DEEP LEARNING BASED MODELS FOR SOFTWARE EFFORT ESTIMATION USING STORY POINTS IN AGILE ENVIRONMENTS

by

Rene Avalloni de Morais

A project report submitted in conformity with the requirements for the degree of Master of Science in Information Technology

Department of Mathematical and Physical Sciences
Faculty of Graduate Studies
Concordia University of Edmonton

MACHINE LEARNING AND NATURAL LANGUAGE PROCESSING BASED APPROACH FOR STORY POINTS ESTIMATION FOR AGILE ENVIRONMENTS

Rene Avalloni de Morais

Approved:

Supervisor	Date
Committee Member	Date
Dean of Graduate Studies: Rossitza Marinova, Ph.D.	Date

DEEP LEARNING BASED MODELS FOR SOFTWARE EFFORT ESTIMATION USING STORY POINTS IN AGILE ENVIRONMENTS

Rene Avalloni de Morais
Master of Science in Information Technology

Department of Mathematical and Physical Sciences
Concordia University of Edmonton
2022

Abstract

In the era of agile software development methodologies, traditional planning and software effort estimation methods are replaced to meet customer's satisfaction in agile environments. However, software effort estimation remains a challenge. Although teams have achieved better accuracy in estimating story points effort required to implement user stories or issues, these estimations mostly rely on subjective assessments, leading to inaccuracy and impacting software project delivery. Some researchers are pointing good results by the adoption of deep learning to address this issue. Given the foregoing, this study proposes deep learning-based models for story points estimation in agile projects. Different algorithms are proposed and trained over a large dataset for story points estimation made by 16 open-source projects. In addition, we take advantage of natural language processing techniques to excavate better features from the software requirements written as user stories.

This thesis is dedicated to the two people who most love and guide me during my entire life:

Izilda Augusta da Silva Avalloni de Morais
Roberto Avalloni de Morais

Even though they are in another country, they still inspire and support me in all my decisions.

Acknowledgments

I would like to extend my gratitude to my supervisor, Dr. Baidya Saha, whose guidance and support made this research possible. I am also very grateful to all my professors who helped and taught me during my graduate program. In special I thank Professor Dr. Rossitza Marinova, for all her support.

A good support system is important to surviving and staying sane in graduate school. Thanks to my family, friends, and classmates for sticking with me through this graduate program and throughout my endeavors.

I would like to express my deep appreciation to my wife, who's helped and has been a true and great supporter and has unconditionally loved me during my good and bad times. She was the reason I came to Canada and why I decided to pursue a master's degree. I always admired her, academically and personally, and she showed me that one step at a time was all it took to get me here, through my journey.

I also gratefully acknowledge the funding received through the Alberta Graduate Excellence Scholarship (AGES) by Alberta's Government.

Finally, I thank my God, my good Father, for letting me through all the difficulties and made things happen in my life. I will keep on trusting you for my future. Thank you, Lord.

Contents

List of Tables

List of Figures

List Of Abbreviations

AI Intelligence Artificial

ANN Artificial Neural Network

ASD Agile Software Development

ASEE Agile Software Effort Estimation

BOW Bag-of-words

CBOW Common Bag of words

CNN Convolutional Neural Network

COCOMO Constructive Cost Model

DL Deep Learning

DNN Deep Neural Network

EEE Esemble Effort Estimation

FP Functional Points

LOC Lines of Code

LSTM Long Short-Term Memory

MAE Mean Absolute Error

MIFS Mutual Information Feature Selection

ML Machine Learning

MSE Mean Square Error

MdAE Median Absolute Error

NLP Natural Language Processing

NN Neural Network

RF Random Forest

RHN Recurrent Highway Network

RMSE Root Mean Square Error

RNN Recurrent Neural Networks

SDEE Software Development Effort Estimation

SEE Software Effort Estimation

SLDC Software Development Life-Cycle

SVM Support Vector Machine

TF-IDF	Term Frequency — Inverse Document Frequency
UCP	Use Case Points
kNN	K-Nearest Neighbor

Chapter 1

Introduction

1.1 Software Effort Estimation and Agile

The software development process is a roadmap to create high-quality software or systems within the time expected by customers. In traditional software development, the entire process is well-documented during planning. However, this can cause many interferences since not all events will occur as expected, generating rework and re-plan. The traditional software environment is known as Waterfall or Waterfall-style, or larges or traditional [1]. The Waterfall models are strictly sequential and follow an extensive initial phase of requirements specification until the final phases of implementation, testing, and software maintenance. As a result, the work of a team may be held up until another team completes its tasks. More importantly, changes can increase delays and the effort required for software development exponentially.

Software effort estimation (SEE) or Software development effort estimation (SDEE) refers to how much effort is required to develop a software program or system [2]. It represents a decisive role for any kind of software projects [3]. SDEE is required to kick-off project budgets, and schedules [4]. It serves as input for planning software project development or maintenance. Thus, accuracy in estimating the effort required for software development is a crucial task for software projects and represents fundamental implications for budget and software development process management [5]. In addition, especially in waterfall models, “if management’s estimate is too low, then the software development team will be under considerable pressure to finish the product quickly, and hence the resulting software may not be fully functional or tested” [6].

In 2016, Rao and collaborators did a systematic literature review showing plenty of research done on effort estimation in traditional process models, and all considered techniques have not given accurate predictions. To surmount this prediction issue, practitioners from the software industry have integrated Agile methodologies into their processes [7]. In recent years, the software industry has been surrendering relevant digital transformation to keep on track in an increasingly competitive market. Thus, companies demand to react and adapt rapidly to changes and meet customer needs. The Agile methodology emerged due to the barriers battled in the traditional software process. Flexible scope and shorter phases for planning and execution allow companies to have better performance [8] and connect to new market realities more quickly. As Agile provide guidelines to work faster and more assertive by short iterations, software effort estimation had to adapt to the Agile scope and occur in every phase of the project. With this backdrop, Agile Software Development (ASD) and Agile Software Effort Estimation (ASEE) methods have been largely implemented across many organizations [9], having customer's expectations as the primary goal [10]. Agile teams commonly use Story Points to estimate the effort required to implement or solve a given Story (User Story or issue). Story Point sizes are used to prioritize User Stories, plan and schedule coming iterations and releases, measure a team's progress, and even cost and allocate resources. Customers (Users) are constantly interacting in every stage of the Agile project. The feature requirements (User Stories) are susceptible to change, and the scope is frequently adjusted accordingly [11]. Hence, at the interaction level, the epics are broken down into User Stories, and tasks and estimates are produced by the team. In contrast to the traditional methods, Story Points estimate the effort to complete the User Story or feature instead of the entire project. By iterative cycles (Sprints in Scrum), the team estimate and plan deliveries of a set of User Stories (incremental deliveries).

Story Points estimation considers the effort needed to accomplish a task (User Story or issues), considering the amount of work to do, risk, uncertainty, and complexity. For Story Points estimation, Agile teams mostly relies on subjective assessments such as, Planning Poker, analogy, and expert judgment, whereas Planning Poker is the most common used technique [12], [13]. However, software effort estimation is still a challenge, like any estimation, has an inherent risk of uncertainty [14], as it is only possible to be sure of the effort required after the software project conclusion. The lack of relevant information, subject metrics and complex interactions can cause imprecision issues [15], [16]. Thus, having good historical information of previous projects is crucial to estimate accurately. Henceforward, a prediction model from Story Points past estimations already made should support Agile teams when estimating the size of new issues or features in Stories.

1.2 Background

Many software companies adopted Story Points estimation for Agile environments [3], [13]. Most projects use tracking systems, such as Atlassian Jira [17], to register and manage User Stories and issues. Story Points estimation is related to overall sizing the workload needed by the team, which should remain the same size for everyone. For instance, a User Story that contains one Story Point is less complex than the one addressed three Story Points. While estimating Story Points, the whole team participates and agrees on the amount of work required, as the definition of time can be different for each developer. Scrum teams mostly employ the Planning Poker [13] method to integrate and encourage interaction between team members, allowing everyone to express their opinions about the User Stories and reach a consensus of Story Point size in the end. The practice of estimating software effort using Story Points in agile software development environment mostly relies on subjective assessments (e.g. planning poker, analogy, and expert judgment) and historical data of project for estimation of cost, size, effort and duration [3]. Expert opinion based estimation methods may lead to inaccuracy of effort estimation [14], [16]. However, with no historical data and specialists, the prior methods like planning poker and analogy are pointless [18]. Although some improvements have been made, accuracy in estimating software effort in agile environments is still a challenge [3]. Literature reveals that data driven estimation models can increase software effort estimation accuracy. Since the 1980s, numerous estimation methods have been proposed, regression-based methods are predominant and, machine learning (ML) has been employed to solve this problem [2]. Several systematic reviews were done on software effort estimation. In 2016, Idri et al. (2016) conducted a systematic review of ensemble effort estimation (EEE) from works published between 2000 and 2016 and observed that machine learning single models are the most common approach in EEE. Further, EEE techniques typically generate satisfactory estimation accuracy, implying more accuracy than single models [19]. Within this context, the research community and industry have been combining machine learning and deep learning techniques with Agile methodologies and it has been proving greater accuracy on agile software development effort estimation prediction [3], [20]. The scope of this work is limited to develop machine learning and deep learning based strategies for agile software effort estimation using story points.

1.3 Problem Statement

In the Agile software industry, accurate software estimation effort is crucial for successful project development. Most software projects need to make important decisions

from the beginning of the project based on initial estimation efforts. In some cases, these estimates are necessary for evaluating contractor cost proposals even before contracting for Agile Software Development projects [4]. However, when the project starts, there are no much data available to estimate the project coherently. In this scenario, it is recommended to use historical data of previous related projects across or outside the organization. Although there are various techniques for performing the estimates, each one faces different challenges. Estimates made by experts are subjective, and besides they may have much experience, they are prone to error. On the other hand, algorithm models can also be subjective since they depend on other variables as programming knowledge and the project member's experience with teamwork. Several machine learning research studies are being discussed to address this issue and support the team on estimating based on historical data from similar projects done previously. Recently, the use of deep learning techniques have been proving great results, achieving success in prediction use cases by capturing the inherent correlation in the training set, as well as tasks that require semantic and syntactic context capturing [20].
Accurate estimation of Story Points from user stories can help to complete the project within budget and schedule. Natural Language Processing (NLP) algorithms can excavate important features from the user story text data to map the corresponding Story Points. As trained features, the word embeddings models can be contextualized and capture word semantics. In addition, machine learning and deep learning can successfully discover the hidden complex patterns from the NLP features and estimate Story Points. This research work aims to explore the efficacy of machine learning especially the deep learning algorithms for estimating agile based software efforts using the Story Points data collected from similar previous projects.

1.4 State of the art

There is a prominent trend towards machine learning, and deep learning models for story points estimation prediction based on the use of historical data of agile projects.

Although, the literature shows many works in story points effort estimation using machine learning and deep learning approaches. Few studies have employed natural language processing techniques to consider textual information related to software specifications, such as by specific terms in the title or description of user stories or the recurrence of words. Some of such important works are reported by [21]–[25]. Based on the intensive use of data, these studies have contrasted the performance of different machine learning algorithms such as Support Vector Machine (SVM), K-Nearest Neighbor (kNN), Logistic Model Tree, Naive Bayes, Decision Tree and Random Forest, which are among the best known and most frequently used techniques in data mining [26] and supervised machine learning [27]. Then, compared results to deep learning models such as Neural Networks [28]. The use of word embedding layers is highlighted in the studies by [24], [25], [29].

The most notable works on machine learning-based and NLP to story points estimation are briefly outlined with the techniques used.

Ionescu (2017) [29] proposed a machine learning-based approach using text from metrics and project management requirements as input, resulting in favourable outcomes. The authors created a custom vocabulary and investigated the use of word embeddings produced by a context-less method (Word2Vec). Afterward, aggregated with design attributes and textual metrics (modified TF-IDF), and generated numerical data to set a bag-of-words, using it as input to a linear regression algorithm.

Choetkiertikul et al. (2018) [24] proposed a deep learning-based prediction model with two neural networks combined: Long Short Term Memory (LSTM) and Recurrent Highway Network (RHN). Raw data of Story points estimation made by previous projects assists the model in learning how to perform story-point estimates for user stories (or issues). This approach generates context-less word embeddings to feed the LSTM layer. As input, the title and description were merged into a single instance, where the description follows the title. The embeddings word vectors served as input to the LSTM layer created a global document representation vector for the complete sentence. The global vector is then fed into the recurrent highway network for multiple transformations, which generates the last vector for each sentence. Finally, a simple regressor predicts predicting the effort estimation. The authors trained the embedding layer for all datasets in a previous process before being fed into the LSTM layer to avoid performance degradation. Their approach provided semantic features with the actual meaning of the issue's description, which showed the most outstanding results.

Marapelli's et al. (2020) [25] proposed a story point model estimation based on the combination of Recurrent Neural Network and Convolutional Neural Network. The user story text description is used as input to the model, then the story point estimation is predicted for that story. The authors considered the contextual word embedding of the user story to serve as input to the Bi-directional Long Short-Term Memory (BiLSTM). Which preserves the sequence data and makes CNN produce feature extraction accurately by multiple transformations [25]. The final vector representation is given as output by the CNN, and then a Neural Net Regressor is fed to predict the story point value for the input vector that represents the user story.

The work results demonstrated that the proposed RNN-CNN model outperforms Choetkiertikul's method on the Bamboo data set.

Apart from the methods described above, some other important and related works are as follows.

Porru et al. (2016) [21] proposed a machine learning classifier for estimating the story points required using the type and attributes of a given issue. Similar to Ionescu's [29] approach, the authors employ TF-IDF derived from summary and description as features to represent the issue. They choose a subset of features by univariate feature selection and had great results when estimating for new issues.

Scott et al. (2018) [23] built a prediction ML model using supervised learning, using as input the features derived from eight open-source projects from the same issue's dataset used by [21], [24], [25]. Besides textual descriptions from the issue reports, the authors also investigated developer features such as reputation and workload. In their experimental results, the models which employ developer features outperformed the models with only the feature extracted from the issue description.

