i = \beta_0 + \beta_1 X_i \qquad (2.17)$$

Where Xi is the independent variable, β_0 is the constant/intercept, β_1 is the slope/intercept and Yi is dependent variable.

2.8.3 K- Nearest Neighbor Regressor

The K-Nearest Neighbors (KNN) Regressor is a non-parametric, instance-based supervised learning technique used for regression tasks. In contrast to conventional regression techniques, KNN Regressor makes predictions for new data points based on their closeness to current data points by using the training data without first learning a model. The method does

nothing more than store the training dataset during the training phase. This method identifies the k-nearest neighbors from the training set and uses that information to predict the target value for a new data point. A distance metric, like the Euclidean distance, is typically used to assess "neighborliness". In regression tasks, the weighted average of the target values of the new data point's k-nearest neighbors is frequently used to get the predicted value for the new data point. Usually, the weights are inversely correlated with the distance to the new site. The KNN Regressor is simple to use and comprehend. Regarding the underlying data distribution, it doesn't make any strong assumptions [42].

The mathematical expression for a k-nearest neighbor (KNN) regressor entails predicting the target variable by computing the average or weighted average of the k-nearest neighbors. In certain instances, weighted averages may be employed, where closer neighbors exert more influence on the prediction compared to farther ones. The specific formula for calculating weights may vary depending on the implementation. As a non-parametric method, KNN does not make assumptions about the structure of the underlying data, allowing it to adapt to complex data relationship. KNN is applicable to jobs involving both classification and regression. Because of its adaptability, it can be used in a variety of issue domains.

There's no formal training period. Since the model is the training data, it works well in online learning environments where the distribution of the data may fluctuate over time. Calculating the distances between each new data point and every point in the training set is a step in the KNN prediction process. The computing cost of this can be high, particularly for huge datasets. KNN is susceptible to data noise and outliers. Predictions can be greatly impacted by outliers, especially when a small number of k is used. The selection of a suitable value for k has a significant impact on the performance of KNN.

2.8.4 Descion Tree

A supervised machine learning method for both regression and classification issues is the decision tree. Recursively dividing the dataset into subgroups according to the most significant feature at each node is how it operates. Until a halting condition is satisfied, like reaching a maximum depth or a particular purity level, this procedure keeps going. Because of

their ease of use, interpretability, and capacity to handle both numerical and categorical data, decision trees are frequently employed.

The technique chooses the characteristic such as mean squared error for regression or Gini impurity for classification that best separates the data into subsets. The root node is this characteristic. Next, depending on the chosen features, the dataset is divided at each internal node to create branches that link to other nodes [43]. Until a halting condition is satisfied or a predetermined depth is reached, the process keeps going. At the terminal nodes, or leaves, predictions are formed according to the majority class (for classification) or mean value (for regression) of the cases in that leaf. Based on the values of the instance's features, the decision tree follows the path from the root node to a leaf node in order to provide predictions for a new instance.

Decision trees are simple to comprehend and analyze since even non-experts may quickly grasp their visual form. They require little preprocessing to handle mixed numerical and categorical variables, and they may be applied to a variety of datasets with minimal presumptions about the distribution of the underlying data. Non-linear correlations and interactions between features can also be captured by decision trees. Nevertheless, decision trees can overfit, particularly if they are deep and pick up noise in the training set. They can also be unstable, with slight alterations in the data perhaps producing an entirely new tree.

2.9 Ensemble Learning

In machine learning, ensemble learning is like a collaborative approach. It uses a combination of techniques known as base learners or inducers to make decisions rather than depending on just one. These base learners are algorithms that build models, such as classifiers or regressors, by learning from labeled samples [45]. Ensemble learning involves integrating the attributes of multiple models to enhance prediction accuracy and resilience in supervised machine learning tasks. There are different techniques of ensemble learning.

- Bagging

- Boosting
- Stacking

A simple ensemble learning architecture comprises a set of diverse base models, each trained independently on a dataset. Three different models Model 1, Model 2, and Model 3 are being trained on the assigned training dataset. These models differ in nature, and some of their variations like decision trees or support vector machines offer distinct insights into the process of learning. Then the average of individual models was calculated. Based on the collective input data, every trained model offers a forecast. The final prediction has been calculated by combination the distinctive predictions of each of its base models, it is typically subject to an aggregation process [46]. Figure 2.5 shows the architecture of ensemble learning, where multiple base models are combined to improve predictive performance by leveraging their individual strengths through methods like bagging, boosting, or stacking.

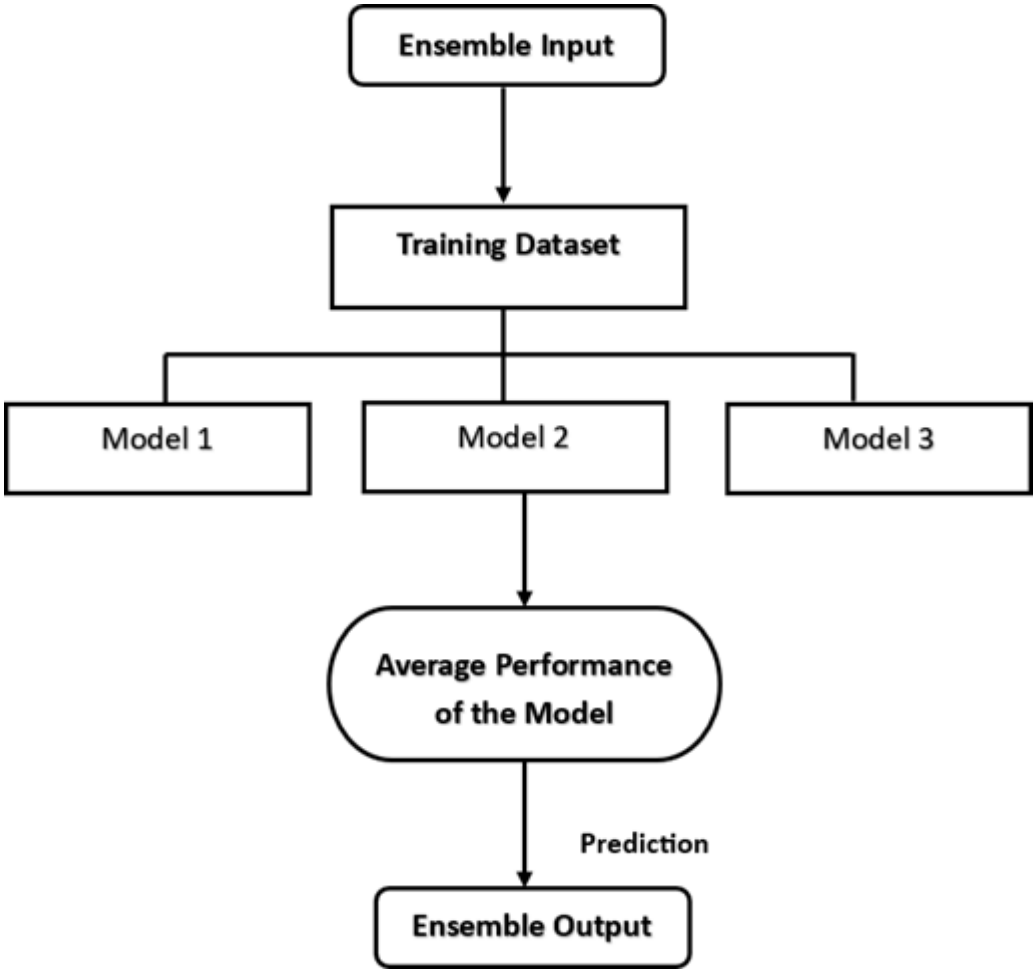

Figure 2.5: Architecture of ensemble learning

2.9.1 Boosting

In machine learning, boosting is a well-known ensemble learning technique where multiple weak learners (models that perform slightly better than random chance) are combined to form a strong learner. The objective underlying boosting is to train models sequentially, with each new model emphasizing data that prior models struggled to accurately identify. The key idea behind boosting is to focus on the mistakes made by the previous models and give more weight to the misclassified instances in the subsequent models [47]. The first phase in the boosting technique is to train the weak learner on the completion. This weak learner could be a simple model with little depth, such as a decision tree with limited length. After the first model is trained, the weights of the misclassified samples are raised. This means that the next inexperienced learner will focus more on the previously misclassified examples.

Subsequent weak learners are trained on a modified dataset, focusing on misclassified cases. This process repeats until a stopping criterion is met. The final strong model is formed by combining the predictions of all weak learners, with more weight given to accurate models. Figure 2.6 illustrates the boosting technique, where sequential models correct previous errors to improve prediction accuracy.

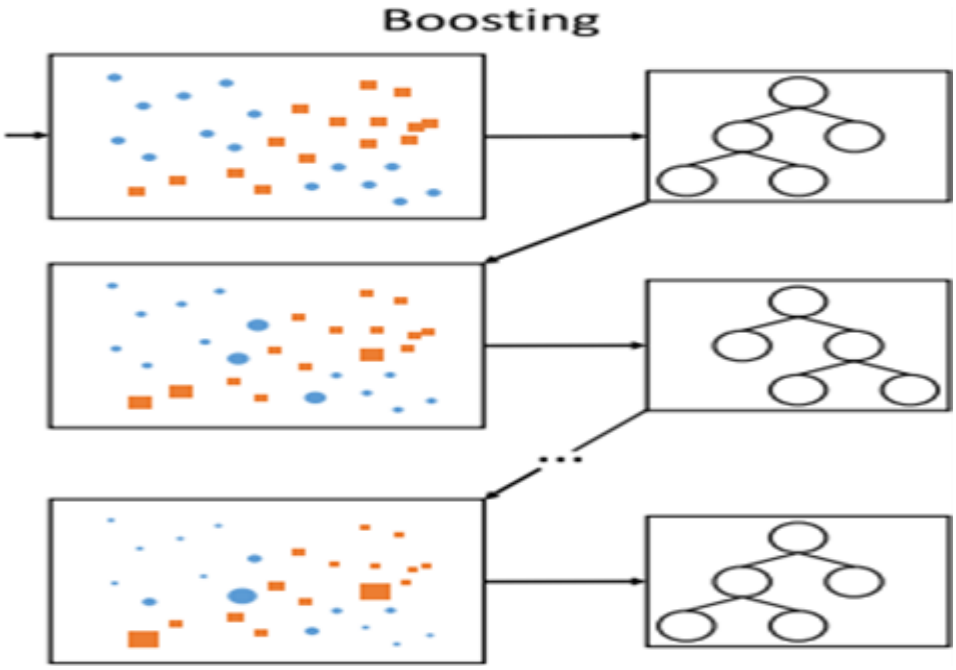

Figure 2.6: Ensemble learning technique boosting

2.9.2 Bagging

By merging predictions from several models trained on various subsets of the training data, an ensemble learning technique known as "bagging," short for Bootstrap Aggregating, is intended to improve the stability and precision of machine learning models. This method uses techniques similar to regression averaging and classification voting to aggregate predictions. Through bagging which is especially useful for less accurate learners many models are trained on different data subsets and their outputs are combined to reduce prediction variation [47]. This idea is expanded upon by Random Forest, a bagging variation, which randomly selects features at every stage of the decision tree building process [47]. Random Forest assesses every feature for split decisions, in contrast to conventional decision trees. Because bagging operates in parallel by nature, it can take advantage of parallel processing capabilities. Concurrent model training is made possible by the independent training of each base model on distinct subsets of data. Bagging is a good choice for effective computational scaling during training because of this parallelization [48]. Figure 2.7 illustrates the bagging ensemble learning technique, where multiple models are trained independently on different data subsets and their predictions are averaged to improve accuracy and reduce variance.

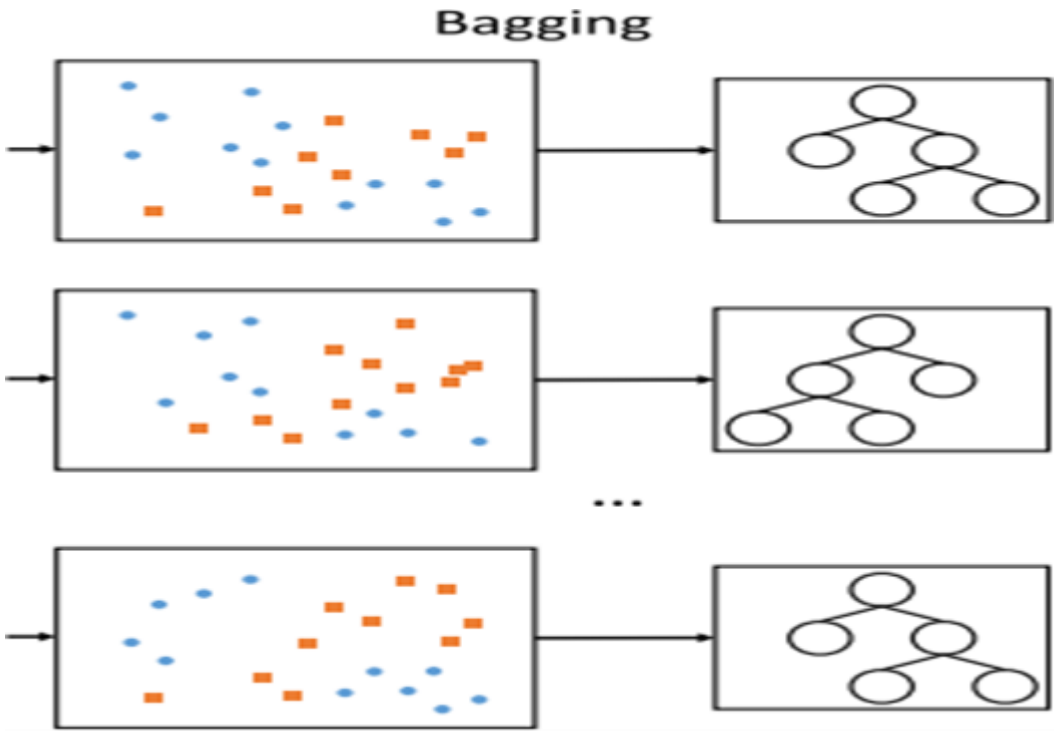

Figure 2.7 Ensemble learning technique Bagging

2.9.3 Stacking

Stacked Generalization, also known as Stacking, is an ensemble learning strategy that includes training a model to aggregate the predictions of numerous base models. Instead of merely averaging or voting on individual model predictions, stacking constructs a meta-model that takes into account the strengths and shortcomings of the basic models.

The main concept is to figure out how to best integrate the forecasts of the base models to increase overall performance [49] Here's a detailed description of how stacking works. On the training dataset, numerous distinct base models are trained. To ensure diversity, these models can be of various sorts or trained with various algorithms. Each base model generates predictions using the same set of instances. The training dataset is split into two parts, with one section dedicated to training the base models and the other serving as a reserved set. Figure 2.8 illustrates the Stacking ensemble learning technique, where multiple base learners are trained independently and their outputs are combined by a meta-learner to improve overall prediction accuracy.

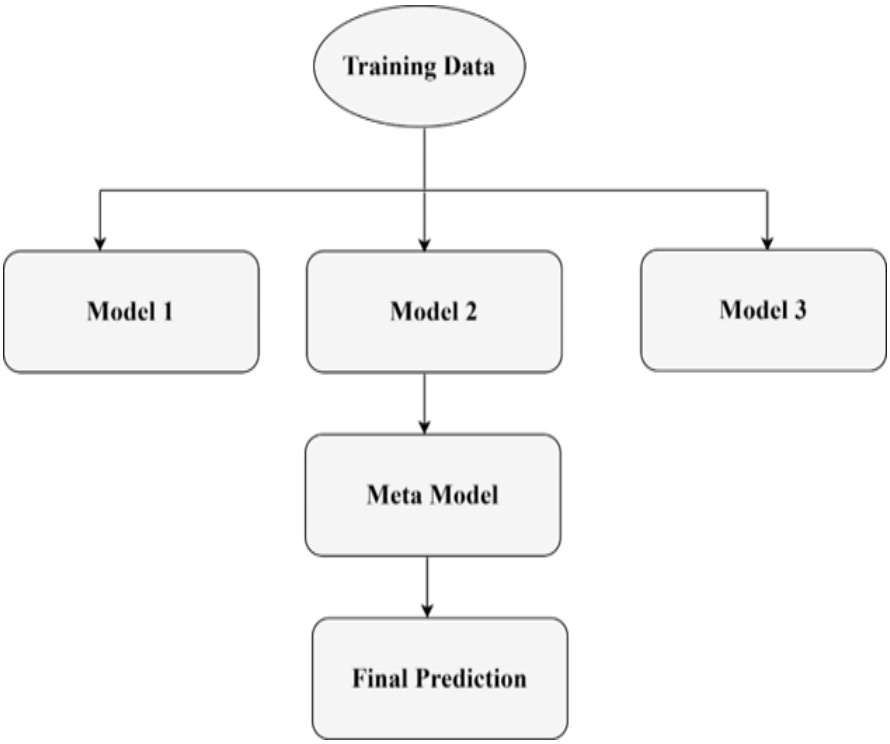

Figure 2.8: Ensemble learning technique Stacking

Predictions from each base model have been obtained by utilizing the k-cross validation. These predictions become the meta-model's input features. The training of a meta-model (alternatively referred to as a blender or combiner) involves utilizing the predictions derived from the k-fold cross- validation of the base models. In the final prediction, the base models have utilized the predictions as input features Cross-validation can be used during the stacking process to mitigate issues.