Gultekin et al. (2020) demonstrate regression-based machine learning algorithms to estimate Story Points effort using the Scrum methodology. The authors calculate effort estimation for each issue where the total effort is measured with aggregate functions for iteration, phase and project. They perceived reasonable error rates by employing the algorithms Gradient Boosting, Support Vector Regression, Random Forest Regression and Multi-Layer Perceptron.

1.5 Contribution of this thesis

Given the foregoing, this thesis aims to contribute to the field of software effort estimation in Agile environments through ML techniques to estimate story points. Prediction models are trained from inputs containing historical story point estimations made by teams. In addition to other methods yet employed, the models should support agile teams when estimating the size of new issues or features in a user story. In this sense, we expose different machine learning and deep learning models trained to predict agile software estimation by raw data, from user stories text description, used as input to infer and give the story points estimation. Additionally, this work explores ways of taking advantage of natural language processing and the structure present in the dataset through pre-trained word embedding models to construct better features. Finally, we hope that this work helps future researchers and practitioners involved in agile environments.

1.6 Organization of this thesis

This thesis is organized as follows:

- Chapter 2 presents an overview of agile methodology, agile software effort estimation and the most prevalent methods used for agile environments found in the literature. Some related work on machine learning techniques approaches to story points estimation.
- Chapter 3 depicts our formulation of machine learning, deep learning and natural language processing approaches to predict story points estimations accurately.
- Chapter 4 introduces the story points dataset we employ and the necessary preprocessing steps to handle all textual information.
- Chapter 5 addresses the regression problem of story points estimation regarding the techniques as described in chapter 3.
- Chapter 6 summarizes our most significant findings and discusses the most promising paths toward enhancing the proposed model's prediction.

Chapter 2

Literature Review

Since machine learning and agile are fields relatively new and can involve different approaches and techniques, it is mandatory to analyze its outcomes by searching for different studies of implementation and literature reviews. Thus, in order to understand and elucidate the state of the art of, for this review, data between the relationship of software effort estimation, agile, machine learning, deep learning and natural language processing were identified by searches on the following databases: IEEE Xplore, Google Scholar, ACM Digital library, Science Direct, Spring and references from relevant articles using the search term: "Software effort estimation", "Agile", "Story point", "User Story", "Machine learning", "Deep Learning", "LSTM", "RNN", "NLP". The keywords in each component were linked using "OR" as a Boolean function, and the results of more than one section were combined by utilizing the "AND" Boolean in the final search. This review was also performed by e-books and websites from O'Reilly Media Inc., Scrum and Agile manifesto websites.

2.1 Agile methodology

Since agile methodology has emerged (2001), most software companies have been shifted to agile environments due to the need to speed the software development lifecycle (SLDC) and improve the quality of delivery for any size of project [30]. Then new methods and processes were created attending to shorter development cycles [31]. Schwaber and Beedle state that they emerged as alternatives to the traditional or waterfall models as most executives were not satisfied with their organization's ability to deliver systems at a reasonable cost and timeframes [32]. Agile methods "aim to remove or reduce much of the traditional project management formalism" [8]. Their processes seek to eliminate effort with unnecessary documentation, focusing on the interaction between people on the team and concentrating on activities that effectively will generate value in the final product or part of the product delivered (features) with the quality expected by customers [14], [31], [33]. This quality is achieved through iterative development cycles with short scopes, making it possible to adapt feature requirements in the development phase [32]. According to Cohn (2005), features are the unit of customer value [14]. Every agile iteration includes all the traditional software activities (planning, requirements analysis, design, coding, testing and documentation), but with a different focus: to deliver the features instead of complete tasks [4], [14]. The agile methods (especially Scrum) have shown a higher degree of success in projects due to flexible scope, frequent deliveries, and their ability to handle changing customer requirements [5], [8], [34].

The nature of the agile process causes concerns and challenges to manage agile projects in companies and projects of different sizes [30]. In contrast, a survey made by Jorgensen (2019) has shown that increased project size was associated with decreased project performance for both agile and non-agile environments. Moreover, the projects using agile methods had better results [8]. Jorgensen (2018) found that agile projects are only correlated with a higher proportion of successful projects if including frequent deliveries to production [35]. Rosa and collaborators (2021) stated that successful agile development projects in the United Department of Defense (DoD) "involves continuous planning, continuous testing, continuous integration, continuous feedback, and continuous evolution of the product" [4].

2.1.1 Agile manifesto

The agile methods follow the principles of the agile manifesto [10]. In 2001, experts got together to discuss ways to improve the performance of their software projects. After exchanging project experiences, they concluded that there was always a set of common principles that, when respected, projects worked well. Henceforward, the Agile Manifesto and its principles were established. The main principle of the agile practice is to deliver customer value and satisfaction as the first and highest priority, and by early and continuous software delivery of valuable software [10]. The stakeholders (customers or users) are constantly involved in the software development process to have requirements and priorities accordingly. Hence, it is prevalent having changes of requirements even in later stages of software development [18], [36].

The most discussed methods for agile environments are Scrum, Extreme Programming (XP), Feature-Driven Development, Dynamic System Development Method (DSDM), Crystal Methods, Lean Development (LD), and Adaptive Software Development [31], [37]. Although methodologies are several, Scrum is the most widespread among practitioners, having variants and common associations with other agile methods such as Kanban [13], [38], known as Scrumban [39].

2.1.2 Scrum

Scrum is an agile methodology created in 1993 by Jeff Sutherland and Ken Schwaberem and later collaborations with Mike Beedle. In the literature, Scrum is often cited as a framework in which various processes and techniques can be employed [40].They based the framework on the article ”The new product development game” by Hirotaka Takeuchi and Ikujiro Nonaka published in 1986 in the Harvard Business Review, where the authors compared product creation processes to sports [39]. Scrum name is given by the concept of scrum from the rugby game, where the team moves forward as a unit.

> “The traditional sequential or “relay race” approach to product development [. . .] may conflict with the goals of maximum speed and flexibility. Instead, a holistic or “rugby” approach — where a team tries to go the distance as a unit, passing the ball back and forth — may better serve today’s competitive requirements.” Takeuchi and Nonaka (1986) [41], “The New New Product Development Game”, Harvard Business Review

Scrum and its variants are present in many contexts, from small, medium and large companies to government agencies [8], [30]. Such companies are often present in many studies such as, ITTI [30], Department of Defense of USA [4], BMC [42], SAP [43] and Facebook [44].

The Annual State of Agile (2020) [13] states Scrum as the most broadly practiced Agile framework, with at least 76% practicing Scrum or its variants and hybrid versions. This survey provide annually a global report regarding Agile methodologies across a range of different industries worldwide. Over 14 years, this report is the longest-running and most widely cited Agile survey [30]. Schwaber and Sutherland (2020) state that Scrum is founded on empiricism and lean thinking. Empiricism states that knowledge comes from experience and from making decisions based on what is known. At the same time, lean thinking reduces waste and focuses on what is vital. Scrum form three empirical pillars:

(i) Transparency - Meaningful regards of the process must be noticeable to those qualified for the deliverables; (ii) Inspection - Scrum team must frequently inspect artifacts and progress toward detecting unwanted variations or issues. Inspection facilitates adaptation. It provides events designed to promote change; (iii).Adaptation - If any character of the process has deviated from acceptable limits or the deliverable is unacceptable, the process must be adapted to achieve expectation. Whereas it must be adjusted as soon as possible to minimize deviations.

Among the agile methods, Scrum best defines set of roles, artifacts, and events [45], which are listed below.

1) Roles: Product Owner, Scrum Master, Developer. 2) Artifacts: Product Backlog, Sprint Backlog, Increment. 3) Sprint Planning, Sprint, Daily Meeting or Scrum Daily, Sprint Review, Sprint Retrospective.

The Scrum Team is a unit formed by one Scrum Master, one Product Owner, and Developers. Typically, a small team of people, usually ten or fewer. Although they have different roles, all members work together focused on the same product goal [40]. Table 2.1 better describes Scrum's roles and responsibilities.

Role	**Responsibility**
Scrum Master	Ensure all events are productive and following the values and rules. Coach the team to meet high-value increments. Remove all possible blockers to keep the work team in progress within the timebox.
Product Owner	Manage and prioritize the activities defined in the Product Backlog. It must ensure all items are transparent, visible and understandable.
Developers	Build any aspect of a functional product increment every sprint or interaction.

Table 2.1: **Scrum roles and responsibility**

Essentially, the goal of Scrum was to change the way software development was managed and deliver more excellent business value in the shortest time, driven through short intervals, called sprints. The Sprints have the length of one month or less and are considered the core of Scrum, where all events are in place to enable the transparency required. According to Schwaber and Beedle (2001), the Scrum practices for Agile Software Development are: (i) Product Backlog, (ii) Scrum Teams (iii) Daily Scrum Meetings, (iv) Sprint Planning Meeting, (v) Sprint and (vi) Sprint Review [32]. In 2010, Schwaber and Sutherland developed the first version of the Scrum Guide to help worldwide to understand and apply these framework practices. Since then, various small and functional updates have been in place to increase quality, and effectiveness [40]. In a nutshell, awareness and planning details are concentrated in the current sprint. In the next sprints, planning is superficial and constantly changes as the project goes and the team (including stakeholders) is acquiring more knowledge about the project. Each feature has functionality description written from the users' point of view (user story), and it is considered as goal-driven value to achieve customer satisfaction[18], [46]. Product Owner, Scrum Master, and Developers meet to estimate the effort required for each item in the Product Backlog and validate that the current user stories descriptions are adequate. The Product Backlog includes a list of all recognized user stories, then prioritizes and split into releases. Normally, it breaks down the project into 30-day Sprint cycles, each containing a set of backlog features (Sprint Backlog). That said, each Sprint is based on business priority and intends to deliver first the most important feature of software [18]. The development team executes the project based on short iterations (sprints), where all work is split and scheduled [14], [34]. During the sprints, the team catch-up with daily 15-minute meetings (Daily Scrum) to review the status and organize tasks accordingly [31].

When the Sprint ends, the results are discussed and presented to the Product Owner at the Sprint Review Meeting. Afterward, the Scrum Master conducts the Sprint Retrospective Meeting to identify the most notable changes to improve its effectiveness for the next Sprint. Then, Sprint Retrospective closes the Sprint.

The steps described above repeats for every Sprint and accumulates until the product release is ready.The dynamic nature of agile turns its application into a challenging task. Moreover, difficult to be predicted in terms of budget and cost, which consequently, makes accurate cost and effort estimation trivial resources [5]. To address this issue, user story, planning poker and story points are often related [3].

2.1.3 User Story

Cohn (2004) defined a user story as simple, straightforward, and brief descriptions of functionality valuable to real users. Also, Cohn stated that write software requirements in the form of user stories are the best way to satisfy user's needs in agile environments [47]. The literature shows that user stories are the most popular accepted method in agile software development [48] to specify requirements from the user's point of view [49].

Typically, the stories are written on story cards for easy viewing and handling. For this reason, the structure should be simple, as stated by Cohn. The focus on describing what the user says is because the end-user should receive what he needs and not precisely what he wants. Based on short or broken stories, the agile development meets the objectives in the best possible way as described by the user.

The size of the story must correspond with the team and the technologies involved. Every user story should be estimable and small, but not too small. Very small or large stories are difficult to estimate. Large ones should be broken into smaller parts [47]. Cohn (2004) cites that developers must estimate or at least understand the size of a story or the time it will take to turn it into code.

As in Scrum, after gathering user stories or story cards according to their priorities, the agile team performs the effort estimation. At the end of the Sprint, acceptance tests confirm whether each delivered user story met the requirements or not [14].

The most significant advantage of using user stories is that it can be done by any team member, without the need for profound knowledge in requirements gathering, such as the Use Case [50]. However, it is generally used together within the team, as agile teams are multidisciplinary [32].

2.2 Agile Software Effort Estimation

Agile Software Effort Estimation (ASEE) is the process of predicting the correct effort required to build and deliver software earlier as expected by customers in agile environments [14]. As the traditional software effort estimation, agile software effort estimation is mainly related to time and budget constraints in a project and represents a vital role directly impacting the quality of software delivery [3].

The literature shows that ASEE has been demonstrating higher accuracy when compared to traditional or waterfall models [7], [12]. In the traditional model, the limited scope and sequential form of processes cause the cost of a change to be exponential, tending to grow as the development process goes. If the estimated cost and effort are not accurate, it may result in project failure in budget and delivery time [5]. Cohn (2005) recorded estimates as the basis of the project management processes of planning and risk analysis. Planning unfolds steps of a strategy to achieve business goals, based on estimates [14].

"Estimating and planning are critical to the success of any software development project of any size or consequence"
Mike Cohn (2005) [14].

However, estimating and planning are complex and error-prone. In 1998, Steve McConnell called a cone of uncertainty the first project schedule estimate designed by Barry Boehm (1981). By this graphic Boehm's showed initial ranges of uncertainty at different points in a waterfall development process. The cone of uncertainty predicts that initial estimates of a project can vary from 60% to 160%, i.e., the estimate predicted to occur in 20 weeks may take anywhere from 12 or 32 weeks. We notice that the estimates tend to vary a lot at the beginning of the project, but that over time will stabilize when we stop estimating and become a certainty [14]. McConnell (2006), mentions that uncertainty occurs due to bad decision-making [51].

As part of 12 principles of the agile manifesto, [10], for most robust architecture, design and product requirements appear and mitigate uncertainty through cycles of iterative developments and constant feedback [52].

The figure 2.1 contrast the planning iteration between agile and waterfall methodology. The agile effort is part of the planning phase (adaptive planning), and it is welcoming to changes as the project goes. On the other hand, traditional planning (waterfall-based), all stages of development: analysis, design, development, testing and delivery are sequential [52], [53].

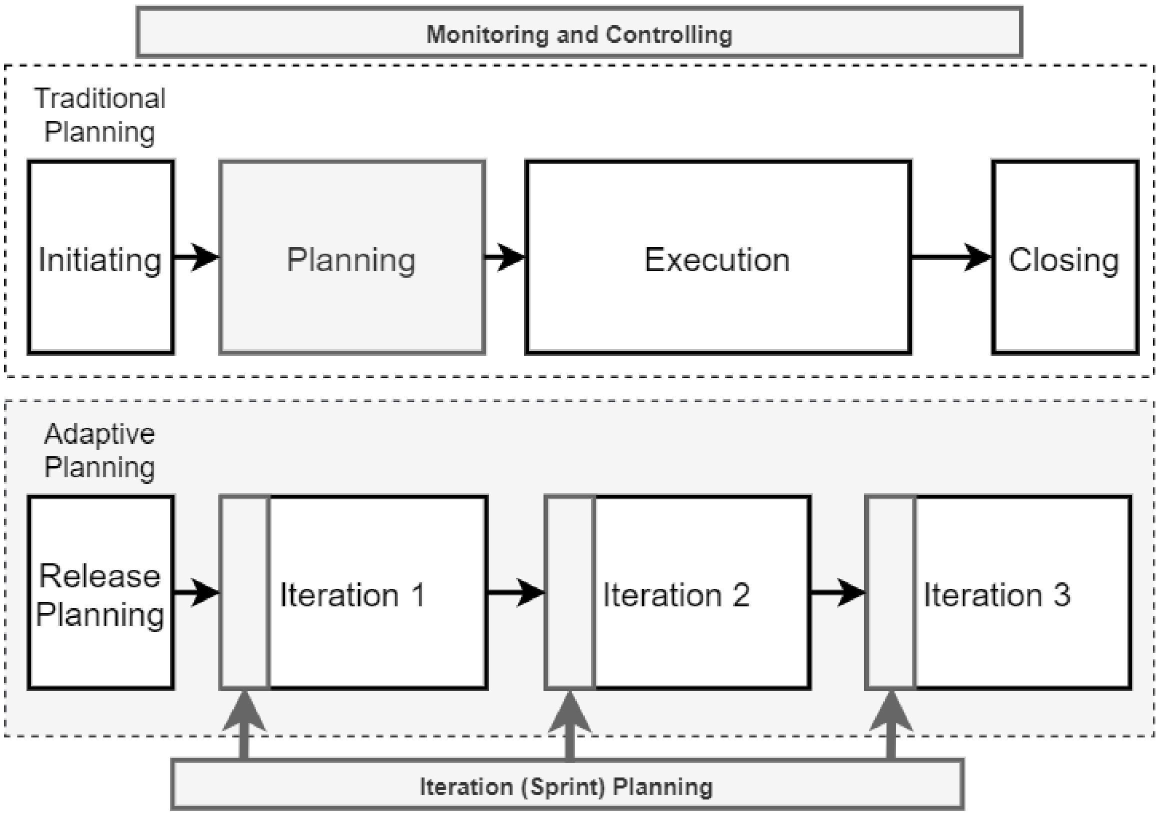


Figure 2.1: Traditional and adaptive planning phases - Source: Adapted from Boral (2016) [52]

In agile methodologies (i.e. Scrum), at the end of each Sprint (usually two weeks), besides having part of the software or feature ready, the historical information will fit the plan for the next Sprint, thus adjusting the other estimates throughout the project [40]. Agile estimation aims to estimate the cost for all items in the backlog. In addition to creating roadmaps, this cost can be used to measure the team's velocity and help in decision-making when prioritizing resources, for example.

According to Cohn (2008), the first step is to define the satisfaction conditions and identify the success or failure criteria. Secondly, the team is responsible for estimating the user stories in the chosen unit (i.e. story points) for two or three releases. Afterward, the steps can follow any sequence: select an iteration length, estimate velocity and prioritize user stories. Finally, the process outputs user stories and the release date. Although, there are different measures and methods to estimate in an agile environment. For Cohn (2008), the team needs to be in consensus about the unit of measure and consequently the estimates of the Product Backlog items.

According to Mike Cohn "estimates are best derived collaboratively by the team, which includes those who will do the work" [14]. As estimating is a collaborative task and team members with different experiences (and skills) should interpret the complexity of a given user story [16], [18], the definition of time and effort can be different for each developer.

Jorgensen (2020) examined the relations between low effort estimates, other commonly used skill indicators, and measured programming skills. The author found that those with the lowest programming skill gave the lowest and most over-optimistic effort estimates for the larger tasks. For the smaller tasks, however, those with the lowest programming skill had the highest and most over-pessimistic estimates [54]. Cohn (2008) and Grenning (2002) recommend using points in agile estimating to mitigate this scenario.

2.2.1 Types of Agile Software Effort Estimation

There is no universally accepted classification for agile software estimation techniques. The techniques can be classified differently depending on the characteristics. In 2017, Usman et al. observed that methods and techniques had not yet been organized. For that reason, the authors proposed another study regarding a taxonomy of effort estimation with a classification scheme to discriminate estimation activities in agile environments. The authors classified the estimation techniques as algorithmic, expert-based and artificial intelligence-based techniques and models [55]. Lately, some researchers have suggested machine learning to be the third major category, as Vyas et al. (2018) [56], Dantas et al. (2018) [57]. Vyas and collaborators classified agile estimation techniques as non-algorithmic, algorithmic and machine learning.

Agile teams can also combine techniques to perform estimates methods. The literature shows these techniques categorized as combination-based methods [3].

The latest SLR made by Fernandez-Diego et al. on effort estimation in agile software development organized the agile estimation methods as follow.

- Expert-based: Planning Poker, Expert Judgment, Wideband Delphi
- Data-based: machine learning, neural network, functional size measurement, regression, algorithmic methods, fuzzy logic, swarm intelligence, Bayesian network, Monte Carlo, statistical combination, principal component analysis, COCOMO II.
- Combination-based: Use Case Point, Change Effort Prediction, Ontology Model, Experience Factory, Prioritization of Stories.

Overall, Usman et al. (2014) [12] found that the most applied estimation technique in agile environments were expert-based subjective methods. In addition to expert estimation, Bilgaiyan et al. (2017) [5] also found that neural networks are the most popular of the current conventional estimation methods.

The systematic literature review carried out by Fernandez-Diego (2020) and collaborators, which updates the SLR published in 2014 by Usman et al. [12], investigated works from 2014 to 2020 by analyzing the data extracted from 73 papers. The authors concluded that the expert-based estimation methods are still the most relevant (Planning Poker 24.66%, Expert Judgment 10.96%, Wideband Delphi 5.48%). Moreover, Planning Poker is very frequently related to story points and used in combination with other methods such as machine learning, artificial neural networks (deep learning) based estimation.

2.2.2 Planning Poker

Planning Poker is a popular card game made for agile estimating effort. First introduced by James Grenning in 2002 [58] and later popularized by Mike Cohn [14] through his book named "Agile Estimating and Planning".

Usman et al. [12] remark that Planning Poker is a combination of elements of Expert Opinion, Analogy and Disaggregation. The Planning Poker should have a scale, and agile teams commonly use the Fibonacci sequence. Despite some teams also use a similar version based on Fibonacci, Cohn (2005) remarks the Fibonacci sequence as the best scale as "they reflect the greater uncertainty associated with estimates for larger units of work" [14]. In short, each development team member receives a set of cards with the values of a specific sequence, which determines the estimate for the phases of the Product Backlog during the Sprint planning meeting. Each number in the sequence corresponds to a card. Then each team member estimates each user story using the cards, which particularly represent story points. Meanwhile, the Product Owner brings an overview of the user story, and the Scrum Master guides the Planning Poker to ensure the process accordingly.

Many studies state that Planning Poker is the best agile method to estimate story points [21], [22], [43].

"The best way I have found for agile teams to estimate is by playing planning poker" James Grenning (2002) [58]

Gandomani et al. (2014) compared the Planning Poker and the Wideband Delphi technique and concluded that Planning Poker gave better estimation accuracy than expert estimation and Delphi technique [56].

2.2.3 Story points

The usage of Story Points to measure effort is not new in Agile teams and started after the industry adopted expressing requirements as a User Story [15].

As cited above in Planning Poker, the main measures to represent story points are numeric sizing, Fibonacci, or T-shirt sizes. However, the most scale sizing technique used for estimating Story Points is the Fibonacci sequence because it reproduces a more realistic estimation for complex tasks [14].

Typically, story points estimation happens during the backlog refinement sessions, where the team evaluates the Product Backlog. The team is responsible for setting the product backlog items to read customer value. In order to estimate user stories, the team has to find one or two baseline stories as a reference, which needs to be understandable by every member team. Once defined, the team estimates the other user stories by comparing them to the reference user story. Thus, the team gives the reference story and a number from the Fibonacci sequence, and then they will take the other user story one by one and ask: does this user story requires more effort than the reference story? If positive, they sort the user story after. If negative (less effort than the reference story), then sort it before. Hence, the team can proceed and give story points to the other user stories. For instance, if a given user story requires twice as much effort as the reference user story, it receives double story points. If it is twice less, then it is less following the sequence. During the estimating session, it is crucial to involve everyone in the team: developers, designers and testers because each team member has a different perspective from the product and the work required to deliver a user story.

Story point is the straightforward size metric for agile environments, despite being found in combination with other metrics [59]. There has been a decrease in general-purpose size metrics and an increase in size metrics that take the particularities of agile methods into account. Consequently, most studies rely on user stories to specify requirements as related by Raharjana et al. (2021) [49] through a systematic literature review made against 718 papers published between January 2009 to December 2020.

The literature shows that the estimation models using Story Points [59]–[63] are the most employed metric when compared to that of the other metrics.

Story points are not unique in estimating points; agile teams also use (less prevalent) methods in which the output are points such as Functional Points (FP), Use Case Points (UCP), Object Points, Lines of Code (LOC). Whereas, Story Points have been proved more accurate results than the others [64].

As by Usman et al. (2014), Datas et al. (2018) [57], Malgonde et al. (2019) [59], Story Points and UCP were the most frequently used size metrics, while conventional metrics like FP or LOC were unusual in agile. Notwithstanding, Diego and collaborators (2020) investigated works from 2014 to 2020 and states that Story Points play a higher responsibility as the widespread estimation technique under agile methods such as User Story and Planning Poker. This scenario is also highlighted by Fernandez-Diego et al. (2020) [3], Story Points, Planning Poker and User Story are frequently connected.

2.2.4 Machine Learning Approach for Story Points Estimation

The first machine learning techniques applied to solve the problem of software effort estimation started in the 1990s [2]. In general, previous historical data of projects estimations serve as a reference to predict a new software effort estimation [65].

Story points have become the main size metric for agile environments and express a notable increase in the literature following the use of machine learning-based methods to produce estimations.

A Systematic Literature Review (SLR) published in 2014 by Usman and collaborators [12] reviewed works from 2001 to 2014, presenting a noteworthy state of the art guide on agile effort estimates. Some years after, Dantas et al. (2018) [57], and Diego et al. (2020) [3] updated this work and evidenced a strong tendency of agile effort estimation models based on machine learning techniques.

Ungan et al.[66] provided a comparison of Story Points and Planning Poker estimation with effort estimation models based on COSMIC Function Points (CFP). By employing diverse sets of regression analyses (simple, multiple, polynomial, power, exponential and logarithmic regressions) and Artificial Neural Networks (ANN) to build the models, the authors observed that Story Points-based estimation models worked more favourably than based models in standard analysis techniques. In comparison, ANN and multiple regression showed the best results, showing accuracy increases on the regression models. Hamouda (2014) [67] proposed a process and methodology which embraces the software size relatively using Story Points. This approach was applied on different projects of level three Capability Maturity Model Integration (CMMI).

Satapathy, Panda and Rath (2014) [68] propose optimal Story Points estimation through various Support Vector Regression (SVR) kernel methods. The authors compared results obtained from all methods, where the SVR Radial Basis Function Neural Networks (RBF) model gives a lower error rate and higher prediction accuracy value.

Moharreri et al. (2016) [22] propose an automated estimation method for agile story cards. Some models were built and contrasted with the manual Planning Poker method. The authors proved that the estimation accuracy increases with the help of supervised learning models.

Rao et al. (2018) [64] evaluated time and cost of software estimation by taking as input Story Points and project velocity. The author showed a performance comparison of three machine learning techniques: Adaptive Neuro-Fuzzy Modeling, Generalized Regression Neural Network, and Radial Basis Function Networks.

Malgonde et al. (2019) [59] developed an ensemble-based model to predict story effort estimation and compared the results. The model outputs were compared to other ensemble-based models and various predictive models such as Bayesian Networks, Ridge Regression, Neural Networks, SVM, Decision Trees and kNN. The authors demonstrated better performance of their approach to optimize Sprint Planning in Scrum.

Souza et al. (2019) [69] propose the use of a Fuzzy Neural Network which is compared with models commonly used in the literature, such as kNN Regression, Independent Component Regression, ANN with a Principal Component Step and Multilayer Perceptrons.

Chapter 3

Deep Learning for Story Points Estimation

Accurate estimation of Agile software development effort using machine learning or deep learning depends on the methods employed, which can contribute favourably or negatively. This thesis aims to distinguish these particular impacts. Figure 3.1 illustrated the overall architecture of the methodology approach, which depicts the proposed methods. The main motivation of the approach for this thesis takes as reference the other works mentioned in 1.4 within the estimation of effort in story points from text requirements, making use of pre-trained embedding models. As shown by the results obtained by the most notable works mentioned before, mainly Ionescu2017, Choetkiertikul et al. (2018) and Marapelli (2020), the deep learning architecture coupled with the use of word embedding models are promising and still unexplored gap to best of our knowledge.