The dataset is separated into numerous folds in cross-validation, and stacking is conducted iteratively on distinct training and validation sets to measure the model's generalization performance. Stacking is a versatile ensemble technique that may be used to solve a variety of machine-learning problems, such as classification and regression. Figure 2.8 illustrates the stacking ensemble learning technique, where multiple models are trained and their predictions are combined by a meta-learner to enhance overall performance.

2.10 Data Augmentation

Data augmentation is a technique used in data science and machine learning to add slightly modified copies of preexisting data points to the dataset in order to artificially increase it. This method seeks to improve machine learning models' robustness and performance, particularly when the initial dataset is small or lacking diversity. Depending on the type of data image, text, audio, or tabular different techniques for data augmentation are used [50].

One significant issue with software effort estimation is the lack of data availability. Obtaining data can be costly and time-consuming, which leaves firms with few training instances. Creating fictitious projects based on finished ones turns out to be a useful strategy to lessen this problem. These fictitious projects are made to be sufficiently tiny to enable reasonable adjustments to the real project values while maintaining their essential features. They should be significant enough to add new items to the dataset and make significant contributions at the same time [51]. Each project is replicated to create two new records, which are then added to the dataset by adding random noise values to the effort and velocity characteristics. Although the noise level is carefully managed to prevent data disruption, it is high enough to set new records. With only little modifications, this strategy guarantees that

small projects stay small and medium-sized projects stay medium-sized. As shown in Equation 2.18, the modified effort is calculated by adjusting the base effort E using a percentage R. The term (100+R) indicates an increase or decrease in effort, based on the adjustment factor. If R is positive, the effort increases, while a negative R would reduce the effort required. The following equation is being used to generate the noise:

$$E_m \frac{E \times (100+R)}{100} \qquad (2.18)$$

Em represents the modified effort, while E stands for the original total effort for the project, and R denotes a random value selected from the range (-5, 5), excluding R=0. A similar equation was used for velocity, incorporating a random value up to a maximum of 5% of the original feature.

2.11 Related Work

In this section, effort estimation in agile software development is discussed, including traditional effort estimation techniques as well as various machine learning-based methods and algorithms identified through survey-based analysis.

2.11.1 Effort Estimation in Agile Software Development

A hybrid effort estimation approach for agile software development was presented to guarantee effective project results in terms of time, scope, and cost [52]. The Zia dataset [2], which consists of 21 projects from six software houses, was used by the model. For Agile projects, like Scrum, it used a story point strategy in conjunction with the k-Nearest Neighbors (KNN) machine learning technique to predict project completion time and overall cost. Metrics such as R-squared, Root Mean Square Error (RMSE), and Mean Magnitude Relative Error (MMRE) were used to assess the model's performance. The study came to the conclusion that better and more competitive estimations for agile projects are produced by the hybrid effort estimation model, which combines the story point technique with the KNN algorithm.

Estimating how much work is needed for software projects is a challenging task in project management, especially because software development is always changing. Over the past few decades, various techniques, such as algorithmic models, non-algorithmic models, and machine learning methods, have been developed to estimate software development costs. To enhance accuracy, machine learning approaches were combined with other models by Sharma & and Chaudhary [53]. They compared agile and traditional development using neural networks (NN) and genetic algorithms (GA). The estimation is carried out on the Zia dataset [2] using story points and a GitHub dataset using lines of code. They found that machine learning approaches, particularly neural networks and genetic algorithms, provided the smallest error and highest accuracy when predicting effort values. They made a comparison with previous works, based on mean magnitude relative error, and concluded that the dataset with story points yielded the best results, followed by projects with lines of code.

Many software businesses face challenges in managing and estimating agile projects, leading to high failure rates. Accurately estimating the effort and cost of a project is crucial for its success. The COCOMO (Constructive Cost Model) was introduced by Kumar & and Singh [54] in 2023 as a method for more precise cost estimation of software projects. COCOMO II, a specific version of this model, was highlighted for its benefits in easy data calculations. The Nave Bayes method machine learning technique was used for accurate predictions. The SEERA dataset was used to test the proposed system's performance. Overall, they conclude that COCOMO II, combined with machine learning methods, provides a reliable and accurate approach for estimating effort, and cost, and predicting the success or failure of software projects.

Artificial intelligence methods were employed by Bilgaiyan [55] to address the effort estimation problem (EEP) in Agile Software Development (ASD). They utilized a dataset comprising details on 21 ASD-based projects from six different software companies. This dataset is three-dimensional, specifically based on agile methodologies, including user stories, the project's initial pace, and the work required to complete the project within the designated sprint size. MATLAB was used to simulate the neural network models. The researchers used MATLAB to determine the estimated time and cost figures for the selected neural networks. They employed two forms of artificial neural networks: feedforward back-propagation and Elman neural networks.

Efficient approximations have always been difficult. A fuzzy logic model for effort estimation in Agile Software Development (ASD) was presented [56]. They monitored project metrics in datasheets from different sectors, including the total number of flaws found prior to project delivery, the project objectives, and the total number of defects found. Using fuzzy logic, the estimation of agile development effort was calculated in MATLAB. They used a fuzzy method to determine the project size rather than an exact figure. They used the Min Max Formula to calculate the output effort utilizing the fuzzy logic structure (Max-Min/Max-v). User stories, teamwork, and complexity are the model's three main estimating criteria. They came to the conclusion that user stories offer a rough approximation of developer feedback. While it may not be completely accurate, it can reduce the risk of uncertainty, which could otherwise waste significant time and resources. They conclude that user stories provide an estimated amount of developer feedback. Although it won't be completely correct, it can lessen the chance of doubt, which could waste a lot of time and resources.

Traditional methods of cost estimate don't produce the best outcomes for agile development. A cost estimation methodology was put up by Sharma & and Chaudhary [57]. A multiple regression model was suggested. The Zia dataset [2] was used. The 21 projects in the dataset come from 6 software houses. The time of the agile development is the dependent variable in the dataset, together with the independent variable's velocity, effort, friction product, initial velocity, and initial velocity. The decision trees, stochastic gradient boosting, and random forest models are examined and contrasted with the multiple regression models. They discovered a link between dependent and independent factors in their research. The effort has the strongest link with the development time, as seen by the correlation values. To determine the amount of effort needed for agile software development, three multiple linear regression models were created utilizing the dataset's variables. The three models consist of two linear models and one polynomial model. To compare the outcomes, the MMRE (Mean Magnitude Relative Error) was utilized. The error rate of the suggested work is lower (0.099%) than that of the earlier effort.

The common challenge in agile software development is accurately estimating the effort required for tasks. Unlike traditional (plan-driven) methods, agile lacks standard metrics for effort prediction. A Bayesian network model [59] was proposed specifically designed for agile methods. The model is simple, small, and requires easily gathered inputs, ensuring it doesn't

compromise agility. It can be applied early in the planning stage. The data was collected from the completed agile project of one company. In their study, the model predicts task effort. They use MMR and MMRE evaluation metrics for the prediction of effort. This model provides good accuracy on misclassified value Pred (m=25) within 25% tolerance. Additionally, they conclude that this model can be applied as early in the planning process as possible and has no practical influence on agility.

The story point approach (SPA), an empirical technique for assessing effort in agile software development, was first presented by Satapathy & Rath [18]. Their study used machine learning approaches like decision trees (DT), random forests (RF), and stochastic gradient boosting (SGB) to increase the prediction accuracy of effort estimate. To evaluate these methodologies' performance, a comparison was made with the current approaches. It was investigated whether the Scrum model's widely used SPA might be used to estimate work in agile software development. Using the Zia dataset as a focus, DT, SGB, and RF approaches were used to further refine the SPA results in order to increase accuracy [2]. The results showed that SGB performed better than the other machine learning methods that were examined. To expand on the approach, the paper also suggested investigating other machine learning methods on datasets relevant to SPA, such as Extreme Learning Machines and Bayesian Networks. Overall, the study recommended using machine learning to obtain improved prediction accuracy and emphasized the significance of precise effort estimation in agile software development.

The current corpus of literature on software development effort estimation has primarily concentrated on traditional projects; agile programs have received relatively less attention, especially when it comes to project-level effort estimation. Story points and team velocity are two important variables frequently taken into account when estimating work for agile projects. An improved software effort estimation model using support vector regression (SVR) optimized using the grid search method (GS) was presented [60]. Story points and velocity are inputs into the model. Using leave-one-out cross-validation, they applied their proposed model to 21 past agile software projects [2]. Based on Pred (0.25), MMRE, and MdMRE measures, the findings show that their strategy improves the performance of the SVR technique, surpassing a number of recent approaches published in the literature. Deep Belief Network (DBN) and Antlion Optimization (ALO) were used by [61].to forecast effort in software development scenarios that

are both agile and non-agile. When their suggested DBN-ALO method was used on datasets from the two development methodologies, it produced better outcomes for a number of evaluation parameters. In order to handle uncertainty, the study provides an effort prediction interval that allows project managers to estimate effort within a range as opposed to a single figure.

The DBN-ALO approach entailed establishing the DBN structure, normalizing the data, and optimizing weights using ALO. Statistical validation verified its effectiveness in both non-agile and agile development techniques. Using the DBN-ALO technique, this study offers a comprehensive analysis of effort estimation in both non-agile and agile software development. In comparison with other approaches, the suggested method performs better and adds an effort prediction interval to improve the flexibility of the estimate.

Eduardo Rodríguez Sánchez, Eduardo Filemón Vázquez Santacruz, and Humberto Cervantes Maceda developed a hybrid model to improve effort and cost estimation in Agile software development [90]. The proposed model integrates algorithmic regression techniques with ensemble methods, including decision trees, random forests, and AdaBoost, using labeled historical data and story points to predict project completion time (in days) and cost (in Pakistani Rupees). The researchers emphasized the importance of data discretization to enhance the accuracy of predictions. The study utilized a dataset comprising 21 Agile software development projects from six software houses in Pakistan. Key features of the dataset included story points, team velocity, sprint duration, and team salaries. To address the dataset's limited size, synthetic data augmentation techniques were applied, expanding the dataset to 42 entries for training. The model’s performance was evaluated using 10-fold cross-validation. The results demonstrated that the hybrid model achieved high prediction accuracy, with the bootstrap aggregation ensemble (bagging) approach outperforming individual techniques. Labeled data significantly improved the Mean Magnitude of Relative Error (MMRE) and prediction accuracy (Pred(x)) for both time and cost estimations. Among individual methods, the decision tree algorithm delivered the best performance, achieving the highest accuracy and R^2 values. In conclusion, the study highlights the value of combining machine learning ensembles and data discretization for improving effort and cost estimations in Agile projects. The proposed approach reduced prediction deviations and enhanced estimation reliability. Future research could explore

expanding datasets and employing advanced machine learning techniques, such as deep learning, to further refine the model's predictive capabilities.

The DBN-ALO approach entailed establishing the DBN structure, normalizing the data, and optimizing weights using ALO. Statistical validation verified its effectiveness in both non-agile and agile development techniques. Using the DBN-ALO technique, this study offers comprehensive analysis of effort estimation in both non-agile and agile software development. In comparison with other approaches, the suggested method performs better and adds an effort prediction interval to improve the flexibility of estimate.

2.11.2 Effort Estimation for Agile Software development using stacking ensemble Model

Onkar Malgonde and Kaushal Chari a model for agile software development that focuses on accurately predicting the effort required for tasks (stories) during the development process [58]. This approach addresses the critical challenge of effort estimation in agile projects, enabling better planning and execution. They tested seven different algorithms including support vector machines, ridge regression, artificial neural networks, K-nearest neighbors, decision trees, linear regression, and Bayesian networks, for predicting story effort but found that none consistently outperformed the others across their dataset of 423 stories. Despite their testing, neither approach consistently outperformed the others. They used an ensemble-based technique to forecast the work as a result, including data from 24 software development projects for their analysis. The performance of the model was evaluated using Mean Balanced Error (MBE), Mean Absolute Error (MAE), and Root Mean Square Error (RMSE) metrics, to ensure accurate and reliable assessment. Several limitations were identified, including the reliance on a homogeneous dataset from a single organization and the exclusion of factors such as scrum master performance due to the absence of relevant data. Additionally, the model faced challenges in predicting cross-project outcomes for initial sprints because of insufficient prior data. Despite these limitations, the ensemble-based technique outperformed other methods by combining the strengths of multiple models, leading to more accurate effort estimation.

Zaineb Sakhrawi and Asma Sellami developed the model to address the challenge of accurately estimating the effort required for Scrum projects focused on software enhancement tasks [92]. These projects' frequent demand changes frequently lead to planning challenges, overestimations, or underestimations, which can cause delays and inefficient use of resources. The accuracy required for trustworthy projections is frequently lacking in conventional techniques like Planning Poker or individual machine learning models. Three machine learning techniques Decision Tree Regression, Linear Support Vector Regression, and Random Forest Regression were combined to create a stacking ensemble model in order to address this issue. The COSMIC Functional Size Measurement (FSM) approach, which offers a consistent and trustworthy means of determining the functional size of software improvements, is a crucial input for this model. Dataset derived from real-world scrum projects. A 70-30 ratio was used for training and testing the model using actual Scrum project data. Accuracy was measured using Mean Absolute Error (MAE), Mean Square Error (MSE), and Root Mean Square Error (RMSE) evaluation metrics. The stacking ensemble model performed well, with an MAE of 0.206, an MSE of 0.406, and an RMSE of 0.595, compared to other machine learning methods. To further enhance the process's practicality, a web-based application named "ERWebApp" was developed. With the help of this application, which automates effort estimating, development teams, product owners, and Scrum masters can enter project details, determine functional sizes, and rapidly get precise effort estimates. The program improves project planning and saves time by streamlining the estimation process. In addition to increasing estimation accuracy, this method offers a simple way to manage effort estimating in dynamic Scrum environments, with potential for further development and wider use.

2.11.3 Effort estimation for traditional software development

In software development, effort estimation is critical since precise forecasts are necessary to ensure project success. Accurately estimating work has become more challenging since the introduction of agile software development. An innovative method to improve the accuracy of predicting agile software effort was presented by Khuat & Thi, My Hanh [62], who used an Artificial Neural Network (ANN) optimized with the Fireworks Algorithm (FWA). This approach was evaluated in comparison to other optimization algorithms in order to determine how well FWA optimized the weights and biases of the ANN. It was tested against

a variety of neural networks and a regression model. The study focuses on the story point approach for estimating agile software effort, using project velocity and total story points as inputs to train the artificial neural network. By tweaking the settings of the ANN, FWA and LM (perhaps referring to Levenberg-Marquardt) were able to achieve gains in accuracy. Experiments show that FWA, as opposed to algorithms like DABC, TLBO, and TLBABC, significantly improves ANN accuracy. Additionally, compared to other ANN types evaluated, such as General Regression, Probabilistic, GMDH Polynomial, and Cascade Correlation neural networks, the suggested ANN performs better. When compared to other neural network types and different methods, the solution that combines ANN and FWA with agile software effort prediction demonstrates improved accuracy overall.

In today's software development world, estimating effort accurately is critical to project success. Though numerous studies have suggested approaches to increase accuracy, accurate effort assessment is still difficult, especially for agile projects. A new method for predicting effort in agile software projects based on team velocity and story points was presented by T. T. Khuat & Le [63].

Swarm optimization approaches, such as a hybrid algorithm that combines the advantages of particle swarm optimization (PSO) and artificial bee colony (ABC), were used to optimize the formula's parameters. The outcomes of the experiments showed that the suggested strategies performed better in terms of prediction accuracy than methods from previous research. Swarm algorithms were used to optimize the new formula, which produced encouraging results. The hybrid ABC-PSO algorithm outperformed the individual ABC and PSO algorithms in terms of effectiveness across a range of evaluation criteria. Furthermore, the new hybrid method outperformed other kinds of artificial neural networks (ANNs) that were employed in previous research. A model was proposed [22] for software development effort estimation. In their study, they applied the algorithm to three datasets (ISBSG R8, Tukutuku, and COCOMO). In their study, they compared a random forest (RF) with a regression tree (RT). They used performance metrics Mean Relative Error (MRE), Mean Magnitude Relative Error (MMRE), and Median Magnitude of Relative Error (MdMRE) as evaluation criteria for effort prediction. Among the two techniques concluded that the RF model outperforms than RT model in terms of MMRE and it made slightly a higher MdMRE (0.67) than the RT model (0.52). They conclude that random forest is an effective method for estimating effort.

A software effort estimation approach was presented by Suresh Kumar and Behera [64] and was used with the Desharnais and COCOMO'81 datasets. A variety of machine-learning approaches were used in this model, such as the backpropagation algorithm on a feed-forward neural network, random forest (RF), neural network (NN), support vector machine (SVM), and k- nearest neighbors (KNN). They implemented the backpropagation algorithm on the feed-forward neural network using Python and the Orange data mining tool for data visualization and comparison of different approaches. By measuring the mean magnitude of the relative error, they evaluated how accurate their predictions were, and they came to the conclusion that backpropagation produced reasonably accurate forecasts when compared to other techniques. They added to their work [65] by putting out a different model that made use of the gradient-boosting regressor method.

Using gradient boosting regression, this technique first determines the difference between the current prediction and the known accurate target value. Next, it trains a weak model that maps features to that residual. Using the COCOMO'81 and CHINA datasets, they implemented this strategy with the goal of enhancing estimate accuracy through machine learning techniques, using gradient boosting regression, this technique first determines the difference between the current prediction and the known accurate target value. Next, it trains a weak model that maps features to that residual.

Using the COCOMO'81 and CHINA datasets, they implemented this strategy with the goal of enhancing estimate accuracy through machine learning techniques, using gradient boosting regression, this technique first determines the difference between the current prediction and the known accurate target value. Next, it trains a weak model that maps features to that residual. Using the COCOMO'81 and CHINA datasets, they implemented this strategy with the goal of enhancing estimate accuracy through machine learning techniques, (MSE), root mean square error (RMSE), and R-squared were used to assess the regression algorithms. Using the COCOMO'81 and CHINA datasets, the gradient boosting regressor model showed its efficacy with accuracy rates of 98% and 93%, respectively. When comparing the two datasets, the findings showed that this method performed noticeably better than any other regression models.

A model for estimating software effort with different machine learning techniques was presented [66]. The NASA dataset, which includes information on 60 programs, was used in the study. They used k-nearest neighbors (KNN), Elman Neural Networks (ENN), and Cascade Neural Networks (CNN) to predict effort. Three criteria were used to evaluate the models: Root Mean Square Error (RMSE), Balanced Relative Error (BRE), and Mean Magnitude of Relative Error (MMRE). The findings demonstrated that, in comparison to Elman Neural Networks and Cascade Neural Networks, the k-nearest neighbor approach generated a more accurate estimation, anticipating almost 85% of the real cost of the target.

In today's software engineering challenges, accurately estimating the cost remains a significant problem, especially during the maintenance phase of the software development life cycle. To address this, optimization algorithms, specifically metaheuristic approaches, are employed to reduce effort and errors. Singh [67] focused on enhancing the meeting rate of the Differential Evolution (DE) algorithm, a metaheuristic approach, to improve accuracy in the semidetached model. They used the NASA 63 dataset in the study. The proposed algorithm, named Enhance- Based Differential Evolution (EABMO), introduces new mutation strategies to optimize parameters more effectively. EABMO demonstrates superior results compared to semidetached model-based DE, Genetic Algorithm (GA), and Particle Swarm Optimization (PSO) algorithms. They used MRE, VAF, MMRE, MSE, Pred, and Convergence for performance evaluation. They concluded that the algorithm reduces the number of fitness evaluations and improves convergence rates through an enhanced adaption-based mutation operator. The proposed approach proves effective across various projects, with minimal drawbacks observed only in very high KLOC (kilo lines of code) scenarios.

One of the most difficult tasks for machine learning researchers is still estimating program effort. The Differential Evolution (DE) algorithm was used with the COCOMO II and COCOMO models [68] to estimate software effort. For their analysis, they used COCOMO'81 and NASA's 1993 records. The Mean Magnitude of Relative Error (MMRE) was assessed in the study as a benchmark. When compared to traditional algorithmic models such as COCOMO II and COCOMO, the DE method improved the accuracy of software effort estimation by efficiently optimizing parameter values.

Finding the most precise machine learning techniques for software development effort estimation was made easier in [69]. They contrasted the performance of ensemble techniques with individual methods and assessed the effectiveness of machine learning-based accuracy metrics. Using measures such as Mean Magnitude Error (MMRE) and PRED, their study evaluated the accuracy performance of 28 selected studies (14 ensembles and 14 individual approaches). The authors came to the conclusion that group approaches regularly performed better than individual ones. Ensemble strategies outperform other methods because they may accurately estimate effort by utilizing a well-balanced combination of rules and processes.

Estimating the cost of developing software is crucial in the software development process. People need to know how much time and resources will be required for a project. There are various methods to estimate costs, such as parametric and non-parametric approaches. Parametric methods are commonly used but may struggle with handling precise data accurately. To address this, different advanced techniques like machine learning and evolutionary algorithms have been applied to optimize the parameters of software cost estimation models. However, achieving accurate cost estimation remains a challenge. A model was presented [70] in which they optimized parameters in the popular Constructive Cost Model II (COCOMO-II) by using the Flower Pollination Algorithm (FPA). Their suggested model was tested against the Bat algorithm and COCOMO II utilizing data from the Turkish software sector. Two of the evaluation criteria were MD and MMRE. The study found that in measures like Manhattan distance and mean magnitude of relative errors, the Flower Pollination Algorithm fared better than previous methods like the Bat algorithm and the original COCOMO-II. This improvement is essential because precise cost estimation helps managers and engineers plan software projects more efficiently.

Software Cost Estimation SCE is critical in software development, and classic approaches such as the Constructive Cost Model (COCOMO) have been utilized since the 1980s. COCOMO is available at three levels: basic, intermediate, and detailed, with each level taking into account more project attributes. However, its precision is restricted due to issues such as cost driver loss. Khalifelu & and Gharehchopogh [71] proposed a model for software cost prediction by using different data mining techniques. Linear Regression (LR), Artificial Neural Networks (ANN), Support Vector Regression (SVR), and k-nearest Neighbors (K-NN) are among the data mining approaches examined in this study. The study trains and tests these

techniques on NASA project data to improve estimation accuracy. When COCOMO and data mining techniques are compared, the latter frequently produces more accurate results.

A hybrid model that combines the cuckoo search and harmony search algorithms was presented [72] with the goal of optimizing the four coefficients of COCOMO-II and improving the precision of effort and time development estimates for software projects. Using metrics like Magnitude of Relative Error (MRE) and Magnitude of Relative Error (MMRE), the suggested method was assessed using the NASA 93 dataset. According to experimental findings, the hybrid model performs better at estimating software project effort and time development than both COCOMO-II and the standalone cuckoo search method. In comparison to applying the COCOMO and cuckoo search algorithms independently, the suggested method specifically obtains a 54.04% improvement in effort estimation MMRE and a 0.68% improvement in time development MMRE. Within the software industry, meeting deadlines, budgetary constraints, and quality standards for project completion is a top priority. But current software projects are dynamic, which makes work and cost estimation difficult. In order to estimate software projects, Patra & Rajnish [73] presented an empirical interpolation model and compared it with the COCOMO model. Numerous cost considerations and COCOMO-based calculations were used in this comparison. Two empirically derived constants (a, b) based on historical data from NASA programs plus an independent variable (KLOC) make up the equation of the empirical interpolation model. Metrics including MMRE, RMSE, and PRED were used to evaluate the model's performance, and COCOMO was used to compare it with varying scale factor values. The findings show that compared to the COCOMO model, the interpolation approach offers more accurate effort estimates, fulfilling the demands of today's dynamic software industry.

It's important to estimate software development expenses, and a number of models, including fuzzy logic and neural networks, have been created to help with this task. A software development cost estimation model utilizing the Artificial Neural Network-Cuckoo Optimization Algorithm (ANN-COA) was presented [74]. This model uses the ISBSG dataset and combines the cuckoo optimization approach with neural network predictions. According to their research, the ANN-COA model outperforms other well-known methods for software cost prediction, making it one of the Neural Network category's best predictors.

For project planning to be effective, software development cost estimation is crucial. A model for software cost assessment utilizing a variety of machine learning algorithms was presented [75]. In their work, they used two publicly available datasets, Usp05-ft and Usp05, to test 13 machine learning methods. The outcomes showed how accurately machine learning techniques can forecast software expenses. The analysis showed that Kstar, REPTree, Additive Regression, and Random Forest performed best with the Usp05 dataset, whereas Random Forest produced the best results with the Usp05-ft dataset. On the other hand, with the first dataset, the ZeroR approach produced the worst results, while the Multilayer Perceptron, IBk, and Linear Regression methods did not fare well with the second dataset.

Agile software development is popular in industries, replacing traditional methods, but accurately estimating effort remains a challenge. The Story Point Approach (SPA), a mathematical method, is used to enhance effort prediction accuracy. Various neural networks (GRNN, PNN, GMDH, Cascade-Correlation) were employed in [61], considering story points and project velocity for initial effort estimation. The results are optimized using these networks, with the cascade network showing superior performance. The study, conducted in MATLAB, suggests extending the approach with additional machine learning techniques like Stochastic Gradient Boosting and Random Forest combined with SPA.The problem of work estimating in agile software projects was addressed by Kamal Tipu & Zia [2], who emphasized the importance of precise estimates for project success. Their study presents a model designed for agile approaches that bases estimation on User Stories. This approach is made to take into account the unique qualities of Agile, such adaption and iteration. Based on MMRE and PRED (n) metrics, it demonstrated good estimation accuracy when calibrated using empirical data from 21 software projects. Although the concept is workable and feasible, it still has certain shortcomings, and more improvements should improve its functionality.

2.11.4 Effort estimation for traditional software development using stacking ensemble model

G. Priya Varshini and K. Anitha Kumari developed a model to address the critical challenge of estimating effort in software development projects [89]. The research examined various machine learning and deep learning approaches, including individual models and

ensemble techniques that combine multiple methods for improved accuracy. A variety of ensemble techniques were examined, including averaging, weighted averaging, bagging, boosting, and stacking. The research focused on stacking models that combined support vector machines, random forests, decision trees, and generalized linear models. In this research for estimating effort, the researcher utilized datasets from Albrecht, China, Desharnais, Kemmerer, Kitchenham, Maxwell, and COCOMO'81. The comparison involved various machine learning models, including Random Forest, SVM, Decision Tree, Neural Network, and Deep Neural Network (DNN). Performance was assessed using metrics such as R-squared, RMSE, and MAE. The research showed that stacking, especially when used with random forests, produced better outcomes than single model approaches and other ensemble methods, proving to be more predictive.

Suyash Shukla and Sandeep Kumar explored innovative approaches to tackle the persistent challenge of accurately estimating effort in software development projects [93]. Traditional estimation techniques, such as expert judgment, Lines of Code (LOC), and Function Point (FP) methods, often struggle to meet the demands of modern software development environments due to issues like code reuse, evolving methodologies, and data heterogeneity. The study used stacking ensemble methods with models like SVM, MLP, and GLM to improve effort estimation in software development. The ISBSG dataset was divided into four subsets based on productivity levels to handle data differences and outliers. Metrics like MAE, RMSE, and MBRE showed that stacking models performed better than individual models and traditional methods like MLR and fuzzy logic, reducing errors by up to 40.79%. Despite its success, the research noted limitations, such as relying on one dataset and the need to explore new sizing measures, leaving room for further improvement.

G. Priya Varshini and K. Anitha Kumari introduced two advanced techniques for software effort estimation: a stacked ensemble model and a hybrid model combining Principal Component Regression (PCR) with Multivariate Adaptive Regression Splines (MARS) [94]. They utilized seven benchmark datasets, including COCOMO81, Desharnais, China, and others, to tackle issues like multicollinearity and non-linear relationships. The stacked ensemble model featured eight base learners, with PCR as the super learner, while the hybrid model combined PCR for dimensionality reduction and MARS for non-linear relationships. The models were evaluated using metrics such as MAE, RMSE, and R-Squared. The stacked

ensemble model achieved the highest prediction accuracy, and the hybrid model offered comparable results with reduced computational time. Despite its strengths, the study faced challenges like the complexity of implementation and computational costs. The study also highlighted that the findings were derived from benchmark datasets, emphasizing the need for further validation using real-world data. Additionally, it pointed out the model's reliance on dataset-specific tuning, which may impact its ability to generalize across different software projects.

P. Sampath Kumar and R. Venkatesan proposed a stacking ensemble model to improve software effort estimation accuracy [96]. Using the ISBSG dataset, the study addressed the limitations of traditional methods, including subjectivity and bias. The model combined three base learners linear Regression, Random Forest Regression, and Neural Networks with a Support Vector Regressor as the meta-learner. Comprehensive preprocessing reduced the dataset to 12 significant features, ensuring better correlation between variables. The model was evaluated using metrics such as MAE, MMRE, and PRED, achieving superior results (e.g., MAE = 0.11, MMRE = 0.10, PRED (0.25) = 0.925). While demonstrating high accuracy, the study acknowledged challenges like computational costs and dataset-specific tuning, emphasizing the need for further validation with real-world project data. This approach highlights the potential of ensemble learning in enhancing the reliability of software effort estimation.

2.12 Literature Review Analysis

Much work has been done in the field of Agile software development research to improve the precision of project effort and cost prediction. By putting forth a variety of models that frequently combine conventional approaches with machine learning strategies, like neural networks and ensemble methods, researchers are attempting to improve these estimations. Datasets like Zia, SEERA, and NASA are frequently utilized to validate and compare these models, enabling a full analysis and comparison of various methodologies. Many machine learning techniques, such a neural network, evolutionary algorithms, support vector regression (SVR), and ensemble techniques, are being investigated in an effort to improve accuracy by

learning from past data trends. In this field of study, comparison analyses of various algorithms, models, and methodologies are common.

In this research, comparisons of different algorithms, models, or techniques are common. Model performance is widely assessed using evaluation measures such as Mean Magnitude Relative Error (MMRE), Root Mean Square Error (RMSE), and R-squared. To ensure model resilience across diverse contexts, datasets from various sources, including industrial initiatives and publicly available datasets, are employed. Some studies highlight the shortcomings of classic models, such as COCOMO, in evaluating effort and cost in Agile contexts. As more effective alternatives, machine learning techniques, particularly ensemble methods, are proposed. Scholar aims to increase the precision of effort and cost estimation in Agile software development.

Researchers test novel combinations of classical and machine learning methodologies, using a variety of datasets and optimization algorithms. Despite advances, the dynamic nature of Agile projects continues to be a difficulty, emphasizing the need for an adaptive and Agile-friendly estimation model. Table 3.1 provides a summary of the literature review analysis, highlighting key studies, their methodologies, findings, and contributions relevant to the research topic. It offers a comparative view to identify gaps and trends in existing work.

Table 2.1: Literature review analysis

Sr #	**Ref & Year**	**Dataset**	**Technique/Method**	**Accuracy/Limitations**
1	[48] 2019	503 stories, 24 projects	Developed an ensemble-based model combining algorithms like Ridge Regression, ANN, SVM, Decision Trees, and KNN. Using feature selection to improve predictions. Blocked cross-validation was applied to keep the sequence of sprints intact. Stacking ensemble model was developed by combining three ML techniques Decision Tree Regression (DTRegr), Linear Support Vector	Data was from a single university, so the results may not generalize to other organizations. Didn't include factors like scrum master performance.