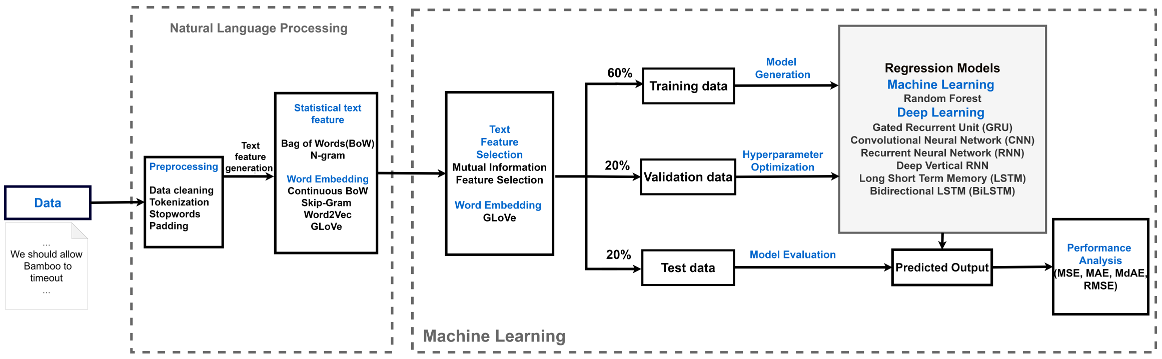


Figure 3.1: Architecture of methodology approach

3.1 Text Preprocessing

Language modelling is one of the most fundamental tasks for ML projects. The text preprocessing step aims to filter and transform raw data coming from the dataset to make it suitable for ML and the problem scenario. In this sense and for this thesis, we performed several steps for preprocessing, taking into account the agile software effort estimation scenario using Story Points. For more details regarding the techniques, refer to 4.3.

3.2 Text Feature Extraction

Natural Language Processing (NLP) is a branch of Artificial Intelligence (AI) that aims to understand and produce information related to the language of humans. While Deep Learning is suitable to extract features from audios (spectrograms) or images (pixels), DL models are also helpful to identify patterns in text from its elements (e.g. words and characters). NLP fuses modelling of human language with statistical, machine learning, and deep learning models to extract features, process and interpret the context of text or voice data. As machine learning models are not compatible to understand text directly, the text features should translate the information to numeric values (word vector representations). This section breaks down the text features techniques.

3.2.1 N-gram

N-gram model is a popular feature identification and analysis in NLP tasks. It is a sequence of words contained in a sentence. Through statistical inference, it tries to predict the next word in a sentence from the previous words.

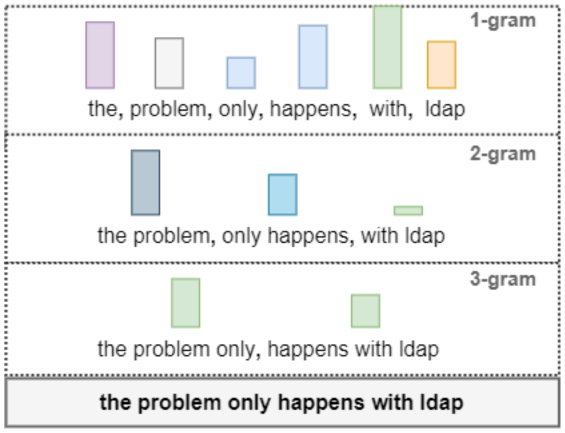

1-gram
the, problem, only, happens, with, ldap
2-gram
the problem, only happens, with ldap
3-gram
the problem only, happens with ldap
the problem only happens with ldap

Figure 3.2: Structure of 1-gram, 2-gram and 3-gram

A sequence of words can occur in a recognized way but befall with an unfamiliar word. By grouping sequences of size n that start with the same (n - 1) words into an equivalence class assumes that the previous local context affects the next word and builds the n-gram model, where the last word in the n-gram is the prediction. Figure 3.2 illustrate the common types of n-gram models. Particularly, n-gram models are called 1-gram, 2-grams, 3-grams and 4-grams, where n = 2, 3 and 4, respectively. The number of classes that divide the data grows as the value of n increases, then better is the inference. The formulas for 1-gram, 2-gram and 3-gram are represented by 3.1 respectively.

$$\begin{aligned} P(x_1, x_2, x_3, x_4) &= P(x_1)P(x_2)P(x_3)P(x_4), \\ P(x_1, x_2, x_3, x_4) &= P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2)P(x_4 \mid x_3), \\ P(x_1, x_2, x_3, x_4) &= P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2)P(x_4 \mid x_2, x_3). \end{aligned} \tag{3.1}$$

3.2.2 Bag of Words (BoW)

The Bag-of-words model is a text feature extraction technique that aggregates words and calculates their term frequency to measure their relevance. It is an oversimplified numeric form (word vectors) of text representation and does not consider the sentence's order. The text classification is the most widespread use of bag-of-words.

3.2.3 Word embeddings

The word embeddings method is a definite trend to extract the semantics of words in a specific context. They are neural networks-based methods that yield dense and low dimensional word vector representation, assisting machine learning algorithms with textual properties [49]. Against this backdrop, the word embedding methods such as Word2Vec [70], GloVe [71], and BERT [72] have emerged to learn from large corpus datasets as pre-trained models. These methods enable to solving vast kinds of problems. Table 3.1 shows the pre-trained word embedding model used for this thesis experiment.

Method	Corpus	Content
GloVe	Wikipedia 2014 + Gigaword 5	400,000 word vectors based on 6 billion tokens

Table 3.1: Pre-trained word embedding

Figure 3.3 demonstrates the neural network architecture to generate word embed-

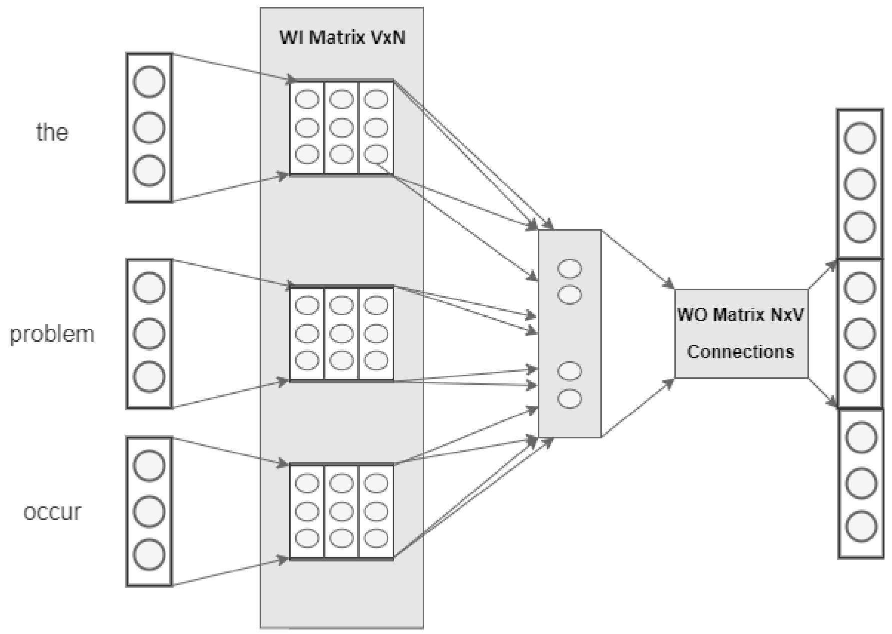


Figure 3.3: Input and output of word embeddings generation - Source: Adapted from Patihullah (2019) [73]

dings. Wherein WI matrix with VxN Connection, V is the vocabulary size, and N is the dimension of word vectors. Next, the WO matrix with NxV Connection takes the hidden layer WI matrix as input and generates the WO representing a word from the given vocabulary.

Word2Vec

Word2Vec is an unsupervised learning algorithm widely used for word embeddings that provides a way to locate vector representations of words and sentences. It holds two contrasting training approaches and somehow oppositely (explained below): Skip-Gram and Continuous Bag-of-Words (CBOW). In both cases, a separate weight matrix forms the model aside from the word embeddings, achieving a speedy log-linear training which can catch semantic information [70]. Typically trains word embeddings fast, and these trained models (pre-trained word embeddings) are employed to initialize the embeddings of some further complex models like deep learning models.

Continuous Bag-of-Words (CBOW) and Skip-Gram

A Continuous Bag-of-Words (CBOW) model tries to predict center words given the context around the target word (few words before and a few words after). Oppositely of Skip-Gram, CBOW is not sequential. The Skip-gram model is a variant of bag-of-words that gathers n-grams, but it permits word skipping. The model tries to predict the context words from the central word [70]. Thus, by having a group of words, they no longer require a continuous process, and we can skip words to generate Skip-Grams.

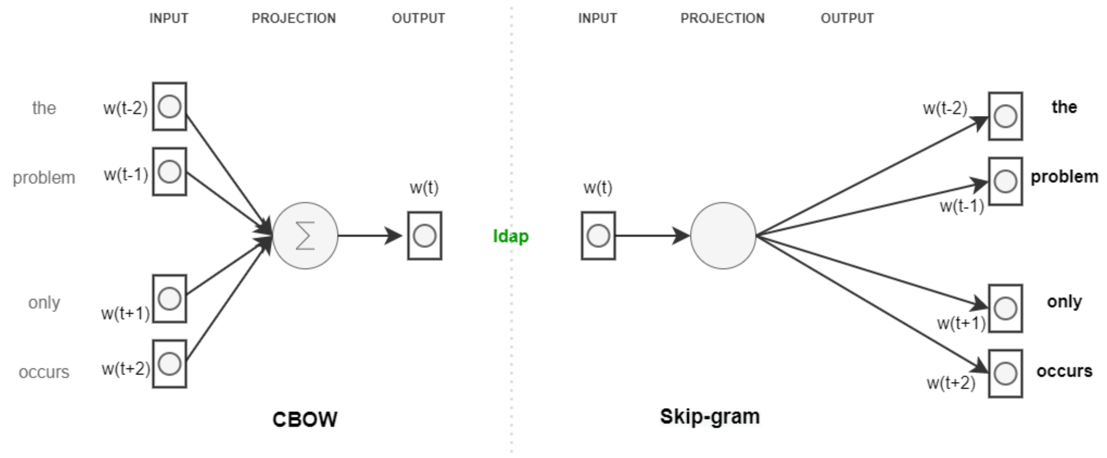


Figure 3.4: Graphical representation of the CBOW model and Skip-gram model - Source: Adapted from Mikolov (2013) [74]

As shown in Figure 3.4 CBOW learns the context, and then the model outputs the most probable word is ”ldap”. On the other hand, Skip-gram recognizes the word ”ldap” and tries to predict the context as ”the”, ”problem”, ”only”, ”occurs” or some additional related context. Mathematically, continuous bag-of-words model can be represented by 3.2, while Skip-gram model is denoted by 3.3.

$$P(w_c \mid \mathcal{W}_o) = \frac{\exp\left(\mathbf{u}_c^\top \bar{\mathbf{v}}_o\right)}{\sum_{i\in\mathcal{V}} \exp\left(\mathbf{u}_i^\top \bar{\mathbf{v}}_o\right)}. \tag{3.2}$$

$$q_{ij} = \frac{\exp(\mathbf{u}_j^\top \mathbf{v}_i)}{\sum_{k\in\mathcal{V}} \exp(\mathbf{u}_k^\top \mathbf{v}_i)}, \tag{3.3}$$

GLoVe

GLoVe is a contemporary word embedding method proposed by Pennington et al. (2014) for obtaining vector representations of words. This method shows enhancements based on matrix-factorization-based methods and the Skip-gram model. GloVe trains the model by a global matrix of word-to-word co-occurrence counts, and local context window [71] (also used in the Skip-Gram and CBOW model). Co-occurrence is the instance of two words appearing in a particular position alongside and counts all documents in the corpus. When training, GloVe reach the loss function 3.4 by calculating the squared error of 3.5 with weights.

$$\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{V}} h(x_{ij})\left(\mathbf{u}_j^\top \mathbf{v}_i + b_i + c_j - \log\, x_{ij}\right)^2. \tag{3.4}$$

$$\mathbf{u}_j^\top \mathbf{v}_i + b_i + c_j \approx \log\, x_{ij}. \tag{3.5}$$

In the original paper [71] the authors showed that GloVe outperformed Word2Vec on the task of word analogy.

3.3 Text Feature Selection

Usually, after the preprocessing step in machine learning projects, different feature selection methods are applied. This thesis focuses on the execution of deep learning models, which can already perform the functions of feature extraction and selection. However, we decided to execute this step as it can help to decrease the overfitting of the models, reduce training time, and improve model accuracy.

As long as this work is about a regression problem, the feature selection was performed based on the Mutual Information Feature Selection (MIFS) technique. The MIFS estimates mutual information for a continuous target variable and can reduce the dimensionality of the dataset. The remarkable properties for this analysis are as follow:

- MIFS is symmetric: $I(X, Y) = I(Y, X)$
- MIFS is non-negative: $I(X, Y) \geq 0$

MIFS regards $I(X, Y) = 0$ if and only if X and Y are independent. Conversely, if X is an invertible function of Y , then Y and X experience all information: $I(X, Y) = H(Y) = H(X)$.

By extending the interpretations of these terms and connect them, MIFS employ the following algebra:

$$I(X, Y) = E_x E_y \left\{ p_{X,Y}(x, y) \log \frac{p_{X,Y}(x,y)}{p_X(x)p_Y(y)} \right\}.$$

3.4 ML and Neural Networks

This section briefly describes the concepts of Machine Learning and Deep Neural Networks models regarding regression problems, especially LSTM and its variants, mostly explored in this thesis.

3.4.1 Supervised Learning

Supervised learning is the most common form of machine learning with two classes of possible algorithms: classification and regression [75]. If discrete class labels, then classification, if continuous values, regression. Essentially, the term supervised learning originates from the perspective of an instructor providing examples of the target and teaching the machine learning system the necessary steps to solve a problem [76]. Generally speaking, supervised learning aims to induce a mapping from x-vectors to y-values to build a hypothesis that allows predicting y-values for unlabeled samples. Thus, by a known dataset, data inputs x (features) and outputs y are presented to the algorithm, formed by pre-labelled examples (correct answers), target data input (training dataset) [77]. The model uses the training dataset to learn how to perform the task (T) and predict the answer (y) from an unknown dataset or no-labelled data (testing dataset). Accordingly, task T is to learn the mapping function f from input variables $x \in X$ to outputs $y \in Y$, which is also called prediction function $y = f(x)$ [75]. Figure 3.5 illustrates the supervised learning flowchart.