			Regression (Linear SVR), and Random Forest Regression (RFR). Functional size is measured using the COSMIC FSM method as the primary independent variable.	
2	[92] 2022	Private industrial dataset from scrum projects	Stacking ensemble model was developed by combining three ML techniques Decision Tree Regression (DTRegr), Linear Support Vector Regression (LinearSVR), and Random Forest Regression (RFR). Functional size is measured using the COSMIC FSM method as the primary independent variable.	Small size and private access restrict generalizability. It includes only numerical attributes, excluding potentially important categorical variables. Functional size measurement relies solely on COSMIC FSM, potentially overlooking other factors.
3	[54] 2023	SEERA dataset	COCOMO II model combined with Nave Bayes algorithm to estimate effort. Stacking ensemble model was developed by combining three ML techniques Decision Tree Regression (DTRegr), Linear Support Vector Regression (LinearSVR), and Random Forest Regression (RFR). Functional size is measured using the COSMIC FSM method as the primary independent variable.	Model is limited to the scope of the dataset, which focuses on Sudan-based projects. Model assumes specific conditions and may not fit agile workflows perfectly.
4	[55] 2023	Dataset from 21 agile software projects across 6 software houses	The study applied artificial neural networks (ANNs) to predict effort estimation in agile software development. Data was normalized, split into training and validation sets, and processed using MATLAB's Neural Network Toolbox.	Proposed models may not perform as effectively on heterogeneous datasets from different software development methods.
5	[52] 2022	Dataset of 21 projects from 6 software houses	A hybrid model was used that Combined linear regression and k-Nearest Neighbors (KNN) to estimate project effort, time, and cost based on analogy.	The dataset size was small and the proposed approach may not perform well for datasets with high-dimensional data or large variations in features
6	[61] 2019	Agile and non-agile datasets from PROMISE repository	Deep Belief Network (DBN) with Antlion Optimization (ALO) were used	Complexity of combining DBN and ALO; performance depends on hyperparameter tuning.
7	[60]	21 agile	Support Vector Regression (SVR)	Limited dataset (only 21 projects), assumes normal

	2018	software projects	optimized with grid search method (GS). Leave-One-Out Cross-Validation (LOOCV) was applied to validate the model.	distribution, computational overhead of Grid Search for large datasets. Model performance was improved in terms of Pred (0.25), MMRE, and MdMRE.
8	[63] 2018	Story points and team velocity data from agile projects	Particle Swarm Optimization (PSO), Artificial Bee Colony (ABC), ABC-PSO were used to estimate the effort.	Requires historical data; hybrid methods may be computationally intensive
9	[18] 2017	COCOMO'81 and Desharnais datasets	Decision Tree (DT), Stochastic Gradient Boosting (SGB), Random Forest (RF) were used.	Results are dataset-dependent, potential overfitting with neural networks, limited to specific datasets

2.14 Summary

This chapter provides a comprehensive overview of various effort estimation methods, including algorithmic models, learning-based techniques, and expert opinion-based approaches. Special attention is given to effort estimation within agile software development, highlighting key terminologies and concepts essential to understanding estimation in agile contexts. The chapter also explores the application of machine learning techniques, such as ensemble learning and data augmentation, emphasizing their role in enhancing estimation accuracy. Despite the advancements in estimation methodologies, the inherently dynamic and iterative nature of agile projects continues to pose challenges, underscoring the need for adaptive, agile-friendly estimation models.

A detailed review of the literature is presented, focusing on effort and cost estimation in agile software development. Numerous researchers have proposed a variety of models and methodologies, many of which leverage advanced machine learning approaches. Studies involving experiments with deep learning and machine learning techniques are examined, providing insight into current trends and outcomes in this area. Both traditional and agile software development estimation models have been the subject of extensive academic investigation. This ongoing need drives researchers to explore hybrid models that combine

traditional and modern techniques. Incorporating domain-specific features has also shown promise in improving prediction performance. Additionally, cross-project learning is emerging as a valuable approach for enhancing generalization in estimation models. However, despite the breadth of existing research, there remains a significant need for further enhancement and refinement of machine learning models to improve the accuracy and adaptability of effort estimation practices.

CHAPTER 3

RESEARCH METHODOLOGY AND PROPOSED FRAMEWORK

3.1 Overview

This chapter's main goal is to outline the research techniques or protocols applied in this work, with a particular emphasis on examining effort estimate in the field of artificial intelligence. The proposed approach for conducting this research will be discussed, along with detailing the experimental stage and the practical performance throughout the study. Additionally, the evaluation metrics used to assess the results will be addressed.

Precisely anticipating the development cost, time, and effort constitutes the most challenging and crucial task in software development. Without such anticipation, wise management decisions that could prevent total failure would be unattainable for project managers, system analysts, and developers. It is widely believed that significant overruns occur only due to faulty estimation.

3.2 Research Methodology for Effort Estimation

Any process, collection of guidelines, or set of ideas that provide guiding principles for comprehending and utilizing particular techniques or procedures to address issues in a particular field is referred to as a methodology. The first stage in this study is to precisely define the research problem [77]. The procedures listed below, which are expanded upon in the following sections, are how the research addressed and resolved this discovered problem after that.

As shown in Figure 3.1, the overall research process is organized into four sequential phases: Problem Formulation, Dataset Preparation, Experiment, and Results. Each phase builds upon the previous one, starting from defining the research problem to collecting and validating data, implementing the experiment, and ultimately deriving the results.

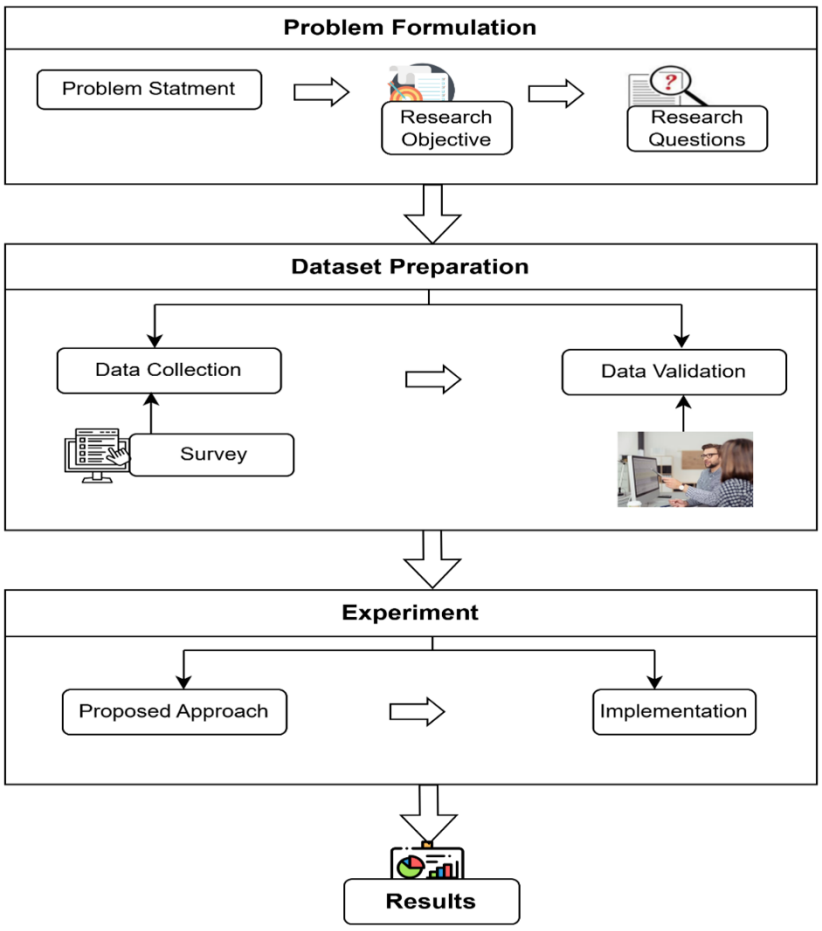

Figure 3.1: Research Methodology for Effort Estimation

Problem Formulation: The chosen subject of study is effort estimation for agile software development, with a focus on the importance of precise estimations in improving the chance of delivering high-quality projects on schedule. Several research publications have been regularly evaluated after the subject domain was chosen to find research gaps. Various research approaches and methodologies were employed in the literature papers, with the primary focus being on effort estimation for agile software development. After studying the research papers,

it is noted that there exists a research gap in the effort estimation for agile software development using artificial intelligence, especially ensemble learning.

Dataset Preparation: After problem identification, the next step involves dataset preparation. In Section 3.5.1, this step is further elaborated upon by the outlined steps for dataset collection. Then collected data has been validated through experts.

Experiment: In the next step, after data collection, the research progresses towards experimenting to address the mentioned research problem. An ensemble learning technique, stacking, has been proposed to address the research gap. After the experiment was performed, the results were calculated using evaluation metrics.

Results: In the last step, after completing the experiment, all the results were evaluated according to the evaluation metrics.

3.3 A Proposed Approach: Stacking Model

In the realm of software development, precisely estimating effort for Agile projects remains a significant challenge. Unlike traditional models, Agile development's iterative and adaptive nature introduces unique dynamics that can affect estimation accuracy. Despite various techniques such as algorithmic, AI-based, and expert opinion approaches being applied to effort estimation in traditional models, their effectiveness in the Agile context is not well-explored. To address this gap, an ensemble-based technique which is the stacking model, is proposed to address this gap, as shown in Figure 3.2.

Ensemble learning is like a collaborative approach, it uses a combination of techniques known as base learners or inducers to make decisions rather than depending on just one as explain in chapter 2 section 5. Ensemble learning comprises three techniques: bagging, boosting, and stacking. Bagging trains multiple instances of the same model on different data subsets and averages their predictions. Boosting sequentially trains weak learners, focusing more on misclassified instances. Stacking combines model predictions by training a meta-

model on their outputs, weighing predictions based on performance. Ensemble methods are currently being successfully applied to solve pattern classification and regression problems. In agile software development effort estimation, stacking is more effective than bagging and boosting because it combines multiple diverse models, capturing complex relationships in the data.

Stacking uses a meta-model to integrate predictions from various base models, providing more accurate and robust results. Stacking is a powerful ensemble learning technique that combines predictions from multiple different models to make the final prediction. Unlike bagging, which averages predictions from similar models to reduce variance, or boosting, which sequentially focuses on correcting errors to reduce bias, stacking uses diverse models to capture their individual strengths. The meta-model learns how to best combine these predictions, making stacking highly flexible and capable of delivering better performance, especially for complex problems with large datasets [95]. However, it requires careful tuning and is computationally more intensive, but its ability to blend different algorithms often makes it the go-to choice for advanced predictive modeling. Overall, stacking's flexibility and ability to reduce bias and variance make it a better choice for effort estimation in dynamic environments [47][49].

In the context of agile development, the stacking model when combined with a meta-model shows to be an effective and flexible approach that offers an in-depth answer to the issues related to effort estimate. The primary goal of the proposed approach is to boost accuracy in terms of cost and time in Agile projects. This helps in capturing diverse patterns in the data that a single model might miss. Some models may perform better with categorical data, while others excel with numerical data. Stacking ensures that the advantages of both are utilized. In the proposed approach, an ensemble is constructed by employing Support Vector Regression (SVR), K-Nearest Neighbors (KNN) Regressor, and Decision Tree as the base.

These machine-learning models were employed based on their demonstrated effectiveness in achieving favorable outcomes, as indicated by the literature review. Different combinations were utilized, in the placement of the meta-model, to explore various arrangements. The results are observed as good when the combination involves linear regression as a model. Each of these models brings unique strengths to the ensemble, capturing

different aspects of the underlying data patterns. For example, SVR optimizes the margin between the data points and the hyperplane in addition to fitting the model as closely to the data as feasible,[40] KNN is good in capturing neighborhood patterns and dependencies, Decision trees, on the other hand, are useful for locating important splits in the data and for partitioning the feature space [78]. The meta-model, in this case, is Linear Using this combination, we test the model and using evaluation metrics we test the accuracy of the model. Ensemble methods have been successfully applied for solving pattern classification and regression tasks. We used stacking technique in the work.

Stacking enhances overall model accuracy by mixing predictions from different base models. The ensemble model can benefit from the individual models' capabilities by stacking them, since each base model may be particularly good at capturing certain features or patterns in the data. Regression, which combines the predictions of the base models to improve performance. This approach not only improves the performance of the model but also reduces the possibility of overfitting, which is a serious problem, particularly when working with noisy or small datasets. To increase the size of dataset we also used data augmentation technique. Before data augmentation, small datasets were typically handled using techniques like cross-validation and simple data splitting. While these methods helped evaluate model performance, they didn’t address the core issue of limited data. Cross-validation used the same data for training and testing but these often lacked the diversity needed for more accurate predictions [51].

Data augmentation is the process of creating modified versions of preexisting data samples in order to artificially enlarge a dataset. Its goal is to increase the dataset's unpredictability and diversity in order to possibly improve machine learning models' performance and ability to generalize. This method works especially well for tasks like object detection, natural language processing, and image categorization. In the context of agile development, the stacking model when combined with a meta- model shows to be an effective and flexible approach that offers an in-depth answer to the issues related to effort estimate. The primary goal of the proposed approach is to enhance accuracy in terms of cost and time in agile projects. As shown in Figure 3.2, the proposed approach begins by dividing the dataset into training data (80%) and testing data (20%). Multiple machine learning models including SVR, Decision Tree, KNN Regressor, and Linear Regression—are trained using the training set. The

models are then tested, and their performance is evaluated using specific evaluation metrics to determine accuracy and effectiveness.

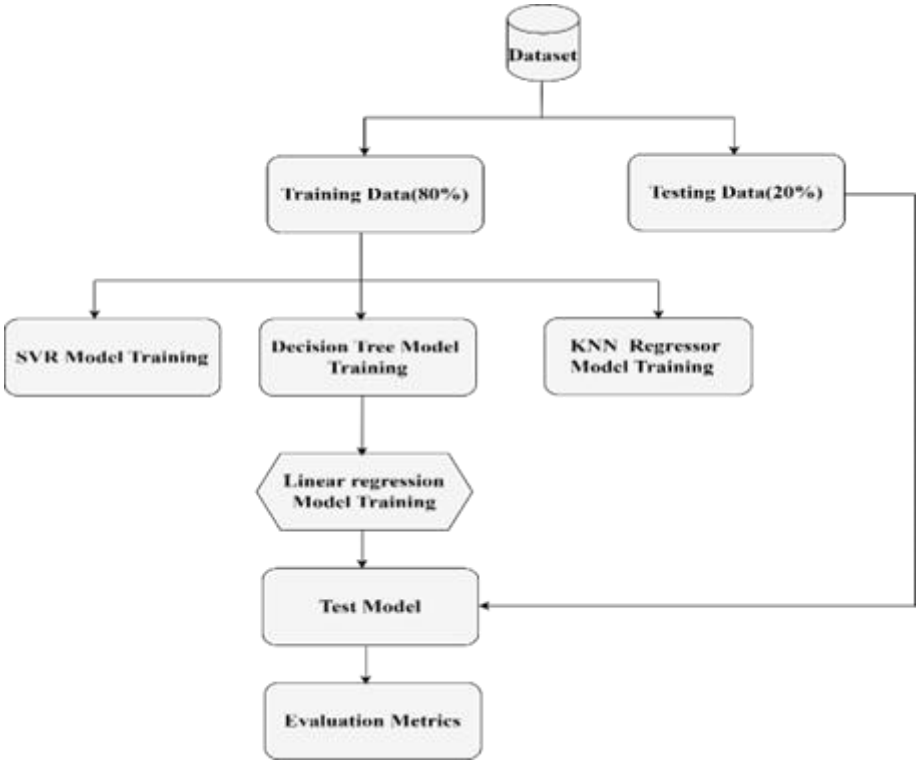

Figure 3.2: Proposed Approach for Effort Estimation in Agile Software Development

3.4 Data Collection

The current dataset was inadequate and out of date, due to this reason, new data is being collected. A thorough questionnaire has been created to collect the dataset needed for agile project effort estimation. The questions were carefully chosen to cover all the important features that are important for effort estimation for agile software development, such as the sprint size, team salary, work days, actual time, actual cost user stories, etc. To create a meaningful dataset a Zia's dataset [2][90] is used as a reference. To collect dataset information, we follow several steps that are described as follows.

3.4.1 Target Software House

The utilization of LinkedIn helped us in the identification of companies. The platform aids in identifying potential participants who offer valuable insights into agile software development in Pakistan. By perusing LinkedIn, profiles and details of various software houses can be easily accessible, simplifying the process of selecting suitable ones for the study. This step is crucial as it facilitates connections with companies actively engaged in software development.

3.4.2 Identify Software metrics

Discuss data link layer issues found in literature. In the data collection step for effort estimation in agile software development, software metrics play a pivotal role in providing valuable insights into various aspects of the development process. One crucial set of metrics includes those related to user story size, sprint size, actual time, and actual cost etc. User story size metrics offer a granular understanding of the scope and complexity of individual features or functionalities within the software. By quantifying user story size through techniques like story points or relative sizing, teams can effectively gauge the effort required for implementation, allowing for more accurate resource allocation and sprint planning. Sprint size metrics capture the volume of work undertaken within each iteration of development, providing a measure of team velocity and progress. Then a proactive approach is taken, involving on-site visits to various software houses.

During these visits, the questionnaire was handed to them, accompanied by a document that explained each attribute. The importance of this face-to-face interaction lies in ensuring that everyone comprehends the inquiries. All efforts are made to guarantee the acquisition of necessary information, with personal explanations and responses provided during the visits. This hands-on approach was about making sure we had a clear and complete set of responses for our study on agile software development in Pakistan.

3.4.3 Response Validation

Data validation is a procedure used to ensure that the collected information is truthful and reliable [79]. Preventing the processing or storing of unsuitable or inaccurate data is its main goal. This includes checking the data for mistakes, validating it against predetermined formats or restrictions, and confirming its dependability in a particular setting. Initially, 49 responses were obtained from the queried software companies. The results were validated by two experts in the field: Muhammad Sheheryar Nasir Khan, a Scrum Master with 4 years of experience at Join App Studio, and Muhammad Qasim, a Product Manager at Garana.com with over 5 years of experience. Both experts thoroughly reviewed the dataset. Their expertise helped validate the data, ensuring that the 47 responses deemed valid were trustworthy for further analysis. Following a thorough verification process to ensure data is correct, it is ascertained that 47 of those responses have been deemed good and accurate.

3.4.4 Data Analysis

The process of modifying and analyzing data to glean insightful information that supports well-informed decision-making is known as data analysis. [80]. It's like trying to find the most important pieces of information that will help us learn and understand results. In this context, the mathematical aspect is deemed essential, as it contributes to enhancing the dataset's quality. By using formulas, we made sure that` all the information followed the same rules and was accurate. Several formulas, which are outlined in Chapter 2, (Equations 2.11, 2.12, 2.13, 2.14, 2.15, 2.16, 2.17), are used in the computation of datasets. After getting responses from 47 projects, we did some math and looked closely at the important details. By doing this, we found out the main things that matter for a computer program to learn from the data.

3.5 Experiment

In this section, the entire process of experimentation as depicted in Figure 4.8, will be discussed. Initially, the dataset was collected because the size of the old dataset was small. Then

the data preprocessing has been performed to identify labels and features. After data preprocessing the data has been moved forward for the split data into training and testing sets. Ensemble learning techniques are then set up, and the subsequent step involves model prediction. Finally, the results have been assessed using evaluation metrics. Figure 3.3 illustrates the sequential steps followed in the effort estimation experiment.

Figure 3.3: Experiment steps for effort estimation

3.5.1 Dataset Preparation

In order to build a structured dataset, fit for analysis or machine learning model training, raw data must be gathered, cleaned, formatted, and arranged. The effort estimator for agile software development is tested, trained, and assessed using a variety of datasets obtained from different software companies. These datasets have been carefully selected to be used in the training and evaluation of machine learning algorithms, with an emphasis on ensemble learning methods, for the purpose of estimating agile software effort.

To conduct the entire experiment, firstly, the dataset is collected, and subsequently, dataset is validated. The dataset shows details of 47 software development projects, including effort spent, team size, actual time taken, and estimated costs. Each project has multiple data points for different aspects like effort for different tasks (Efforts), estimated and actual time for each sprint (Sprint Size, Work days), and team salary and costs (Team Salary, Actual Time, Estimated Time, Actual Amount, Estimated Amount). Lowercasing, stop word removal, stemming and lemmatization, data cleaning, and data filtering are examples of data preparation procedures. Data cleaning and filtering were performed to ensure the dataset's quality and relevance.

One important stage in data preparation is identifying and removing undesirable or irrelevant data points from the dataset. This is referred to as data cleansing or data filtering. Outliers, duplicates, and instances that do not add meaning to the analysis or model training are examples of unwanted data.

3.5.2 Feature Selection

A crucial stage in machine learning and data analysis is feature selection, which focuses on locating and selecting the most relevant features (or variables) from a dataset. The objective of this procedure is to optimize computational complexity or improve model performance. Feature selection plays a crucial role in the field of effort estimation for agile software development by identifying the significant elements that influence the effort required to complete tasks or projects related to software development. To refine and optimize our dataset for subsequent analyses, we directed a thorough correlation analysis to explore the relationships among the various features. The primary objective was to identify and exclude redundant or highly correlated features that could potentially introduce multicollinearity issues and hinder the accuracy of our predictive models. This step is crucial in enhancing the efficiency and interpretability of our dataset, ensuring that only relevant and non-redundant variables contribute to the subsequent analyses.

The correlation analysis was employed to evaluate the strength and way of relationships between pairs of features, allowing us to make informed decisions about feature selection and streamline the dataset for further investigation. So far, we have conducted a comprehensive correlation analysis on our dataset to gain insights into the interdependencies among different features. "Efforts" and "Actual Time" have a very strong positive relationship. Likewise, there is a significant positive association between "Actual Amount" and "Actual Time”. This means that when the actual time spent on a project increases, the actual amount of work completed also tends to increase. Increased velocity might result in completing more work within the same timeframe, resulting in a positive correlation with the actual amount of work done and similarly for effort. We ensure that future research or machine learning models will be based on a more accurate and representative dataset by carefully removing extraneous items from the dataset.

Improving the reliability and robustness of data is a critical function of data cleaning, which increases the overall efficacy of modeling or analytical operations.

Data Discretization: Since labeling lays the groundwork for project analysis, it is crucial. For instance, the model can forecast the estimated time and cost for small, medium, and big projects by classifying them according to size. Managers can investigate different scenario is for every project thanks to this capacity. After the labels were first assigned (small = S, medium = M, and big = L), the dataset was encoded into integers so that machine learning algorithms could be trained on it. To put it simply, labeling data improves the model's capacity to offer more in-depth understanding of possible variances in project costs and schedules. The convention used is:

0=Large =L, 1=Medium=M, 2=Small=S

3.6 Correlation Analysis

In machine learning, correlation analysis is a method used for feature selection. It comprises examining how the target variable (also known as the dependent variable or outcome) and input features (also known as independent variables or predictors) relate to one another. The goal of correlation analysis in this context is to identify which features have the strongest relationship with the target variable [81]. The full dataset is tested, but the findings are not sufficient, thus correlation analysis is used to determine the direction and strength of correlations between feature pairs. This analytical method simplifies the dataset for more research and helps to make well-informed feature selection judgments. A statistical technique called correlation analysis is used to determine whether or not there are relationships between variables or datasets. Its main goal is to determine whether relationships exist and to measure how strong those relationships are.

Features like effort, initial velocity, deceleration, ultimate velocity, actual cost, actual time, and expected time are all included in the dataset that is being examined. Notably, effort, initial velocity, actual time, and actual cost are found to be strongly correlated. The accompanying picture provides a visual representation of the correlation analysis results. Strong

connections between effort, initial velocity, actual time, and actual cost can be expressed in passive form. The feature effort has a strong correlation with Actual time spent also tends to increase. Effort has negative correlation with Sprint Size (-0.52), meaning that projects with higher effort tend to have smaller sprint sizes. There is no significant correlation with other features.

Initial velocity has a moderate positive correlation with work days (0.40), suggesting that projects with higher initial velocity tend to have more work days. Initial velocity has negative correlation with Sprint-Size (-0.47) and Actual-Amount (-0.80). There is no significant correlation with other features. Sprint-Size has a moderate negative correlation with Efforts (-0.52), Initial-Velocity (-0.47), and Actual-Time (-0.40). There is no significant correlation with other features.

Sprint size has a negative correlation with V (-0.40). There is no significant correlation with other features. Actual-Amount has a positive correlation with Team-Salary (0.62) and negative correlation with Initial-Velocity (-0.80). There is no significant correlation with other features. Figure 3.4 presents the correlation heatmap, illustrating the relationships between different variables in the dataset. Darker colors indicate stronger positive or negative correlations.

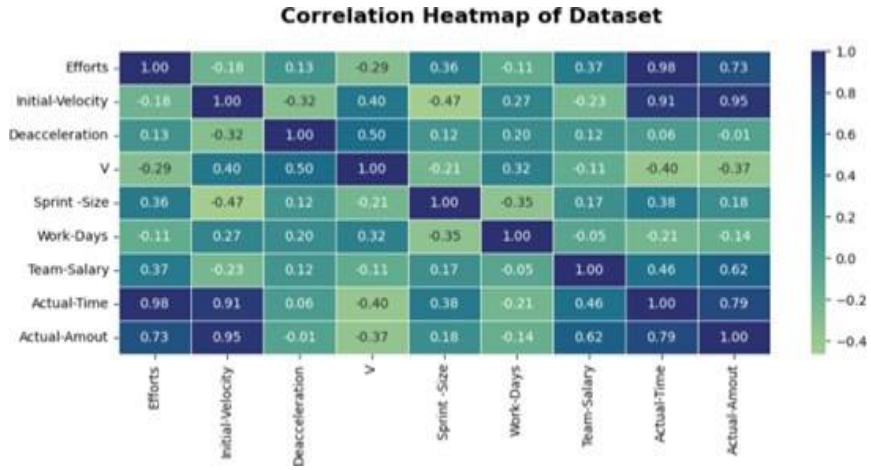

Figure 3.4: Correlation Analysis of features

3.7 Evaluation Metrics

Mean Magnitude Error (MME), Mean Absolute Error (MAE), and Mean Relative Error (MRE) are commonly used metrics in the context of evaluating the performance of models, especially in regression tasks [82]. In Ordinary Least Squares (OLS), model fitness in regression-based models is assessed using the sum of squared errors. For regression models, two relevant function errors are the Mean Squared Error (MSE) and RMSE. The R2 score are commonly found in literature reviews as benchmarks for evaluating model performance.

Mean Absolute Error (MAE) is a commonly used metric to measure the accuracy of a predictive model. It calculates the average of the absolute differences between the actual values and the predicted values. In simple terms, it tells you how far the predictions are from the true outcomes, on average [60].

R^2 Score measures how well the predicted values from a model approximate the actual data points. It indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. Equation 4.1 shows that the coefficient of determination is equal to one minus the ratio of the model's residual sum of squares to the total sum of squares.

$$R^2 = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{\sum_{i=1}^{n}(y_i - \hat{y}_i)2} \qquad (4.1)$$

Root Mean Square Error, or RMSE for short, is a commonly used regression analysis metric to evaluate the precision of prediction models. It measures the typical size of errors in a dataset between expected and actual values. The square root of the average of the squared discrepancies between the expected and actual values is used to compute the root mean square error, or RMSE. With lower values indicating greater model performance, RMSE offers a clear indication of how well a model's predictions match the actual data.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \qquad (4.2)$$

The equation 4.3 equation defines the Mean Absolute Error (MAE), which measures the average magnitude of the errors between predicted and actual values, without considering their direction. It provides an overall indication of how close predictions are to the actual outcomes.

The formula for MAE is:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i | \quad (4.3)$$

y is real data and $\hat{y}$ is the prediction.

3.8 Summary

This chapter describes the suggested strategy for addressing the research problem and achieving the goals while also examining the research methodology. It also describes the experimental procedures used in this investigation. Furthermore, this chapter provides a detailed explanation of the methodological framework, including the rationale for choosing specific techniques and tools. The experimental procedures are also elaborated upon, offering insight into how data was collected, processed, and analyzed to derive meaningful results. Through this comprehensive overview, the chapter ensures a clear understanding of the overall research design and execution.

CHAPTER 4

RESULTS

4.1 Overview

This chapter presents the results derived from an experiment involving a model trained using ensemble learning techniques, specifically stacking, applied to a dataset sourced from the software industry. The outcomes are assessed using standard evaluation metrics commonly used in literature for effort estimation in agile software development, namely R2-Score, mean absolute error, and mean relative error. The research primarily aims to enhance the accuracy of effort estimation in agile software development.

4.2 Results for effort estimation in agile software development

An ensemble-based effort estimate model is proposed that incorporates the ideas of ensemble learning and allows for the simultaneous use of various methodologies. As described in detail in Chapter 2, Section 4, the ensemble technique known as stacking is chosen for implementation. Correlation analysis is conducted on the dataset, focusing on the features selection for effort estimation as described in Chapter 4. After selecting features, we utilized data augmentation techniques to expand the dataset size because it was initially small. This step was essential for improving the reliability of our model and ensuring more precise estimations. This section describes the results of our model with correlation analysis, without correlation analysis results, results with data augmentation and results without data augmentation. We used a dataset that we composed from the software industry. The features of the data set are effort, initial velocity, deacceleration, final velocity, actual cost and actual time.

These features are input of the model and predicted outputs are predicted time and predicated cost. As already discussed, data discretization of features in Chapter 4 section 4.3.2. Similarly, here data discretization for class labels is performed. In the classification of classes, they are categorized into small, medium, and large. Small classes are referred to as being labeled small, medium classes are characterized as being classified as medium-sized, and large classes are described as being designated as large. This method ensures that classes are accurately categorized based on their respective sizes. The agile project is classified into three groups small, large and medium and we have effort size, cost size and time size group as discussed in chapter 4 section 3.5.2. Table 4.1 presents the classification of projects according to labels, showing the distribution of time, cost, and effort for small, medium, and large projects.

Table 4.1: Classification of projects according to labels

Label	**Small**	**Medium**	**Large**
Time	37	5	5
Cost	5	35	7
Effort	25	12	10

The Figure 4.1 shows the classification of projects according to effort, cost and time in different color combinations.

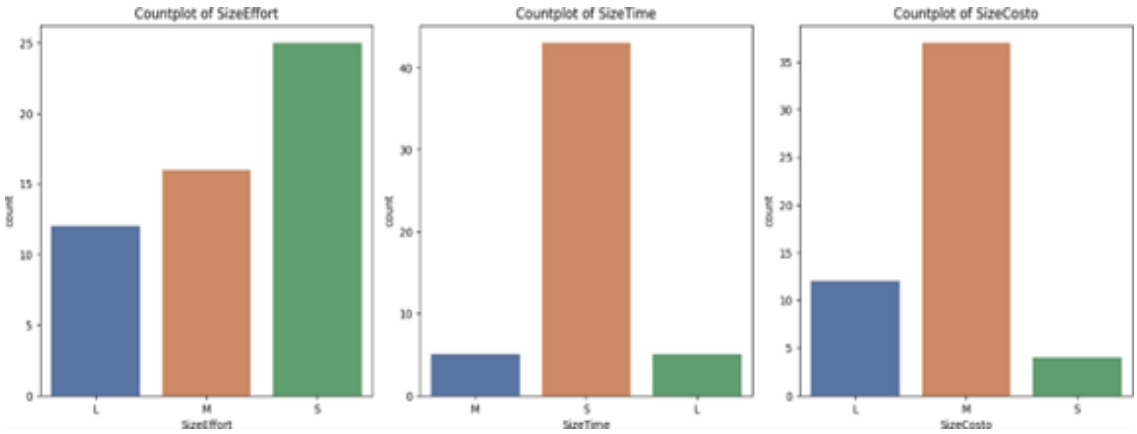

Figure 4.1: Classification of projects according to effort, cost and time

4.2.1 Results of ensemble model without data augmentation technique

Table 4.2 below presents the results generated from ensemble model. We utilized three evaluation metrics: R2-Score, MAE, and RMSE, as described in Chapter 3, Section 3.7. The overall performance of our model demonstrates improvement, particularly in terms of the R2-score evaluation metric.

Table 4.2: Result of Ensemble Model Without Data Augmentation Techniques

Parameter	R^2-Score	MAE	RMSE
Time	90	0.341	2.56
Cost	97	0.38	0.180

As illustrated in the table, our model achieves 90% accuracy in terms of time estimation and 97% accuracy in terms of cost estimation. The table provides a comprehensive breakdown of different evaluation metric values for effort estimation in agile software development using the ensemble-based stacking model for both cost and time. This outcome underscores the effectiveness of our model in accurately estimating effort in agile software development projects, enhancing both time and cost estimation accuracy.