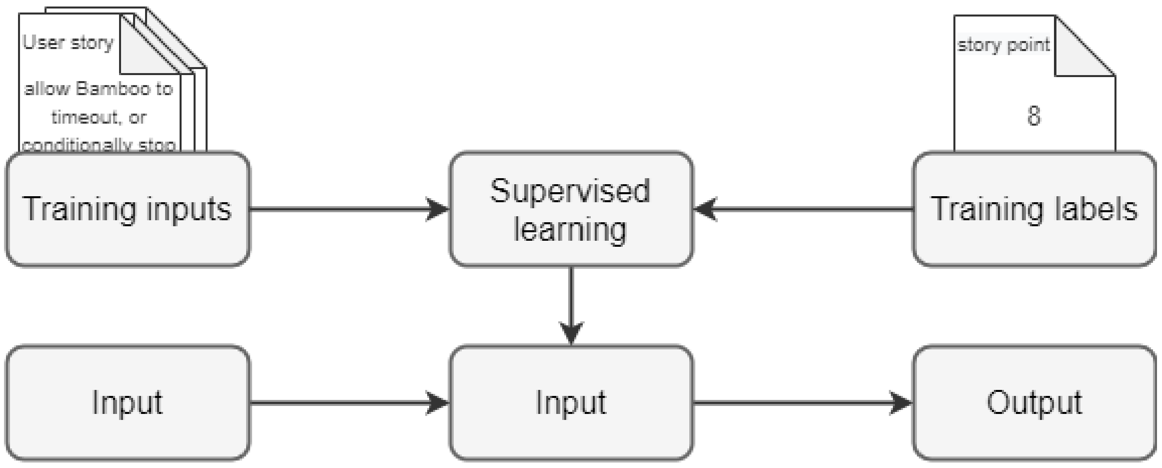


Figure 3.5: Supervised learning - Source: Adapted from Zhang (2021)[78]

3.4.2 Semi-supervised learning

In semi-supervised learning, the algorithm receives a small set of labelled examples and a more extensive set of unlabeled examples. This case aims to observe both sets of examples to find a hypothesis that can predict new observations among the existing classes.

3.4.3 Unsupervised Learning

In unsupervised learning, there are no labelled training examples. The algorithm receives a set of unlabeled training examples, which are analyzed and correlated to built clusters. Unsupervised learning strives to discover similarities or deviations in the set of attribute values of the examples that allow clustering. Thus, clustered examples can be assigned to the same class, while scattered examples are more likely to belong to different classes. This type of ML technique is the case of word embeddings described in 3.2.

3.4.4 Linear regression models

Linear regression is a supervised learning algorithm where a continuous range is used to make predictions. It can be divided into two main categories: simple regression and multivariable regression. In simple regression, the linear relationship exists if the data approximate a straight line. It has a slope-intercept form and can be represented in the formula: $y = mx + b$, where m and b are the variables to be learned by the algorithm. The input data is represented by x, while y represents the prediction. However, when it is required to investigate extra parameters, multiple linear regression takes place. With more complexity, a multivariable linear equation can be summarized as $f(x, y, z) = w_1x + w_2y + w_3z$. Whereas w represents the weights and x, y, z the attributes for each observation. The model remains linear as its yields are a linear combination of the input variables.

Cost Function

ML algorithms employ a loss or cost function to measure the distance (error) from what the model encounters when predicting by current parameters (i.e. a set of weights). The most common way to find the best set of weights in regression problems is by employing the mean squared error (MSE) 3.6, which calculates the average squared difference between the observation's actual and the values predicted by the model. This is also the function we employ for all algorithms in the experiments.

$$MSE = \frac{1}{N} \sum_{i=1}^{n} (y_i - (mx_i + b))^2 \tag{3.6}$$

3.4.5 Random Forest model

The Random forest (RF) [79] is an algorithm proposed to solve performance issues related to the decision tree algorithm [80]. RFs employ an ensemble of tree predictors, where each tree depends on random values formed by two methods: bagging [81] and random selection [79]. Both methods concurrently applied with a higher number of trees improve the prediction of the model. In addition, it makes the model faster and more resistant to overfitting. In regression, the average of prediction outcomes acquired from single trees signifies the forest's prediction accuracy [82]. Figure 3.6 represents a RF model architecture. Ensemble methods like RFs have proven more accurate results for software effort estimation [19], [24], [83]. That said, we have chosen the RF algorithm for comparison with Deep Neural Networks.

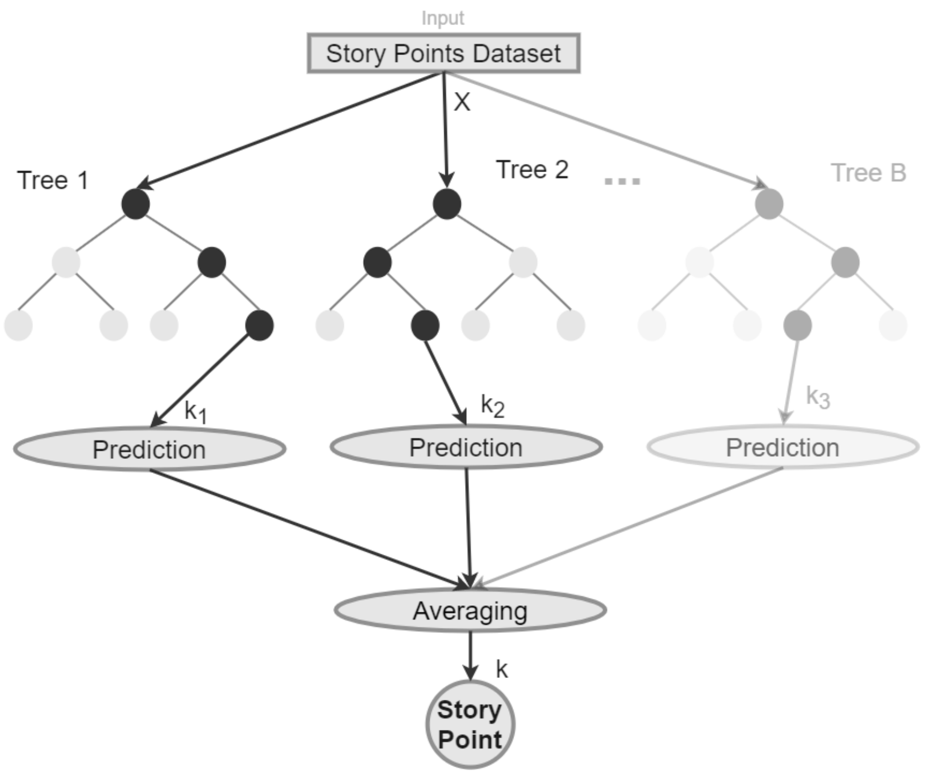


Figure 3.6: Random Forest architecture - Source: Adapted from [84]

3.4.6 Deep Learning and Neural Networks

Deep Learning (DL) is a subset of machine learning based on artificial neural networks (ANN), also known as Deep Neural Networks (DNN) [85], but with two or more hidden layers. Unlike classic ANNs, the DNN architecture can have many hidden layers and non-linear activation functions to solve more complex problems. The additional hidden layers assist in extensive calculations and weights and biases adjustments for optimization. Each layer is typically an algorithm containing one type of activation function. The first layer of the Deep Neural Network is the input layer, which passes the information to multiple hidden layers. The last layer is the output, where the result of the regression or classification is given. There are no connections between neurons in the same layer.

3.4.7 Recurrent Neural Networks (RNN)

Unlike conventional neural networks, Recurrent Neural Networks preserve the sequence of the data inputs. In addition to the data received from the previous layer, the hidden neurons also receive the current computation results (done by themselves). Figure 3.7 illustrates the RNN cell with hidden states. The network can memorize short-term sequence information. However, the gradients might disappear over time, which leads to loss of temporal data relation that the network must learn to solve sequential problems [86]. This problem is a common issue known as vanishing gradient [87]. In natural language processing it refers to the loss of relevant context information necessary to identify the textual semantics of given words or phrases.

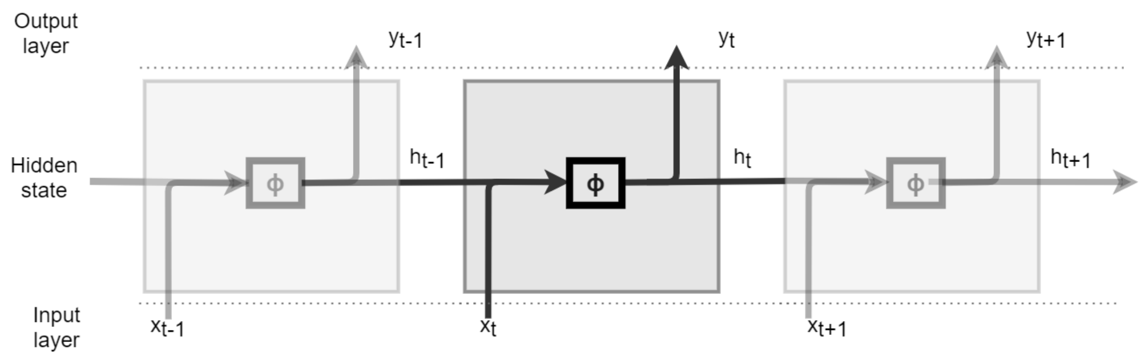


Figure 3.7: RNN Cell - Source: Adapted from Zhang et al. (2021) [78]

Besides, there are various forms to build RNNs, the hidden state form represented by the equation 3.7 is the most used. Where $\mathbf{w}_{xh} \in \mathbb{R}^{d\times h}$, $\mathbf{w}_{hh} \in \mathbb{R}^{h\times h}$ are the weight parameters and the bias $\mathbf{b}_h \in \mathbb{R}^{1\times h}$ of the hidden layer. Followed by the weights $\mathbf{W}_{hq} \in \mathbb{R}^{h\times q}$ and the bias $\mathbf{b}_q \in \mathbb{R}^{1\times q}$ of the output layer [78].

$$\mathbf{H}_t = \phi(\mathbf{X}_t\mathbf{W}_{xh} + \mathbf{H}_{t-1}\mathbf{W}_{hh} + \mathbf{b}_h). \tag{3.7}$$

Stacked RNN (sRNN)

The Stacked RNN (Figure 3.8) is a variation of RNN stretched deeper by stacking multiple recurrent hidden layers on top of each other. Designed to boost recurrent levels by operating at a distinct timescale and use shortcut connections to alleviate learning issues made by increasing depths [88].

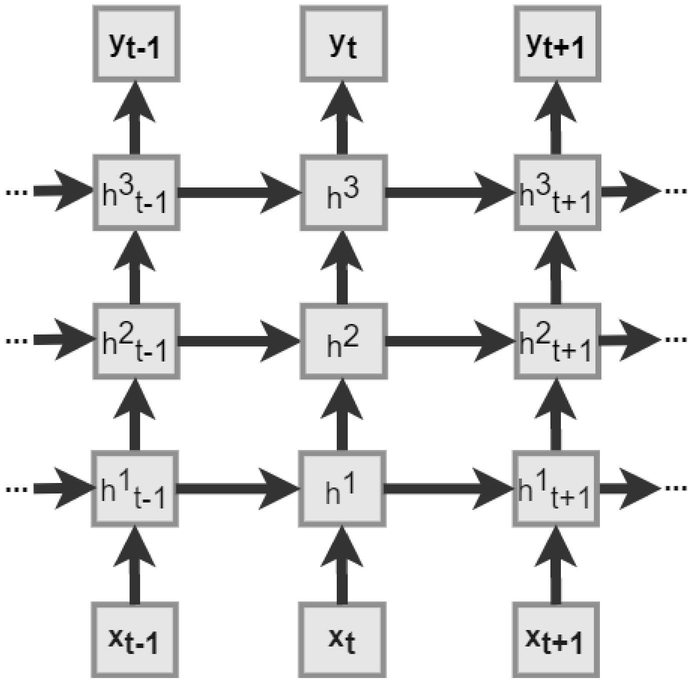


Figure 3.8: Conventional Stacked RNN architecture - Source: Adapted from Lambert (2014)[89]

$$\mathbf{h}_t^{(l)} = \mathbf{f}_h^{(l)}(h_t^{(l-1)}, \mathbf{h}_{t-1}^{(l)}) = \phi_h(\mathbf{W}_l^T \mathbf{h}_{t-1}^{(l)} + \mathbf{U}_l^T \mathbf{h}_t^{(l-1)}) \tag{3.8}$$

The formula for Stacked RNN is defined by 3.8, "where $\mathbf{h}_t^{(l)}$ is the hidden state of the l-th level at time t. When $l = 1$, the state is computed using $\mathbf{x}_t$ instead of $\mathbf{h}_t^{(l-1)}$. The hidden states of all the levels are recursively computed from the bottom level" $l = 1$ [88], [89].

This approach was considered for this thesis due to some studies [88], [89] have shown results that Stacked RNNs outperformed Vanilla RNNs, and either Vanilla LSTM, achieving the state-of-the-art results on the tasks of word-level language modelling and image captioning.

Long Short Term Memory (LSTM)

Long Short Term Memory (LSTM) is a type of RNN that efficiently solves several problems related to sequential data [90]. Unlike traditional RNNs, LSTM networks enable temporal relations through cells to store memories, input and output gates and forget gates that control the flow of information. The vanilla LSTM architecture (Figure 3.9) is similar to a regular cell, but its state is split into two vectors: short-term and long-term. Whereas the most important is short-term: where vector stores all information passed over time [91]. The network can learn if the data is necessary to store in the long-term state, or to throw away, as well as, what to read from it [92].