4.2.2 Results of ensemble model with data augmentation technique

Data augmentation is a machine learning and data science technique that involves making slightly altered copies of preexisting data points in order to artificially enhance the size of a dataset. As mentioned in Chapter 2, Section 5, the goal of this method is to improve the robustness and performance of machine learning models, especially in situations when the amount or diversity of the original dataset may be restricted. Following the expansion of the dataset, there were 77 total projects instead of just 47. There was a total of 77 submissions for

the project, of which 54 synthetic projects (about 70%) were used for training and 23 projects (30%) had their genuine values set aside for testing. Table 4.3 shows the results of the stacking model with the data augmentation technique, presenting the R^2 score, MAE, and RMSE values for both time and cost parameters.

Table 4.3: Result of Stacking Model with Data Augmentation Technique

Parameter	R^2-Score	MAE	RMSE
Time	91.51	0.312	0.020
Cost	98	0.35	0.170

The table demonstrates that our model attains a 90% accuracy rate for time estimation and a 97% accuracy rate for cost estimation. The result emphasizes the efficacy of our model in precisely predicting effort in agile software development projects, thereby improving accuracy in both time and cost estimation. Figure 4.2 illustrates the predicted cost, where the x-axis epitomizes the predicted cost and the y-axis represents the actual cost. The diagonal line represents a perfect fit, where the predicted cost aligns precisely with the actual cost. Throughout the graph, scattered data points are observed, with potential clusters towards the bottom left and top right corners.

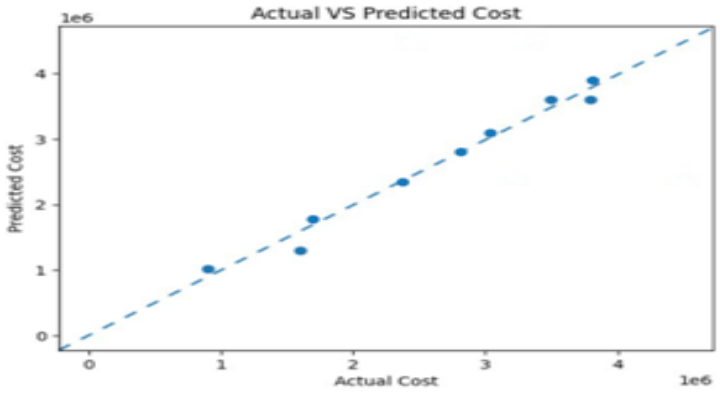

Figure 4.2: Ensemble model prediction for cost

Towards the top of the figure 4.2 as shown in below, a notable instance occurs where the predicted value is approximately 0.9 million PKR, while the actual value is 0.8 million PKR. Conversely, towards the bottom of the graph, there is a data point where the actual cost is 3.7 million PKR, closely matched by the predicted cost of 3.9 million PKR.

These observations provide insights into the performance of our model, indicating areas where the predictions closely align with the actual values and areas where there may be discrepancies.

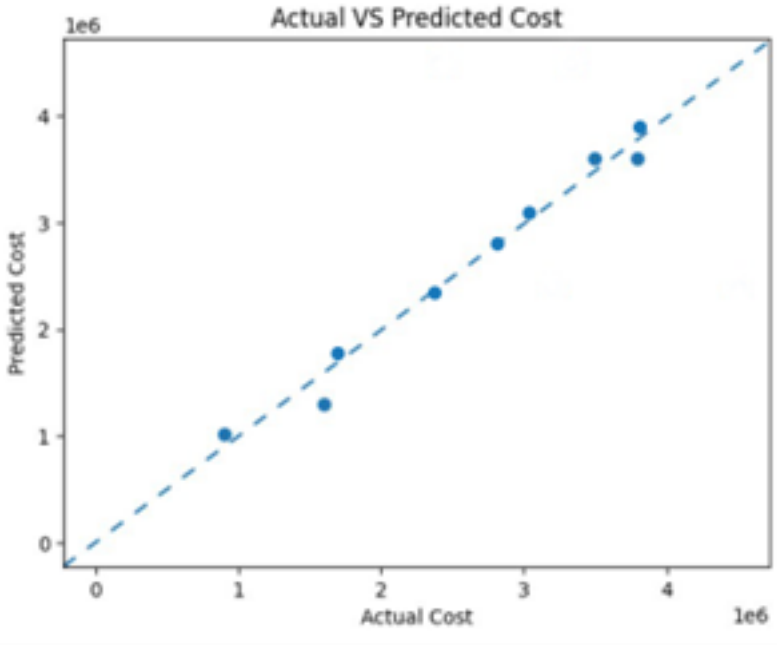

Figure 4.2: Ensemble model prediction for cost

Figure 4.3 illustrates the comparison between predicted and actual time using our stacking model. Each point on the graph represents a specific data instance, with the x-axis indicating the predicted time and the y-axis showing the actual time. At the top of the graph, there is a data point where both the actual time and the predicted time are 80 days, indicating a close alignment between our model's prediction and the ground truth. Conversely, towards the bottom of the graph, there is another data point where the actual time is 750 days, while our stacking model predicts 690 days.

These observations provide a detailed view of the performance of our stacking model in estimating time for agile software development projects. While some predictions closely match the actual time, there are instances where there may be deviations, highlighting areas for further refinement and improvement in our model.

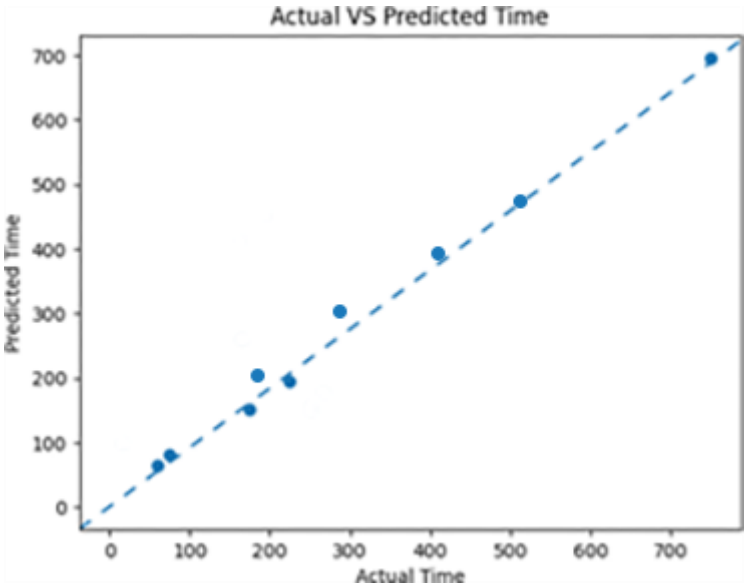

Figure 4.3: Ensemble model prediction for time

When it comes to completion time prediction, the stacking model performs better than the others; comparable outcomes are seen when estimating total cost. In machine learning and data science, data augmentation is the process of creating altered versions of preexisting data points in order to artificially extend the dataset. This method seeks to improve machine learning models' robustness and performance, particularly when the size or diversity of the original dataset is limited.

4.2.3 Results of regression model

The regression model was applied to estimate effort in Agile software development using a dataset expanded through data augmentation techniques. Data augmentation was employed to increase the diversity and volume of the original dataset, thereby enhancing the

model's learning capability and generalization. The augmented dataset consisted of 77 Agile software projects, where 54 synthetic entries (approximately 70%) were used for training and 23 real project entries (30%) were reserved for testing and evaluation. Regression models are widely adopted in the fields of predictive analytics and software engineering due to their straightforward structure, computational efficiency, and interpretability. These models are effective in capturing linear relationships between input features and target variables, making them suitable for many practical applications.

In this context, the regression model was used to predict both time and cost efforts required for Agile software projects. The model demonstrated a solid performance, achieving an R^2 score of 83% for time estimation and 78% for cost estimation. These results indicate that the regression model was able to effectively learn from the training data and generate reasonable predictions on unseen real project data. The Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) values further support its capability to estimate effort with acceptable accuracy. Table 4.4 presents the results of the regression model, highlighting its performance metrics and predictive accuracy based on the evaluated dataset.

Table 4.4: Result of regression model

Parameter	**R^2-Score**	**MAE**	**RMSE**
Time	83	0.56	0.48
Cost	78	0.51	0.040

It is important to note that while Agile environments are often characterized by rapid changes, evolving requirements, and non-linear development processes, the regression model still managed to deliver consistent and informative outputs. This highlights its potential as a practical tool for organizations seeking a balance between predictive accuracy and model transparency. The regression model provides a reliable and interpretable framework for estimating time and cost in Agile projects, particularly when working with structured datasets and when a clear understanding of feature impact is desired.

4.3 Comparison of Stacking Model

This section discusses the comparison of stacking model with data augmentation technique and without data augmentation technique and comparison of results between correlation analysis and without correlation analysis.

4.3.1 Comparison of stacking model with and without data augmentation

In this section, a comparison is made between with and without the augmentation technique. Table 4.5 presents a comparison between the Stacking Model and the Stacking Model enhanced with a data augmentation technique. The results indicate that the application of data augmentation slightly improved the model's performance.

Table 4.5: Comparison of Stacking Model with and without data augmentation technique

Model	R^2-Score	MAE	RMSE
Stacking Model	**Time:** 90	**Time:** 0.341	**Time:** 0.020
	Cost: 97	**Cost:** 0.38	**Cost:** 0.180
Stacking Model with data augmentation Technique	**Time:** 91.51	**Time:** 0.312	**Time:** 0.020
	Cost: 98	**Cost:** 0.35	**Cost:** 0.170

Stacking Model without data augmentation technique: The stacking model's R-squared score is 90, meaning that 90% of the variance in the dependent variable (target) can be explained by the independent variables (features) in the model. The average absolute error, which displays the average difference between the predicted and actual values, is 0.341. The dispersion of errors between the anticipated and actual values is measured by the RMSE (root mean square error) of 0.020 for the stacking model. Stacking Model with data augmentation technique. The R-squared score for the Stacking Model with data augmentation technique is 91.51. This

indicates that this model explains 91.51% of the variance in the target variable, which is slightly higher than the for this model is 0.020, this suggests that despite the slightly lower MAE, the spread of errors around the actual values remains similar between the two models. Both models have high R-squared scores, indicating good explanatory power. The model with the data augmentation technique performs well as the values shown in Table 4.5.

4.3.2 Comparison of stacking model with and without correlation analysis

In this section, a comparison is made between the dataset without correlation analysis and the dataset with correlation analysis. Table 4.6 provides a comparison between the Stacking Model trained with a Correlation Analysis Dataset and the model trained without correlation analysis. The results show that incorporating correlation analysis significantly improved model performance.

Table 4.6: Comparison of results without correlation analysis and with correlation analysis

Model	R^2-Score	MAE	RMSE
Stacking Model With Correlation Analysis Dataset	**Time:** 90	**Time:** 0.341	**Time:** 0.020
	Cost: 97	**Cost:** 0.38	**Cost:** 0.0.180
Stacking Model without Correlation Analysis Dataset	**Time:** 85	**Time:** 1.65	**Time:** 0.026
	Cost: 93	**Cost:** 1.68	**Cost:** 0.0220

RMSE is slightly lower for the Stacking Model with a Correlation Analysis Dataset (0.020) compared to the Stacking Model without a Correlation Analysis Dataset (0.026). This suggests that the model without correlation analysis makes predictions that are, on average, closer to the actual values.

4.4 Comparison of Results with Predictions in the Literature

This section discusses the comparison of various techniques with each other. The comparison was conducted to validate and benchmark our research against existing studies in the time and cost prediction. Such comparisons are essential to understand how our proposed approach performs relative to established methods, identify its strengths, and assess areas where it brings improvements. The studies [90] and [52] were chosen for comparison based on clear and relevant criteria, both studies focus on predicting time and cost in projects, which directly aligns with the goals of our research. These studies present their findings using widely recognized metrics R2-Score, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) for meaningful comparison.

The first baseline paper, [90], demonstrates strong performance in predicting Time, as evidenced by its high R2-Score, indicating that the model effectively explains the variance in time-related data. However, its performance in predicting Cost is less reliable, with high RMSE and MAE values. These metrics suggest that the model struggles to accurately predict cost, resulting in larger errors and reduced precision for this variable. The second baseline paper, [52], exhibits moderate performance across all metrics While it does not achieve exceptional results in any particular area, it also does not fall short in any of the evaluated categories. Its performance is generally in the mid-range, indicating that it delivers adequate outcomes, but there is potential for improvement in certain areas.

Our research project is really good at predicting Time and Cost. The model has high R^2-scores, which means it explains most of the changes in these two factors. This means our model is accurate and works well for forecasting Time and Cost, making it reliable for practical use. This comparison highlights the superior performance of our model in terms of both precision and consistency. The findings demonstrate its potential applicability in real-world project management scenarios. This strong predictive capability adds value to the overall efficiency of project evaluations. The comparison clearly shows that our model performs competitively with, and in some cases outperforms, existing approaches. This reinforces the effectiveness of our methodology in accurately predicting Time and Cost. Table 4.7 presents a comparison of the results from our research project with previous studies documented in the literature.

Table 4.7: Comparison of Results with Predictions in the Literature

Author	R^2-Score	MAE	RMSE
[90]	**Time:** 95	**Time:** N/A	**Time:** 2.4
	Cost: 91	**Cost:** N/A	**Cost:** 51,37
[52]	**Time:** 0.94	**Time:** N/A	**Time:** 0.053
	Cost: 0.94	**Cost:** N/A	**Cost:** 0.0546
Our Research Project	**Time:** 0.91	**Time:**0.312	**Time:** 0.020
	Cost: 0.98	**Cost:** 0.35	**Cost:** 0.170

4.5 Comparison of Stacking Model with Regression Model

A comparative evaluation explores the performance of the stacking model against the regression model developed in this research. This internal comparison highlights the predictive capability of the ensemble-based stacking model in contrast to a traditional regression approach within the context of Agile effort estimation. Both models rely on the same augmented dataset consisting of 77 Agile software projects. The dataset includes 54 synthetic entries for training and 23 real entries for testing. Standard evaluation metrics R^2 Score, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) serve to assess model effectiveness in predicting both time and cost.

The regression model, recognized for its simplicity and interpretability, delivers moderate accuracy levels. Results indicate 83% R^2 for time prediction and 78% for cost. MAE and RMSE values reflect a reasonable level of precision but show a tendency toward larger errors, particularly in cost estimation. On the other hand, the stacking model demonstrates superior predictive power. Achieving an R^2 score of 91.51% for time and 98% for cost, the model maintains lower MAE and RMSE values across both parameters. Its ensemble learning structure enables it to capture more complex patterns and nonlinear relationships in the dataset.

This comparison illustrates the stacking model's effectiveness in Agile software effort estimation. By leveraging multiple learning algorithms, the ensemble approach adapts more efficiently to the complexities of project data. The stacking model consistently delivers higher accuracy and lower error rates, making it a strong candidate for real-world applications where precise forecasting is critical. Table 4.8 shows a comparison between the Stacking model and the regression model, illustrating differences in prediction accuracy and overall performance.

Table 4.8: Comparison of Stacking model with regression model

Model	R^2-Score	MAE	RMSE
Regression model	**Time:** 83	**Time:** 0.56	**Time:** 0.040
	Cost: 78	**Cost:** 0.51	**Cost:** 0.048
Stacking Model	**Time:** 91	**Time:** 0.312	**Time:** 0.020
	Cost: 98	**Cost:** 0.35	**Cost:** 0.170

4.6 Threat to Validity

In research, threats to validity are crucial considerations as they impact the accuracy, reliability, and generalizability of the results [91].

Internal Validity: Internal validity refers to the extent to which a study establishes a reliable cause-and-effect relationship between the independent and dependent variables [91]. To ensure internal validity, the features used in this study were carefully chosen based on their importance, as highlighted in previous research studies [90] and [2]. These studies identified key factors that significantly influence project outcomes like cost and time, providing a reliable basis for feature selection. By relying on these findings, we ensured that the chosen features are relevant and impactful, reducing the risk of missing important variables. In addition, we performed rigorous data validation to maintain accuracy and consistency throughout the

analysis. This included handling missing values, identifying and addressing outliers, and ensuring the dataset was free from errors or inconsistencies. By carefully selecting relevant features and thoroughly validating the data, the study ensures reliable and trustworthy results.

External Validity: External validity refers to how well the findings of a study can be applied or generalized to different contexts, populations, or settings beyond the specific conditions of the research [91]. To ensure the external validity of this research, the findings were designed to be applicable across different agile software development contexts. The study focused on collecting datasets from various industries and project types to ensure diversity and representativity. This approach supported the generalizability of the results to a wide range of real-world scenarios. Additionally, the research utilized data augmentation techniques, creating simulated datasets that replicated various project settings. Validate the model on unseen data or real-world projects from industries or organizations that were not part of the initial dataset. This step ensures the model performs reliably outside the training conditions.