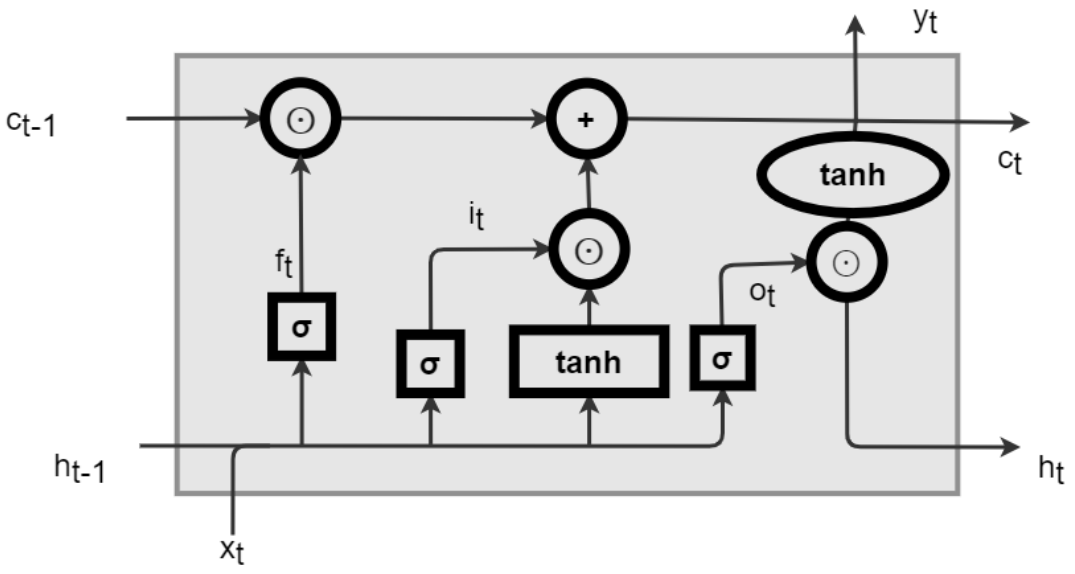


Figure 3.9: LSTM Cell - Source: Adapted from Weidman et al (2019)[78]

As the long-term state or $\mathbf{c}_{t-1}$, walk through the network, it first passes by forget gate, where it is decided if the information is thrown away. If not, then it stores new memories by addition operation and memories filtered at the input gate. The result $\mathbf{c}_t$ is sent straight out. Thus, this flow is straightforward without any transformation wherein each step and some memories are dropped or added. The long-term state is copied and goes through the tanh function, and afterward, the result is filtered by the output gate. This generates the short-term state or $\mathbf{h}_t$, which has the same content of the cell's output for this time step, $\mathbf{y}_t$ [91], [92].

The equations for the three gates (input, output, and forget) in LSTM cells are represented by 3.9. Where $\mathbf{w}_{xi}, \mathbf{w}_{xf}, \mathbf{w}_{xo} \in \mathbb{R}^{d\times h}$ and $\mathbf{w}_{hi}, \mathbf{w}_{hf}, \mathbf{w}_{ho} \in \mathbb{R}^{h\times h}$ are weight values and $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^{1\times h}$ are bias values.

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{x}_t \mathbf{w}_{xi} + \mathbf{h}_{t-1}\mathbf{w}_{hi} + \mathbf{b}_i),\\
\mathbf{f}_t &= \sigma(\mathbf{x}_t \mathbf{w}_{xf} + \mathbf{h}_{t-1}\mathbf{w}_{hf} + \mathbf{b}_f),\\
\mathbf{o}_t &= \sigma(\mathbf{x}_t \mathbf{w}_{xo} + \mathbf{h}_{t-1}\mathbf{w}_{ho} + \mathbf{b}_o).
\end{aligned} \tag{3.9}$$

The equations for the candidate memory cell, memory cell state and the hidden state are as follow in 3.10.

$$\begin{aligned}
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{x}_t \mathbf{w}_{xc} + \mathbf{c}_{t-1}\mathbf{w}_{hc} + \mathbf{b}_c),\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{c}}_t,\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t).
\end{aligned} \tag{3.10}$$

In 2017, Greff presented the most extensive study on LSTM networks, showing that none of the LSTM variants significantly outperforms the vanilla LSTM on classification tasks. However, as this thesis focuses on solving a regression problem, the LSTM and following LSTM variants are considered in the experiments of this thesis.

3.4.8 Stacked LSTM

This type of LSTM network differs from the vanilla LSTM left-to-right model. The Stacked LSTM (as shown in Figure 3.10) extends the architecture with a stack-pointer. As with LSTMs, the new inputs flow from the right-most position. But during calculating new memory contents, the stack pointer defines which cell of the network produces $ct - 1$ and $ht - 1$.

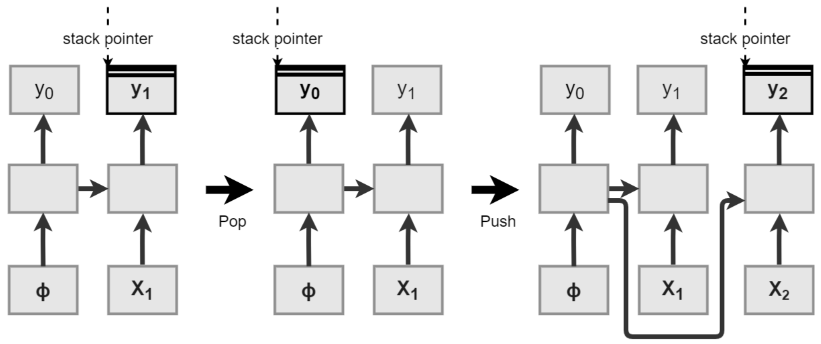


Figure 3.10: Stacked LSTM architecture - Source: Adapted from Dyer et al. (2005) [93]

On the top, the stack-pointer translocates to the previous component by a pop update operation. After, push operation adds a new register at the end of the list where back-pointer guides to the previous top [93].

3.4.9 Gated recurrent unit (GRU)

Gated recurrent units are a variation of vanilla LSTMs [94]. The principal dissimilarities between GRUs and LSTMs are the number of gates and maintenance of cell states. An update gate replaces the input and output gates, which control how much information to retain and update. The reset gate replaces the forget gate, which works similarly but in a different location. GRUs are faster and easier to train than LSTMs. However, they do not have high task performance.

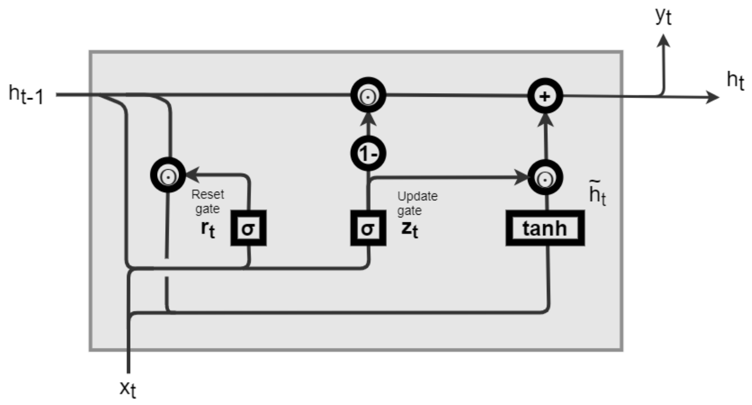


Figure 3.11: GRU Cell - Source: Adapted from Zhang et al. (2021)[78]

In GRU (3.11), the model computes reset gate $\mathbf{r}_t \in \mathbb{R}^{n\times h}$ and update gate $\mathbf{z}_t \in \mathbb{R}^{n\times h}$ as follows:

$$\begin{aligned} \mathbf{r}_t &= \sigma(\mathbf{x}_t\mathbf{w}_{xr} + \mathbf{h}_{t-1}\mathbf{w}_{hr} + \mathbf{b}_r), \\ \mathbf{z}_t &= \sigma(\mathbf{x}_t\mathbf{w}_{xz} + \mathbf{h}_{t-1}\mathbf{w}_{hz} + \mathbf{b}_z), \end{aligned} \tag{3.11}$$

where $\mathbf{w}_{xr}, \mathbf{w}_{xz} \in \mathbb{R}^{d\times h}$ and $\mathbf{w}_{hr}, \mathbf{w}_{hz} \in \mathbb{R}^{h\times h}$ are weight values and $\mathbf{b}_r, \mathbf{b}_z \in \mathbb{R}^{1\times h}$ are biases.

After updating reset and update gates, the candidate hidden state $\tilde{\mathbf{h}}_t \in \mathbb{R}^{n\times h}$ at time step t is calculated as:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{X}_t\mathbf{w}_{xh} + (\mathbf{r}_t \odot \mathbf{h}_{t-1})\,\mathbf{w}_{hh} + \mathbf{b}_h), \tag{3.12}$$

where $\mathbf{w}_{xh} \in \mathbb{R}^{d\times h}$ are weight values, the symbol $\odot$ represents an elementwise product operator and $\mathbf{b}_h \in \mathbb{R}^{1\times h}$ is the bias. Different than what happens with RNNs, the previous states are minimized through $\odot$ multiplication of $\mathbf{r}_t$ and $\mathbf{h}_t$.

3.4.10 Bidirectional LSTM (BiLSTM)

Two RNNs stacked on top of each other form the bidirectional LSTM architecture (3.12). The first network learns the sequence in its standard series, while the other reads in the opposite direction. Thus, at a given time step t, the network output depends on the outputs at all previous time steps. BiLSTM has the advantage of representing elements sequentially without losing their context. In this sense, such output can also depend on future outputs, which significantly helps natural language processing tasks, as the whole data iteration can represent a context given necessary to predict words or phrases accordingly [95].

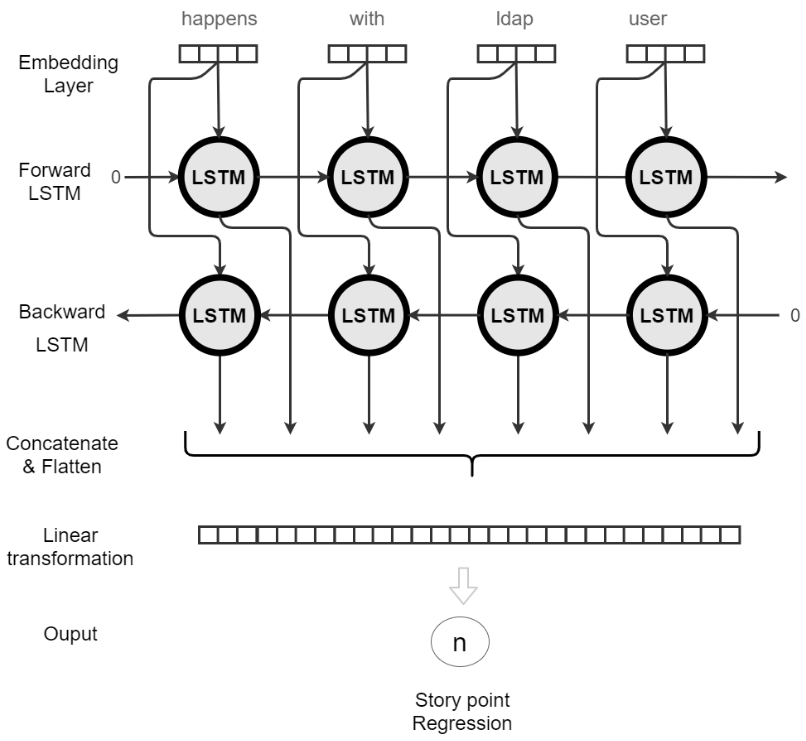


Figure 3.12: Source: Adapted from Cornegruta et al. (2016)

3.4.11 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN, ConvNet) is a deep learning algorithm inspired by the visual structure of the human cortex, where isolated neurons respond to stimuli at a specific location in the visual (receptive) field. The ConvNet layers contain many filters, which by numeric matrices are responsible for reducing the input data's size and highlighting local patterns. From the results of these filters, the information is summarized through a pooling operation. The purpose is to assign a semantic value to the original data and classify them through a feed-forward neural network. CNNs have been widely applied and achieved state-of-the-art in different applications using video and image recognition, recommender systems and natural language processing [96]. Although CNNs mostly rely on classification problems, some researchers [97] achieved state-of-the-art using CNNs for regression problems. Thus, we can also employ CNN to 1-dimensional problem, such as predicting the next value or word in a sentence.

Figure 3.13 illustate a simplified schema of a 1-dimensional CNN, such as the model used in the experiments of this thesis.

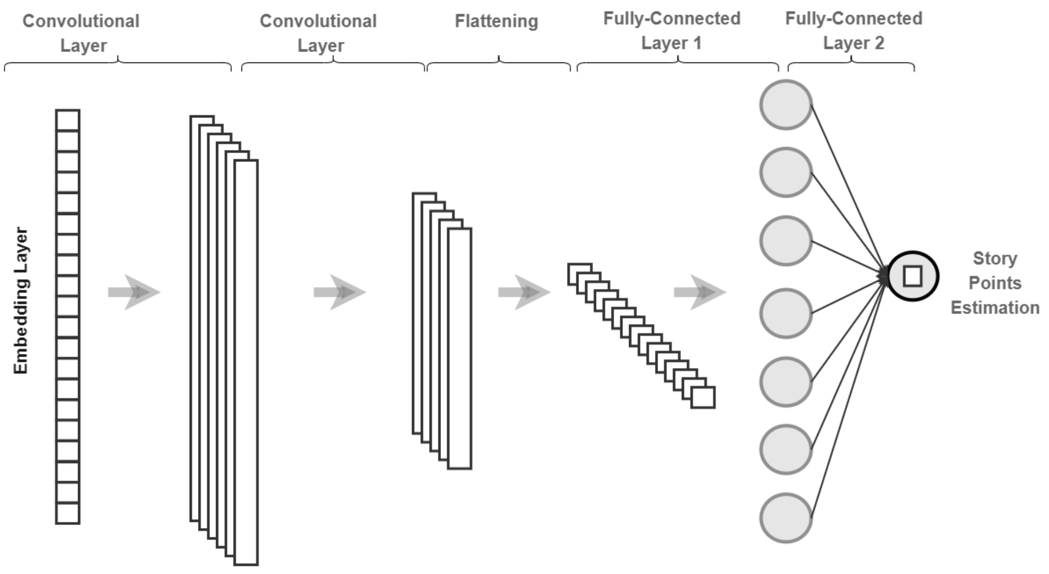


Figure 3.13: Simplified schema of a 1D CNN - Source: Adapted from Lewinson (2020) [98]

Mathematically, the convolutional layer can be represented by the formula 3.13.