4.7 Summary

The examination of the research findings is covered in this chapter. Correlation analysis is performed on the dataset, and data augmentation techniques are employed to increase the dataset size. A comparison is made between the results obtained with correlation analysis, those without correlation analysis, and those with data augmentation but without feature selection. Various metrics are being utilized for evaluation purposes, and the outcomes are presented in the form of graphs. At the last of chapter threats to validity are also discuss

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

This research aimed to address the ongoing challenges associated with effort estimation in Agile software development an essential yet complex task that, when miscalculated, often results in budget overruns, time delays, and overall project inefficiencies. Traditional estimation techniques, while valuable in structured development environments, frequently fall short in capturing the iterative and dynamic nature of Agile projects. This gap underscores the need for more adaptive and intelligent estimation models.

To address this issue, the study introduced an ensemble learning approach specifically, the stacking technique to improve prediction accuracy. The experimental phase involved compiling a dataset specific to Agile software projects and evaluating the model's performance using multiple evaluation metrics. The results demonstrated that the stacking model outperformed traditional estimation approaches in terms of accuracy and consistency. Specifically, the model achieved 97% accuracy in estimating cost and 90% accuracy in estimating time, marking a significant improvement in prediction reliability.

These findings highlight the effectiveness of ensemble learning techniques in improving the precision of effort estimations. The use of stacking allows for the integration of multiple predictive models, leveraging their combined strengths while minimizing individual weaknesses. This contributes meaningfully to the growing body of knowledge surrounding machine learning applications in Agile software development. The results also reinforce the broader idea that the more intelligent and data-informed the estimation approach, the more realistic and dependable the outcomes will be.

Moreover, the study reinforced the broader idea that the more intelligent and data-informed the estimation approach, the more realistic and dependable the outcomes will be. By incorporating diverse models, it becomes possible to reflect the multifaceted nature of Agile projects and produce more nuanced estimations. The results also emphasize the importance of continuous learning and model refinement, suggesting that as more data becomes available from completed projects, the models can be retrained to improve their accuracy and adapt to emerging trends in Agile development practices.

Furthermore, the study suggests that incorporating more granular and context-specific project details such as team dynamics, historical data, and risk-handling strategies could further enhance prediction accuracy. By using smarter, adaptive models that are able to learn from diverse project environments, effort estimation in Agile settings can become more precise and robust, ultimately leading to better decision-making and project success.

5.2 Future Direction

Although the current research has demonstrated the effectiveness of stacking models for effort estimation in Agile environments, there remains considerable scope for future enhancements. One key area for improvement is the inclusion of richer project-related data. Integrating detailed information about project management practices, such as how teams respond to changes in requirements, manage unexpected delays, or address technical obstacles, could significantly enhance the realism and accuracy of effort predictions.

For instance, incorporating data on previous risk incidents and their mitigation strategies would allow models to learn from historical challenges and better anticipate similar issues in new projects. This approach could help build more robust predictive systems that adapt not only to project size and scope but also to organizational behavior and team resilience under varying circumstances.

Additionally, future work could explore the adaptation of the proposed model to a wider range of Agile methodologies. While this research focused on a general Agile framework, methodologies like Kanban, Extreme Programming (XP), and lean each have unique practices and workflows that may influence estimation needs. Developing flexible models that can

accommodate these methodological differences would extend the applicability of the research and ensure broader relevance across Agile implementations.

Moreover, the study suggests that the development of real-time prediction models, capable of continuously updating based on live project data, would be a valuable contribution to Agile project management. By incorporating real-time data feeds on project progress, team performance, and external factors such as market conditions, a dynamic prediction model could provide ongoing effort estimations throughout the lifecycle of a project. This would allow teams to make timely adjustments to their strategies and avoid the costly consequences of misestimations that are often discovered too late in the process.

Finally, collaboration with industry practitioners to validate these models in real-world settings will be crucial for further refinement and fine-tuning. Although the study's experimental results are promising, applying the models in real-world Agile projects will provide valuable insights into their practicality, adaptability, and scalability. This collaboration will not only help enhance the model's accuracy but also ensure its usability in different organizational contexts. By combining academic rigor with practical application, the research can contribute to the adoption of machine learning-based estimation techniques within the software development industry, ultimately leading to more efficient project management and successful outcomes.

In conclusion, while the current research has laid a strong foundation for using ensemble learning models in Agile effort estimation, future efforts should focus on enriching the data used, adapting the models to various Agile methodologies, and integrating real-time predictive capabilities to further enhance the accuracy and adaptability of these models in dynamic project environments.

REFERENCES

[1] J. Murillo-Morera, C. Quesada-López, C. Castro-Herrera, and M. Jenkins, A genetic algorithm-based framework for software effort prediction, vol. 5, no. 1. Journal of Software Engineering Research and Development, 2017. doi: 10.1186/s40411-017- 0037-x.

[2] S. Kamal Tipu and S. k, "An Effort Estimation Model for Agile Software Development," Adv. Comput. Sci. its Appl., vol. 314, no. 1, pp. 2166–2924, 2012, [Online]. Available: www.worldsciencepublisher.org

[3] Ziauddin, S. K. Tipu, K. Zaman, and S. Zia, "Software Cost Estimation Using Soft Computing Techniques," Adv. Inf. Technol. Manag., vol. 2, no. 1, pp. 233–238, 2012, [Online]. Available: www.worldsciencepublisher.org

[4] "The Curious Case of the CHAOS Report 2009." Accessed: May 16, 2024. [Online]. Available: https://www.projectsmart.co.uk/it-project-management/the-curious- case-of-the-chaos-report-2009.php

[5] I. Thamarai and S. Murugavalli, "Using differential evolution in the prediction of software effort," 4th Int. Conf. Adv. Comput. ICoAC 2012, no. January, 2012, doi: 10.1109/ICoAC.2012.6416816.

[6] B. W. Boehm, "Software Engineering Economics," IEEE Trans. Softw. Eng., vol. SE-10, no. 1, pp. 4–21, 1984, doi: 10.1109/TSE.1984.5010193.

[7] L. H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Trans. Softw. Eng., vol. SE-4, no. 4, pp. 345–361, 1978, doi: 10.1109/TSE.1978.231521.

[8] A. J. Albrecht and J. E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Trans. Softw. Eng., vol. SE-9, no. 6, pp. 639–648, 1983, doi: 10.1109/TSE.1983.235271.

[9] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro, "Genetic programming for effort estimation: An analysis of the impact of different fitness functions," Proc. - 2nd Int.Symp. Search Based Softw. Eng. SSBSE 2010, no. October, pp. 89–98, 2010, doi: 10.1109/SSBSE.2010.20.

[10] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, "Cost models for future software life cycle processes: COCOMO 2.0," Ann. Softw. Eng., vol. 1, no. 1, pp. 57–94, 1995, doi: 10.1007/BF02249046.

[11] M. Cohen, "User Stories Applied," Engineering, 2004.

[12] J. W. Grenning, "Planning Poker Planning Poker or How to avoid analysis paralysis while release planning," Hawthorn Woods Renaiss. Softw. Consult., vol. 3, no. April, pp. 22–23, 2002.

[13] S. Porru, A. Murgia, S. Demeyer, M. Marchesi, and R. Tonelli, "Estimating story points from issue reports," ACM Int. Conf. Proceeding Ser., no. September, 2016, doi: 10.1145/2972958.2972959.

[14] A. Jain, Shilpi; Purohit, “Estimating Earned Business Value for Agile Projects using Relative Scoring Method,” ResearchGate, no. May, 2015, doi: 10.13140/RG.2.1.2693.2645.

[15] S. Sharma, D. Kumar, and M. E. Fayad, “An Impact Assessment of Agile Ceremonies on Sprint Velocity under Agile Software Development,” 2021 9th Int. Conf. Reliab. Infocom Technol. Optim. (Trends Futur. Dir. ICRITO 2021, no. May, 2021, doi: 10.1109/ICRITO51393.2021.9596508.

[16] E. Rodr, “Software Effort Estimation for Agile Software Development Using a Strategy Based on k-Nearest Neighbors Algorithm,” pp. 11–16, 2022.

[17] L. Cao, “Estimating Efforts for Various Activities in Agile Software Development: An Empirical Study,” IEEE Access, vol. 10, no. August, pp. 83311–83321, 2022, doi: 10.1109/ACCESS.2022.3196923.

[18] S. M. Satapathy and S. K. Rath, “Empirical assessment of machine learning models for agile software development effort estimation using story points,” Innov. Syst. Softw. Eng., vol. 13, no. 2–3, pp. 191–200, 2017, doi: 10.1007/s11334-017-0288-z.

[19] S. Kang, O. Choi, and J. Baik, “Model-based dynamic cost estimation and tracking method for agile software development,” Proc. - 9th IEEE/ACIS Int. Conf. Comput. Inf. Sci. ICIS 2010, pp. 743–748, 2010, doi: 10.1109/ICIS.2010.126.

[20] C. Nagar, “Software efforts estimation using Use Case Point approach by increasing Technical Complexity and Experience Factors,” Int. J. Comput. Sci. Eng., vol. 3, no. 10, pp. 3337–3345, 2011.

[21] O. Marbán, a D. A. Seco, J. Cuadrado, and L. García, “Cost Drivers of a Parametric Cost Estimation Model for Data Mining Projects (DMCOMO).,” Adis, no. May 2014, 2002, [online]. Available: http://www.researchgate.net/publication/220958274_Cost_Drivers_of_a_Parametric_Cost_Estimation_Model_for_Data_Mining_Projects_(DMCOMO)/file/79e4150a929859bd94.pdf

[22] Z. Abdelali, H. Mustapha, and N. Abdelwahed, “Investigating the use of random forest in software effort estimation,” Procedia Comput. Sci., vol. 148, pp. 343–352, 2019, doi: 10.1016/j.procs.2019.01.042.

[23] A. Mahindru and A. L. Sangal, Perbdroid: Effective malware detection model developed using machine learning classification techniques, vol. 185. 2020. doi: 10.1007/978-3-030-40928-9_7.

[24] J. M. Desharnais, L. Buglione, and B. Kocatürk, “Using the COSMIC method to estimate Agile User Stories,” ACM Int. Conf. Proceeding Ser., pp. 68–73, 2011, doi: 10.1145/2181101.2181117.

[25] M. Jørgensen, “A review of studies on expert estimation of software development effort,” J. Syst. Softw., vol. 70, no. 1–2, pp. 37–60, 2004, doi: 10.1016/S0164-1212(02)00156-5.

[26] K. Klokgieters and R. Chu, “Creating an Environment for Successful Innovation,” Evol. Innov. Manag., pp. 1–7, 2013, doi: 10.1057/9781137299994.0021.

[27] M. Kuutila, M. Mäntylä, U. Farooq, and M. Claes, “Time pressure in software engineering: A systematic review,” Inf. Softw. Technol., vol. 121, 2020, doi: 10.1016/j.infsof.2020.106257.

[28] M. Khazaiepoor, A. K. Bardsiri, and F. Keynia, “A Dataset-Independent Model for Estimating Software Development Effort Using Soft Computing Techniques,” Appl. Comput. Syst., vol. 24, no. 2, pp. 82–93, 2019, doi: 10.2478/acss-2019-0011.

[29] C. C. Hsu and B. A. Sandford, “The Delphi technique: Making sense of consensus,” Pract. Assessment, Res. Eval., vol. 12, no. 10, pp. 1–8, 2007.

[30] F. C. Waksler, Studying the social worlds of children: Sociological readings. 2003. doi: 10.4324/9780203214770.

[31] N. Sharma, A. Bajpai, and R. Litoriya, “A comparison of software cost estimation methods: A Survey,” Int. J. Comput. Sci. Appl., vol. 1, no. 3, pp. 121–127, 2012.

[32] L. Angelis, I. Stamelos, and M. Morisio, “Building a software cost estimation model based on categorical data,” Int. Softw. Metrics Symp. Proc., pp. 4–15, 2001, doi: 10.1109/metric.2001.915511.

[33] S. S. Gautam and V. Singh, “The state-of-the-art in software development effort estimation,” J. Softw. Evol. Process, vol. 30, no. 12, 2018, doi: 10.1002/smr.1983.

[34] M. Jørgensen, “Top-down and bottom-up expert estimation of software development effort,” Inf. Softw. Technol., vol. 46, no. 1, pp. 3–16, 2004, doi: 10.1016/S0950-5849(03)00093-4

[35] A. Sharma and D. S. Kushwaha, “Estimation of Software Development Effort from Requirements Based Complexity,” Procedia Technol., vol. 4, pp. 716–722, 2012, doi: 10.1016/j.protcy.2012.05.116.

[36] V. Nasteski, “An overview of the supervised machine learning methods,” Horizons.B, vol. 4, no. December 2017, pp. 51–62, 2017, doi: 10.20544/horizons.b.04.1.17. p05.

[37] L. Torgo and J. Gama, “Regression by classification,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 1159, pp. 51–60, 1996, doi: 10.1007/3-540-61859-7_6.

[38] F. Palla and S. W. Stahler, “Accelerating Star Formation in Clusters and Associations,” Astrophys. J., vol. 540, no. 1, pp. 255–270, 2000, doi: 10.1086/309312.

[39] Y. Forghani, R. S. Tabrizi, H. S. Yazdi, and M. R. Akbarzadeh-T, “Fuzzy support vector regression,” 2011 1st Int. Conference Comput. Knowl. Eng. ICCKE 2011, no. Vc, pp. 28–33, 2011, doi: 10.1109/ICCKE.2011.6413319.

[40] F. Zhang and L. J. O’Donnell, Support vector regression. Elsevier Inc., 2019. doi: 10.101A6/B978-0-12-815739-8.00007-9.

[41] T. M. H. Hope, “Linear regression,” Mach. Learn. Methods Appl. to Brain Disord., pp. 67–81, 2019, doi: 10.1016/B978-0-12-815739-8.00004-3.

[42] Y. Song, J. Liang, J. Lu, and X. Zhao, “An efficient instance selection algorithm for k nearest neighbor regression,” Neurocomputing, vol. 251, pp. 26–34, 2017, doi: 10.1016/j.neucom.2017.04.018.

[43] B. Bakir, “Defect Cause Modeling with Decision Tree and Regression Analysis: a Case Study in Casting Industry,” vol. 2, no. August 2016, pp. 1–5, 2007.

[44] M. I. Khan, “An Effort Estimation Model for Agile Software Development By degree of Ph. D. sin Computer Science Institute of Computing and Information Technology Gomal University Dera Ismail Khan Pakistan,” no. July, 2020.

[45] O. Sagi and L. Rokach, “Ensemble learning: A survey,” Wiley Interdiscip. Rev. Data Min. Knowl. Discov., vol. 8, no. 4, pp. 1–18, 2018, doi: 10.1002/widm.1249.

[46] T. G. Dietterichl, “The handbook of brain theory and neural networks-ensemble learning,” MIT Press, vol. 40, 2002, [Online]. Available: https://courses.cs.washington.edu/courses/cse446/12wi/tgd-ensembles.pdf

[47] S. González, S. García, J. Del Ser, L. Rokach, and F. Herrera, “A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities,” Inf. Fusion, vol. 64, no. July, pp. 205–237, 2020, doi: 10.1016/j.inffus.2020.07.007

[48] R. O. Odegua, “An Empirical Study of Ensemble Techniques (Bagging, Boosting and Stacking) Rising Odegua Nossa Data An Empirical Study of Ensemble Techniques (Bagging, Boosting and Stacking),” Proc. Conf. Deep Learn, no. March 2019, 2019, [online]. Available: https://www.researchgate.net/publication/338681864

[49] W. Jiang, Z. Chen, Y. Xiang, D. Shao, L. Ma, and J. Zhang, “Sem: A novel self- adaptive stacking ensemble model for classification,” IEEE Access, vol. 7, pp. 120337–120349, 2019, doi: 10.1109/ACCESS.2019.2933262.

[50] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, “Understanding Data Augmentation for Classification: When to Warp?,” 2016 Int. Conf. Digit. Image Comput. Tech. Appl. DICTA 2016, 2016, doi: 10.1109/DICTA.2016.7797091.

[51] L. Song, “Learning to Cope with Small Noisy Data in Software Effort,” no. January, 2019.