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{V}]_{a,b}[\mathbf{X}]_{i+a,j+b}. \tag{3.13}$$

3.5 Performance Metrics

The metrics for evaluating machine learning algorithm performance against datasets are performed to identify how well the predictions arising from the model reproduce the observed value for the expected response. Although there are several metrics in the context of regression-based software effort prediction, the main ones are: Mean Absolute Error (MAE), Median Absolute Error (MdAE), and Mean Square Error

(MSE).

The MAE is considered the most fundamental error in regression. It measures the difference between two continuous variables, i.e. the average error that each variable has relative to the regression line. Supposing X and Y are observations for effort estimation prediction, we can consider X as the actual story points effort and Y as the predicted story points effort. The MAE can be defined by the formula 3.14 below:

$$MAE = \frac{1}{N}\sum_{i=1}^{n} |ActualSP_i - EstimatedSP_i| \tag{3.14}$$

Where N is the number of user stories or issues referred to the test set, and ActualSP is the current story point measure, and EstimatedSP is the estimated story point for a given user story [24].

In this thesis, MdAE was also used, suggested by Choetkiertikul et al. (2018), as a more robust metric to large outliers. The MdAE is represented by the formula 3.15.

$$MdAE = median|ActualSP_i - EstimatedSP_i| \tag{3.15}$$

The MSE 3.6 is the same mentioned previously, which is the error between predicted and actual values returned by the regression. Then, MSE value equal to zero is ideal and indicates a better model performance.

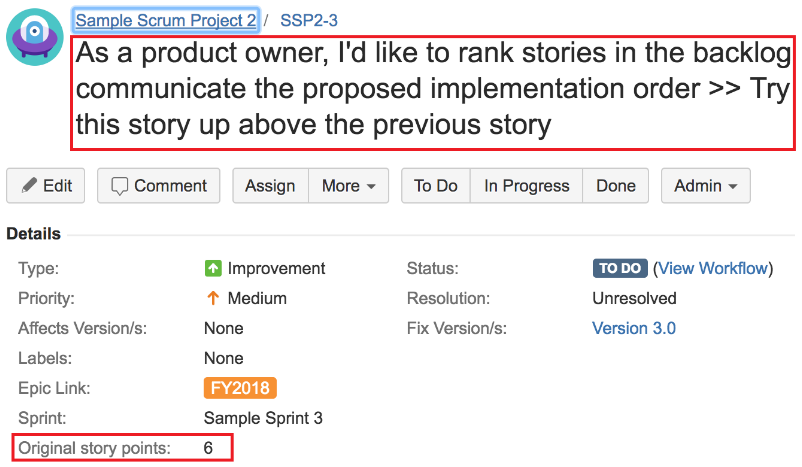
Chapter 4

Data Collection and Preprocessing

This chapter briefly describes Jira, the dataset, and some preprocessing techniques for handling the textual requirements present in the dataset chosen for this thesis.

4.1 Jira

Jira is a powerful work management tool for agile software development, initially designed as a bug and issue tracker tool [17]. Nowadays, Jira is the most adopted tracking system tool that supports agile teams to manage requirements [13] such as User Stories.


Sample Scrum Project 2 / SSP2-3

As a product owner, I'd like to rank stories in the backlog communicate the proposed implementation order >> Try this story up above the previous story

Details

Type:	Improvement	Status:	TO DO (View Workflow)
Priority:	Medium	Resolution:	Unresolved
Affects Version/s:	None	Fix Version/s:	Version 3.0
Labels:	None		
Epic Link:	FY2018		
Sprint:	Sample Sprint 3		
Original story points:	6		

Figure 4.1: Example of an User Story in Jira - Source: Atlassian [99]

The figure 4.1 shows an instance of a User Story registered as SSP2-3 in the project named "Sample Scrum Project 2" in Jira. This register contains various attributes, in which the ones mainly related to effort estimation are: Title, Description and Original Story Points. Typically, the title has some brief but valuable information about the functionality requirement. At the same time, the description contains further specifications. Finally, the Original Story Points show the estimation value made by the team.

4.2 Dataset

This thesis employs a dataset formed by textual software requirements and story points effort estimation for user stories and issues extracted from Jira [17] project's repositories. A similar story point dataset with 5607 issues was made available by Porru et al. (2016). Subsequently, Choetkiertikul et al. (2018) provided a comprehensive story points-based dataset from most of the same public repositories, extracting story points, titles and descriptions from 16 projects. Henceforward, Scott et al. (2018), Marapelli et al. (2020) employed this dataset or somehow. Table 4.1 shows the dataset attributes.

Project	File	Ab	# issues
Mesos	mesos.csv	ME	1680
Usergrid	usergrid.csv	UG	482
Appcelerator studio	appceleratorstudio.csv	AS	2919
Aptana studio	aptanastudio.csv	AP	829
Titanium SDK/CLI	titanium.csv	TI	2251
Duracloud	duracloud.csv	DC	666
Bamboo	bamboo.csv	BB	521
Clover	clover.csv	CV	384
Jira Software	jirasoftware.csv	JI	352
Moodle	moodle.csv	MD	1166
Data management	datamanagement.csv	DM	4667
Mule	mule.csv	MU	889
Mule studio	mulestudio.csv	MS	732
Spring XD	springxd.csv	XD	3526
Talend Data Quality	talenddataquality.csv	TD	1381
Talend ESB	talendesb.csv	TE	868
Total			23,313

Table 4.1: **Numbers of user stories, issues and project descriptions extracted from Jira issue tracking system [24]**

To the best our knowledge, the dataset produced by Choetkiertikul et al. (2018) [24] is currently the most extensive Story Points dataset available. The dataset is formed by 23,313 user stories and issues extracted from Jira issue tracking system, collected from 16 large open-source projects issues estimated with story points: Apache Mesos (ME), Apache Usergrid (UG), Appcelerator Studio (AS), Aptana Studio (AP), Titanum SDK/CLI (TI), DuraCloud (DC), Bamboo (BB), Clover (CV), JIRA Software (JI), Moodle (MD), Data Management (DM), Mule (MU), Mule Studio (MS), Spring XD (XD), Talend Data Quality (TD), and Talend ESB (TE) [24]. The Figure 4.2 illustrates the most frequently words for User Stories and issues from Bamboo dataset.

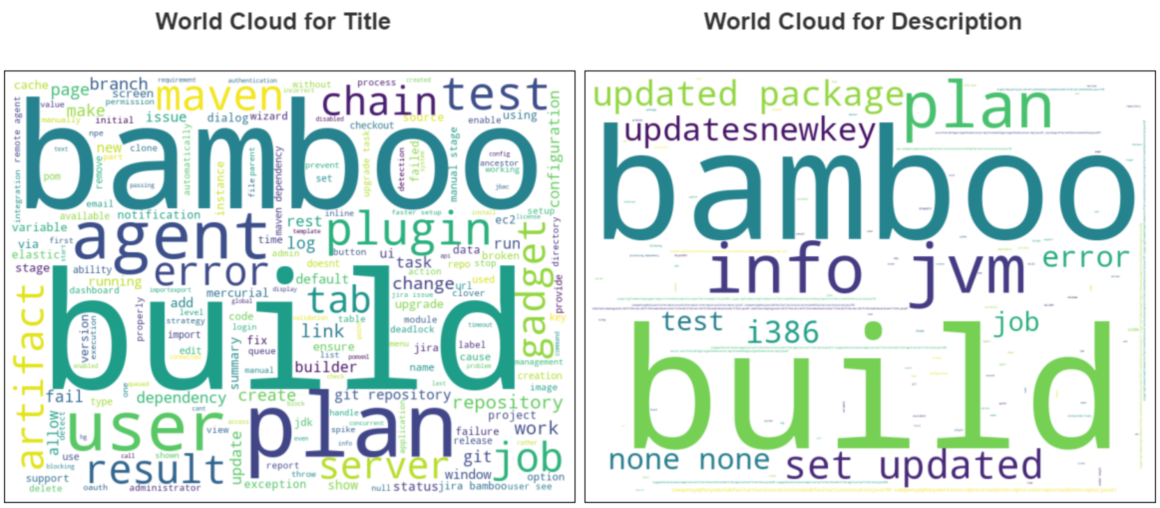


Figure 4.2: WordCloud for User Stories present in Bamboo Dataset

4.3 Text Preprocessing

Since machine learning and deep learning models do not understand plain text, we must perform several texts preprocessing steps against the raw data (dataset). Inconsistencies in the dataset should be transformed or excluded to the most suitable form before extracting features that will feed the models. For data modelling, we followed the same approach taken by Choetkiertikul et al. (2018) [24]. That is, merging title and description attributes of a given user story (or issue) into a single text column.

At first, after loading the dataset, we have created one new column by joining the most critical columns (title and description). After this new structure,we performed the steps described below against the new feature column ”titledesc”.

i Text data cleaning: converts all words to lowercase, remove line breaks, normalize spaces, removes punctuation marks and spell-check.

ii Tokenization: splits the document into a list of strings (tokens) by delimiters like white space between terms, line breaks, tabs, and some special characters. In order to prevent the process from ignoring out-of-vocabulary words, we set the parameter oov_token (Out Of Vocabulary) when instantiating the Tokenizer object and thus represent unknown words with the value "OOV".

iii Stopwords: cleans each token split based on irrelevant terms that do not belong to the class, such as numbers, articles, adverbs, prepositions and pronouns.

iv Lemmatization: reduces a word to a common base form, where words derived from the same root count as a single term. Stemming is more straightforward and faster as it simply chops off the ends of words and typically removes derivational affixes. However, we decided to do Lemmatization, as it is based on vocabulary and morphological analysis of words and intends to eliminate inflectional endings only and reinstate the base form of a word (known as the lemma) [100].

v Padding: pads sequences of tokens smaller than a defined maximum length size with zero. That is why sentences do not have the same length and could be longer or shorter. This technique is necessary as neural networks require the same shape and size.

Chapter 5

Results and Discussion

This chapter will present the results and evaluation of the effectiveness of machine learning and deep learning to estimate story points as by the methodology described in Chapter 3. Most of the current deep learning-based models for story points estimation considered approaches with Convolution Networks, Recurrent Networks, LSTM, BiLSTM. However, few papers have explored deep learning in combination with word embeddings. To excavate semantic text features, we considered GloVe as our word embedding methods. Later we compare our results with Ionescu et al. (2017) and Choetkiertikul et al. (2018), which also considered semantic text features from user stories, but with different word embeddings methods: Doc2Vec and Word2Vec.

Although most of the algorithms we employ in the experiments are from deep learning architecture, we also implemented the traditional Random Forest ML algorithm as it have been demonstrating a powerful model for software effort estimation. That said, we carried out seven different deep learning regressor to evaluate the story points based effort prediction accuracy of each algorithm and further comparison of results.

5.1 Text Feature Selection

After a comprehensive selection of attributes, the MIFS algorithm considers the mutual information related to both class and attributes previously selected. The algorithm decides the following attribute from a set of attributes that increases the information about the classes. Therefore, an attribute shows information about the class without having predictability within the current set of attributes.

The Figure 5.1 illustrate MIFS results over the Bamboo dataset, in which the most notable sentence was ”access done anonymous mode”. Given the variables, MIFS derived an expression for the mutual information based on terms put into a wordlist before and then found the information shared between the random variables.

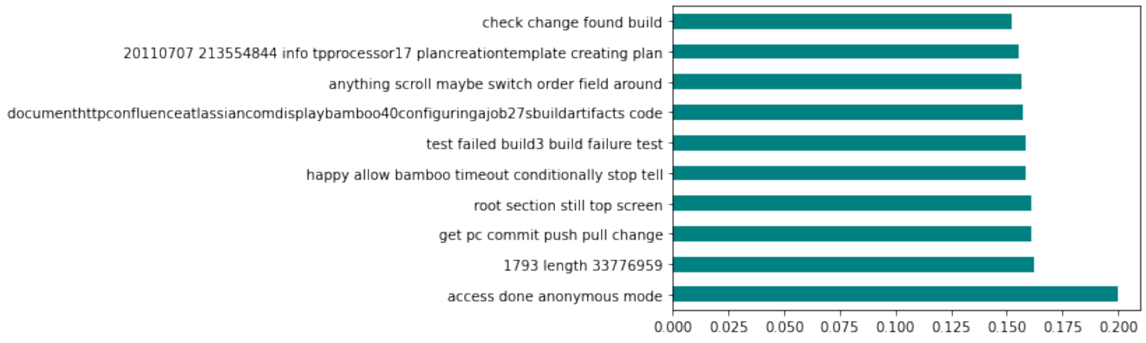


Figure 5.1: Mutual Information Feature Selection on Bamboo dataset

The figure 5.2 shows the 1-gram and 2-gram text features extracted from Bamboo dataset.

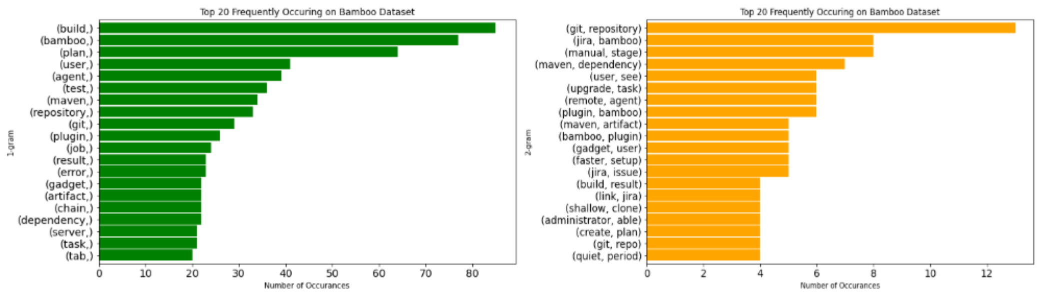


Figure 5.2: The 1-gram and 2-gram model over Bamboo dataset

5.2 Word2vec Visualization

For the word embeddings analysis, we used T-SNE to visualize high-dimensional word vectors cluster. T-SNE is used to project these vectors into two dimensions while preserving local stucture. Figure 5.3 presents the Word2Vec visualization of the Bamboo dataset.