[52] E. R. Sanchez, H. C. Maceda, and E. V. Santacruz, “Software Effort Estimation for Agile Software Development Using a Strategy Based on k-Nearest Neighbors Algorithm,” 2022 IEEE Mex. Int. Conf. Comput. Sci. ENC 2022 - Proc., pp. 11–16, 2022, doi: 10.1109/ENC56672.2022.9882947.

[53] A. Sharma and N. Chaudhary, “Analysis of Software Effort Estimation Based on Story Point and Lines of Code using Machine Learning,” Int. J. Comput. Digit. Syst., vol. 12, no. 1, pp. 131–140, 2022, doi: 10.12785/ijcds/120112.

[54] S. Kumar and M. P. Singh, “An Improved Technique for Software cost estimations in Agile Software development using Soft Computing.,” IET Conf. Proc., vol. 2023, no. 5, pp. 514–519, 2023, doi: 10.1049/icp.2023.1541.

[55] S. Bilgaiyan, S. Mishra, and M. Das, “Effort estimation in agile software development using experimental validation of neural network models,” Int. J. Inf. Technol., vol. 11, no. 3, pp. 569–573, 2019, doi: 10.1007/s41870-018-0131-2.

[56] A. Saini, L. Ahuja, and S. K. Khatri, “Effort Estimation of Agile Development using Fuzzy Logic,” 2018 7th Int. Conf. Reliab. Infocom Technol. Optim. Trends Futur. Dir. ICRITO 2018, pp. 779–783, 2018, doi: 10.1109/ICRITO.2018.8748381.

[57] A. Sharma and N. Chaudhary, “Linear Regression Model for Agile Software Development Effort Estimation,” 2020 5th IEEE Int. Conf. Recent Adv. Innov. Eng. ICRAIE 2020 - Proceeding, vol. 2020, pp. 4–7, 2020, doi: 10.1109/ICRAIE51050.2020.9358309.

[58] O. Malgonde and K. Chari, an ensemble-based model for predicting agile software development effort, vol. 24, no. 2. Empirical Software Engineering, 2019. doi: 10.1007/s10664-018-9647-0.

[59] S. Dragicevic, S. Celar, and M. Turic, “Bayesian network model for task effort estimation in agile software development,” J. Syst. Softw., vol. 127, pp. 109–119, 2017, doi: 10.1016/j.jss.2017.01.027.

[60] A. Zakrani, A. Najm, and A. Marzak, “Support Vector Regression Based on Grid- Search Method for Agile Software Effort Prediction,” Colloq. Inf. Sci. Technol. Cist, vol. 2018-Octob, pp. 492–497, 2018, doi: 10.1109/CIST.2018.8596370.

[61] A. Kaushik, D. K. Tayal, and K. Yadav, “A Comparative Analysis on Effort Estimation for Agile and Non-agile Software Projects Using DBN-ALO,” Arab. J. Sci. Eng., vol. 45, no. 4, pp. 2605–2618, 2020, doi: 10.1007/s13369-019-04250-6.

[62] T. Khuat and L. Thi My Hanh, “An Effort Estimation Approach for Agile Software Development using Fireworks Algorithm Optimized Neural Network,” Int. J. Comput. Sci. Inf. Secur., vol. 14, no. August, pp. 122–130, 2016.

[63] T. T. Khuat and M. H. Le, “A Novel Hybrid ABC-PSO Algorithm for Effort Estimation of Software Projects Using Agile Methodologies,” J. Intell. Syst., vol. 27, no. 3, pp. 489–506, 2018, doi: 10.1515/jisys-2016-0294.

[64] P. Suresh Kumar and H. S. Behera, Estimating Software Effort Using Neural Network: An Experimental Investigation, vol. 1120. Springer Singapore, 2020. doi: 10.1007/978-981-15-2449-3_14.

[65] P. Suresh Kumar, H. S. Behera, J. Nayak, and B. Naik, “A pragmatic ensemble learning approach for effective software effort estimation,” Innov. Syst. Softw. Eng., 2021, doi: 10.1007/s11334-020-00379-y.

[66] A. A. Abdulmajeed, M. A. Al-Jawaherry, and T. M. Tawfeeq, “Predict the required cost to develop Software Engineering projects by Using Machine Learning,” J. Phys. Conf. Ser., vol. 1897, no. 1, 2021, doi: 10.1088/1742-6596/1897/1/012029.

[67] S. P. Singh, “Cost estimation model using enhance-based differential evolution algorithm,” Iran J. Comput. Sci., vol. 3, no. 2, pp. 115–126, 2020, doi: 10.1007/s42044-019-00049-8.

[68] P. Singal, A. C. Kumari, and P. Sharma, “Estimation of Software Development Effort: A Differential Evolution Approach,” Procedia Comput. Sci., vol. 167, no. 2019, pp. 2643–2652, 2020, doi: 10.1016/j.procs.2020.03.343.

[69] Y. Mahmood, N. Kama, A. Azmi, A. S. Khan, and M. Ali, “Software effort estimation accuracy prediction of machine learning techniques: A systematic performance evaluation,” Softw. - Pract. Exp., vol. 52, no. 1, pp. 39–65, 2022, doi: 10.1002/spe.3009.

[70] A. Ullah, B. Wang, J. Sheng, J. Long, M. Asim, and F. Riaz, “A novel technique of software cost estimation using flower pollination algorithm,” Proc. - 2019 Int. Conf. Intell. Comput. Autom. Syst. ICICAS 2019, no. May 2020, pp. 654–658, 2019, doi: 10.1109/ICICAS48597.2019.00142.

[71] Z. A. Khalifelu and F. S. Gharehchopogh, “Comparison and evaluation of data mining techniques with algorithmic models in software cost estimation,” Procedia Technol., vol. 1, pp. 65–71, 2012, doi: 10.1016/j.protcy.2012.02.013.

[72] A. Puspaningrum and R. Sarno, “A Hybrid Cuckoo Optimization and Harmony Search Algorithm for Software Cost Estimation,” Procedia Comput. Sci., vol. 124, pp. 461–469, 2017, doi: 10.1016/j.procs.2017.12.178.

[73] H. P. Patra and K. Rajnish, “A new empirical model to increase the accuracy of software cost estimation,” Int. J. Eng. Trans. A Basics, vol. 30, no. 10, pp. 1487– 1493, 2017, doi: 10.5829/ije.2017.30.10a.09.

[74] V. S. Desai and R. Mohanty, “ANN-Cuckoo Optimization Technique to Predict Software Cost Estimation,” 2018 Conf. Inf. Commun. Technol. CICT 2018, pp. 1– 6, 2018, doi: 10.1109/INFOCOMTECH.2018.8722380.

[75] M. M. Al Asheeri and M. Hammad, “Machine learning models for software cost estimation,” 2019 Int. Conf. Innov. Intell. Informatics, Comput. Technol. 3ICT 2019, no. December, 2019, doi: 10.1109/3ICT.2019.8910327.

[76] A. Panda, S. M. Satapathy, and S. K. Rath, “Empirical Validation of Neural Network Models for Agile Software Effort Estimation based on Story Points,” Procedia Comput. Sci., vol. 57, pp. 772–781, 2015, doi: 10.1016/j.procs.2015.07.474.

[77] A. Y. M and O. E. D, Challenges of Enforcement of Forestry Legislation in Taraba State, Nigeria, vol. 6, no. 3. 2017. doi: 10.18488/journal.10/2017.6.3/10.3.48.57.

[78] A. J. Myles, R. N. Feudale, Y. Liu, N. A. Woody, and S. D. Brown, “An introduction to decision tree modeling,” J. Chemom., vol. 18, no. 6, pp. 275–285, 2004, doi: 10.1002/cem.873.

[79] D. N. Bonter and C. B. Cooper, “Data validation in citizen science: A case study from Project FeederWatch,” Front. Ecol. Environ., vol. 10, no. 6, pp. 305–307, 2012, doi: 10.1890/110273.

[80] A. Fruscione , “CIAO: et data analysis system,” Obs. Oper. Strateg. Process. Syst., vol. 6270, p. 62701V, 2006, doi: 10.1117/12.671760.

[81] N. J. Gogtay and U. M. Thatte, “Principles of correlation analysis,” J. Assoc. Physicians India, vol. 65, no. MARCH, pp. 78–81, 2017.

[82] M. Fernández-Diego, E. R. Méndez, F. González-Ladrón-De-Guevara, S. Abrahão, and E. Insfran, “An update on effort estimation in agile software development: A systematic literature review,” IEEE Access, vol. 8, pp. 166768–166800, 2020, doi: 10.1109/ACCESS.2020.3021664.

[83] A. Ash and M. Shwartz, “R2: A useful measure of model performance when predicting a dichotomous outcome,” Stat. Med., vol. 18, no. 4, pp. 375–384, 1999, doi: 10.1002/(SICI)1097-0258(19990228)18:4<375: AID-SIM20>3.0.CO;2-J.

[84] B. Tanveer, A. M. Vollmer, and S. Braun, “A hybrid methodology for effort estimation in agile development: An industrial evaluation,” in ACM International Conference Proceeding Series, Association for Computing Machinery, May 2018, pp. 21–30. doi: 10.1145/3202710.3203152.

[85] N. Benschop, C. A. R. Hilhorst, A. L. P. Nuijten, and M. Keil, “Detection of early warning signals for overruns in IS projects: linguistic analysis of business case language,” Eur. J. Inf. Syst., vol. 29, no. 2, pp. 190–202, 2020, doi: 10.1080/0960085X.2020.1742587.

[86] I. Jason, L. Davis, A. Prof, and S. G. Faculty, “TMGT 514 – Engineering & Technology Project Management COURSE SYLLABUS: Fall 2020 (208),” vol. 2020, no. 208, pp. 1–18, 2020.

[87] M. Owais and R. Rama Kishore, “Effort, duration and cost estimation in agile software development,” 2016 9th Int. Conf. Contemp. Comput. IC3 2016, pp. 1–5, 2017, doi: 10.1109/IC3.2016.7880216

[88] Z. Fan, “Handbook of Software Engineering and Knowledge Engineering - Vol 2: Emerging Technologies,” Handb. Softw. Eng. Knowl. Eng. - Vol 2 Emerg. Technol., no. April 2001, 2010, doi: 10.1142/9789812389701.

[89] A. G. Priya Varshini, K. Anitha Kumari, D. Janani, and S. Soundariya, “Comparative analysis of Machine learning and Deep learning algorithms for Software Effort Estimation,” in Journal of Physics: Conference Series, IOP Publishing Ltd, Feb. 2021. doi: 10.1088/1742-6596/1767/1/012019.

[90] E. Rodríguez Sánchez, E. F. Vázquez Santacruz, and H. Cervantes Maceda, “Effort and Cost Estimation Using Decision Tree Techniques and Story Points in Agile Software Development,” Mathematics, vol. 11, no. 6, 2023, doi: 10.3390/math11061477.

[91] R. Feldt and A. Magazinius, “Validity threats in empirical software engineering research - An initial survey,” SEKE 2010 - Proc. 22nd Int. Conf. Softw. Eng. Knowl. Eng., pp. 374–379, 2010.

[92] Z. Sahrawi, A. Sellami, and N. Bouassida, “Software Enhancement Effort Estimation using Stacking Ensemble Model within the Scrum Projects: A Proposed Web Interface,” no. January, pp. 91–100, 2022, doi: 10.5220/0011321000003266.

[93] S. Shukla and S. Kumar, “A Stacking Ensemble-based Approach for Software Effort Estimation,” Int. Conf. Eval. Nov. Approaches to Softw. Eng. ENASE - Proc., vol. 2021-April, no. Enase, pp. 205–212, 2021, doi: 10.5220/0010405002050212.

[94] A. G. Priya Varshini and K. Anitha Kumari, “Software Effort Estimation Using Stacked Ensemble Technique and Hybrid Principal Component Regression and Multivariate Adaptive Regression Splines,” Wirel. Pers. Commun., vol. 134, no. 4, pp. 2259–2278, 2024, doi: 10.1007/s11277-024-11010-9.

[95] D. Wolpert, “Stacked Generalization (Stacking),” Neural Networks, vol. 5, pp. 241–259, 1992.

[96] P. Sampath Kumar and R. Venkatesan, “Improving Accuracy of Software Estimation Using Stacking Ensemble Method,” pp. 219–227, 2021, doi: 10.1007/978-981-15-5243-4_18.

APPENDICES

APPENDIX A

QUESTIONNAIRE

Respected Participant,

I'm a research student in NUML University, Islamabad. My research topic is effort estimation in agile software development using Machine Learning. For this research, I proposed a model that will automatically estimate the software effort for the software projects. To train the ML model, I need software project information from the Pakistani software industry. The attributes (needed for this research) and their brief description are described in the attached document. You can provide the information of your projects by filling the form in the google form in the link below: Software development effort estimation (SDEE) is a crucial component of software management, which measures the effort of developing software. The process of estimation of software involves predicting the size of the software product, necessary development efforts, project schedules, and an estimation of the project's final cost. The information of attachment:

https://docs.google.com/forms/d/1eoHdrT7oy0d2zNTu8hCHJV-4bp7cDni6Qdfu29FD8bw/edit?usp=drivesdk

Effort estimation in agile software development:

During effort estimation, we determine the order in which the various tasks must be carried out to finish the project. Calculate the amount of time (in person-hours or days) needed to finish each task. Software effort estimation provides accurate estimates about the software development team or the size of the team. Budgeting, risk analysis, and project planning are just a few of the uses for effort estimation. Additionally, it aids in determining the resources that must be allocated to the project and their efficiency. This study helps to prioritize and categorize development projects according to the overall business plan. The effort estimation process is very critical because the success or failure of a project depends entirely on effort estimation. The attributes of effort estimation dataset are described below.

Sprint Size: Team needs to able to get user stories done/time required to complete any user story

Initial Velocity: Amount of effort completed per unit of time. There are some factors that may affect the velocity. These factors are friction forces and dynamic forces.

Friction forces: Friction forces are those negative forces that have negative impact on project productivity. They reduce team velocity. These forces may reduce by project manager or developer but cannot be eliminated.

Category	**Factors**
Product Factor	Product nature, product category, product usage, product performance & quality, product development complexity.
Project Factor	Project Constraints, Project Characteristics, Project Management, Risk Management
People Factor	Personal Expertise, Tool Expertise, Tool Availability
Process Factor	Process Maturity & stability

Dynamic forces: dynamic forces are often unexpected and unpredictable. these forces may cause project decelerate and cause loss of velocity. The following Dynamic Forces are selected that affect estimation process. Expected Team Changes, Introduction of New Tools, Vendor's Defect, Team member's responsibilities outside the project, Personal Issues, Expected Delay in Stakeholder response, Expected Ambiguity in Details, Expected Changes in environment, Expected Relocation.

Deceleration: Negative rate of change of velocity. Hence my case deceleration is the product of friction and dynamic forces and its effect team velocity

Final Velocity: Since Team velocity is deaccelerated by Friction Forces and Dynamic Forces. Thus, the final team velocity (V) is obtained by optimizing Vi as under

Deacceleration D =1/($FR*DF$) V

The final velocity is calculated as

V = (Vi)D

Actual Time: After the completion project we determined the time, the team invest

Actual cost: After the completion project we determined the cost

I am very thankful to you if you fill this form and it will help me in my research.

SR. NO	Question	Response data
Section 1: General Project questions		
1	What is your organization name?	
2	What was project name?	
3	What was project ID? (Not Compulsory)	
Section 2: Project Scope and effort		
4	What was the number of user stories for completion whole project?	
5	what was unit of effort completed in sprint?	
Section 3: Time and cost consideration		
6	what was sprint size?	
7	How many work days was in a month?	
8	How much salary were required for team per month? (Approximately)	
Section 4: Project challenges and influences		
9	what was the friction forces that affect the project regarding project, product, process, personal aspect?	
10	what was the dynamic forces that affect project?	
Section 5: Project timeline and budget		
11.	what was the actual time period in which you complete the project?	
12	what was the actual cost/budget when you start the project?	

APPENDIX B

Software Houses Information

SR.NO	Software House Name	Employe e Size	Domain
1	Ninesol Technologies	201-500	Android IOS application
2	Funsol Technologies	201-500	Android IOS application
3	Join App studio	51-200	Android IOS application
4	Netsol Technologies	1001-5000	Client Base and Mobile application
5	Autoadvisors.com	11-50	Web applications
6	Emumba	201-500	Web applications
7	Graana.com	501-1000	Web applications
8	9D technologies	51-200	Android IOS application
9	Techinn 360	10-5	Web applications
10	Beta Angels	11-50	Android IOS application
11	Funtash Technologies	51-200	Web applications