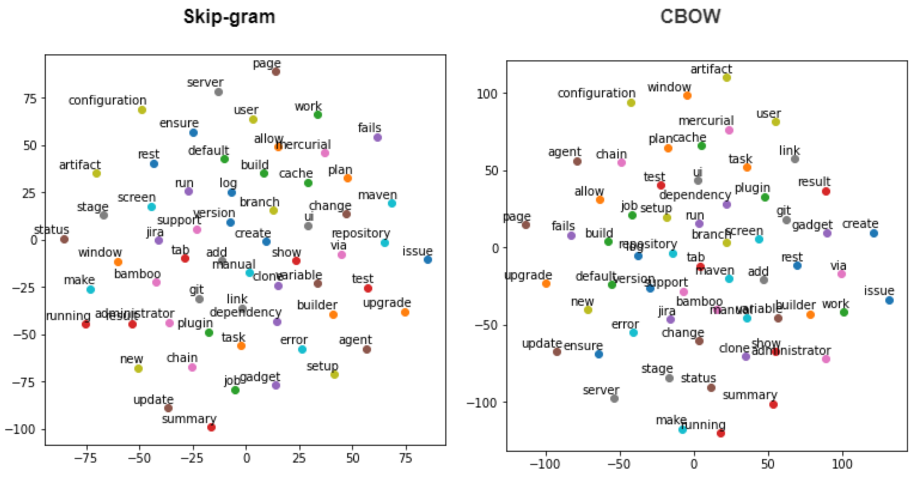


Figure 5.3: Word2Vec visualization of Bamboo dataset

5.3 Deep learning Architectures

We set different epochs for training the DL models: 20 and 50. The reason for some having 20 epochs is that during training, the number of epochs greater than 20 no longer improves model accuracy. Thus to avoid underfitting the model, we keep fewer epochs for the models described. The figure 5.4 illustrates the overfitting of the sRNN model during the experiments with Bamboo dataset.

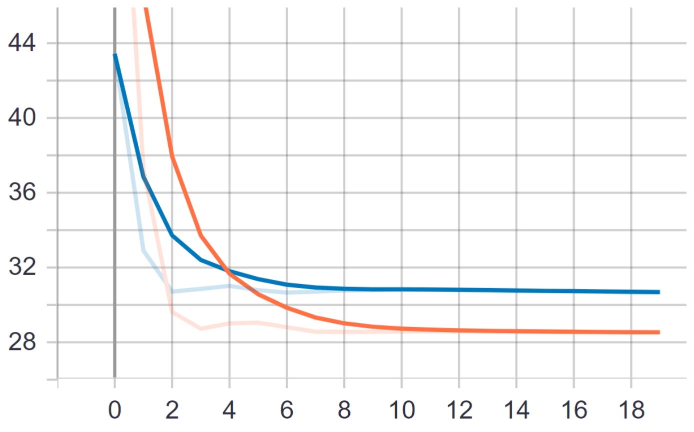


Figure 5.4: Epoch x loss - sRNN model on Bamboo dataset

During training the models, we performed a parameter observation to better estimate and learn the effect of specific parameters. While training, we changed gradually the following parameter configuration: hidden neuron, learning rate, number of epochs, and dropout rate. For the LSTM model, the SGD optimizer was used with a learning rate of 0.01, for the sLSTM and BiLSTM models, the Adam optimizer was used. For models CNN, GRU, RNN, and sRNN, the optimizer Adam was also used with a dropout rate of 0.3. For all models, the batch_size parameter was 256. The dropout layers in neural network learning help in stopping over-fitting of the network by provisionally avoiding weights of neurons in the learning process. The learning rate of 0.01 or greater caused the cost of training and validation to converge faster with each iteration. As stated by Suayano et al. (2020), smaller values of learning rate equal to 0.000001 or less can make the cost change slightly until the iteration ends [101]

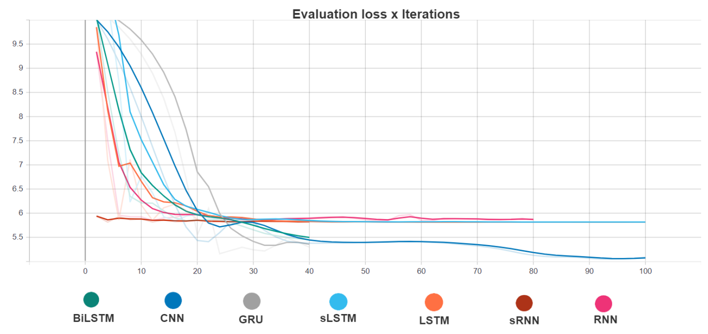


Figure 5.5: Evaluation loss x Iteration - All models over the Bamboo training set

Figure 5.5 demonstrates the evaluation loss at each epoch iteration of all models during the training set over the Bamboo dataset. After training the models, we generated the summary and diagrams for each DL architecture, as shown in the appendixes (LSTM A.1, CNN A.3, GRU A.5, RNN A.7, sRNN A.9, BiLSTM A.11, sLSTM A.13). The vectors representing the word embedding layers are loaded and passed as parameters to the "module_wrapper()" layer of the deep learning architecture.

5.4 Evaluation of the Deep Learning models

Next are presented the results of the experiments performed on the sixteen story points datasets. The following tables (5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 5.10, 5.11, 5.12, 5.13, 5.14, 5.15, 5.16) demonstrate the performance of each model when estimating story points using the datasets (Bamboo, Appceleratorstudio, Aptanastudio, Clover, Datamanagement, Duracloud, Jirasoftware, Mesos, Moodle, Mule, Mulestudio, SpringXD, Talenddataquality, Talendesb, Titanium, Usergrid) respectively. The values of MAE, MdAE and MSE, MdAE and RMSE are shown, which were obtained after applying the 10-fold cross-validation. The best results are highlighted in bold. For all the metrics used, the smaller the value, the better the result.

Regressors	MSE	MAE	MdAE	RMSE
LSTM	5.82	1.36	1.20	2.41
CNN	6.23	1.42	**0.94**	2.50
GRU	5.88	1.41	1.50	2.41
RNN	5.86	1.32	1.28	2.42
RNN-Stack	5.82	1.38	1.23	2.41
Bi-LSTM	5.45	**1.29**	0.98	2.97
LSTM-Stack	5.82	1.35	1.33	3.34
Random Forest	**5.26**	1.47	1.02	**2.29**

Table 5.1: Bamboo dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	**7.58**	2.06	2.46	2.75
CNN	9.21	2.29	1.81	3.03
GRU	7.32	2.00	2.16	**2.71**
RNN	7.60	2.09	2.37	2.76
RNN-Stack	14.02	2.89	2.24	3.74
Bi-LSTM	6.85	**1.91**	1.55	5.16
LSTM-Stack	7.59	2.09	2.37	6.09
Random Forest	7.39	2.02	**1.70**	2.72

Table 5.2: Appceleratorstudio dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	40.65	4.41	4.62	6.38
CNN	11.78	2.55	**2.01**	3.43
GRU	**7.50**	**2.10**	2.32	**2.74**
RNN	41.95	4.38	4.68	6.48
RNN-Stack	41.98	4.43	4.85	6.48
Bi-LSTM	40.99	4.33	4.18	8.74
LSTM-Stack	42.00	4.44	4.87	9.88
Random Forest	41.01	4.45	3.93	6.40

Table 5.3: Aptanastudio dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	39.48	3.72	2.14	6.28
CNN	**23.62**	**3.18**	2.28	**4.86**
GRU	51.81	4.01	**1.59**	7.20
RNN	39.84	3.65	2.17	6.31
RNN-Stack	42.58	3.68	1.82	6.53
Bi-LSTM	36.59	3.66	2.44	7.36
LSTM-Stack	38.70	3.63	2.06	7.89
Random Forest	37.41	4.07	2.99	6.12

Table 5.4: Clover dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	284.19	9.59	7.48	16.86
CNN	221.22	**8.05**	3.58	14.87
GRU	250.83	8.48	4.53	15.84
RNN	286.16	9.20	6.66	16.92
RNN-Stack	243.55	7.77	**3.50**	15.61
Bi-LSTM	**219.25**	7.41	3.64	**14.81**
LSTM-Stack	284.05	9.63	7.54	16.85
Random Forest	254.33	10.19	6.83	15.95

Table 5.5: Datamanagement dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	3.74	1.15	1.13	1.94
CNN	4.42	1.41	1.06	2.10
GRU	3.90	1.18	1.14	1.97
RNN	4.10	**1.04**	**0.47**	2.02
RNN-Stack	5.90	1.47	1.03	2.43
Bi-LSTM	**3.67**	1.17	0.90	**1.92**
LSTM-Stack	3.84	1.11	1.00	1.96
Random Forest	4.21	1.34	0.99	2.05

Table 5.6: Duracloud dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	6.16	2.13	2.40	2.48
CNN	4.05	**1.51**	**1.17**	**2.01**
GRU	7.34	2.24	2.03	2.71
RNN	6.58	2.22	2.56	2.57
RNN-Stack	6.28	2.14	2.41	2.51
Bi-LSTM	6.26	2.14	2.28	2.50
LSTM-Stack	6.74	2.24	2.68	2.60
Random Forest	5.34	1.81	1.43	2.31

Table 5.7: Jirasoftware dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	4.94	1.48	1.01	2.22
CNN	**3.94**	1.43	1.00	**1.99**
GRU	4.87	1.48	1.06	2.21
RNN	5.00	1.48	**0.76**	2.24
RNN-Stack	4.89	1.71	1.44	2.21
Bi-LSTM	4.00	**1.37**	0.88	2.00
LSTM-Stack	4.77	1.63	1.28	2.18
Random Forest	4.92	1.70	1.36	2.22

Table 5.8: Mesos dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	516.03	13.99	**9.44**	22.72
CNN	478.93	**13.19**	7.00	21.88
GRU	441.94	13.40	10.14	21.02
RNN	444.04	13.26	9.68	21.07
RNN-Stack	443.05	13.38	10.10	21.05
Bi-LSTM	**441.29**	13.34	10.03	**21.01**
LSTM-Stack	443.33	13.59	10.58	21.06
Random Forest	480.51	14.81	11.03	21.92

Table 5.9: Moodle dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	11.85	2.59	2.90	3.44
CNN	11.93	2.64	**2.16**	3.45
GRU	12.03	2.62	2.71	3.47
RNN	11.96	2.59	2.79	3.46
RNN-Stack	11.87	2.56	2.98	3.45
Bi-LSTM	**10.71**	**2.55**	2.40	**3.27**
LSTM-Stack	11.85	2.58	2.93	3.44
Random Forest	13.66	2.97	2.58	3.70

Table 5.10: Mule dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	30.67	3.96	3.40	5.54
CNN	31.71	4.03	2.79	5.63
GRU	31.94	**3.63**	2.47	5.65
RNN	30.78	3.89	3.17	5.55
RNN-Stack	30.67	3.98	3.44	5.54
Bi-LSTM	**30.55**	3.89	3.12	**5.53**
LSTM-Stack	30.74	3.91	3.23	5.54
Random Forest	34.66	4.29	**3.11**	5.89

Table 5.11: Mulestudio dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	7.66	2.07	1.56	2.77
CNN	10.64	2.27	1.58	3.26
GRU	7.78	2.13	1.72	2.79
RNN	7.71	2.07	1.52	2.78
RNN-Stack	8.32	2.23	1.80	2.88
Bi-LSTM	**6.87**	**1.90**	**1.51**	**2.62**
LSTM-Stack	7.31	2.05	1.58	2.70
Random Forest	8.62	2.22	1.89	2.94

Table 5.12: SpringXD dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	31.76	4.02	3.05	5.64
CNN	32.54	4.18	3.29	5.70
GRU	30.54	3.90	**2.74**	5.53
RNN	31.56	3.89	3.48	5.62
RNN-Stack	36.22	4.22	3.31	6.02
Bi-LSTM	**29.34**	**3.78**	2.81	**5.42**
LSTM-Stack	30.21	3.94	2.93	5.50
Random Forest	31.31	4.02	3.26	5.60

Table 5.13: TalendDataQuality dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	1.81	0.96	0.96	1.34
CNN	1.89	0.97	0.76	1.38
GRU	1.94	0.96	0.95	1.39
RNN	2.07	1.00	0.98	1.44
RNN-Stack	2.10	1.03	1.07	1.45
Bi-LSTM	**1.65**	**0.91**	**0.68**	**1.29**
LSTM-Stack	1.76	0.95	0.86	2.51
Random Forest	2.15	1.11	1.00	1.47

Table 5.14: Talendesb dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	34.23	3.68	**2.06**	5.85
CNN	35.92	3.89	2.60	5.99
GRU	34.27	3.70	2.15	5.85
RNN	35.05	3.59	2.30	5.92
RNN-Stack	34.29	3.68	2.44	5.86
Bi-LSTM	30.61	**3.31**	2.17	5.53
LSTM-Stack	33.91	3.67	2.09	8.76
Random Forest	**28.24**	3.36	2.27	**5.31**

Table 5.15: Titanium dataset

Regressors	MSE	MAE	MdAE	RMSE
LSTM	2.39	1.03	**0.24**	1.55
CNN	**2.20**	0.99	0.59	1.48
GRU	2.49	1.07	0.55	1.58
RNN	2.42	1.07	0.34	1.56
RNN-Stack	2.43	1.04	0.34	1.56
Bi-LSTM	**2.19**	**0.99**	0.59	**1.47**
LSTM-Stack	2.27	1.04	0.63	1.51
Random Forest	2.26	1.05	0.72	1.50

Table 5.16: Usergrid dataset

Although the metrics MSE, MAE, MdAE and RMSE of the BiLSTM model prediction outperformed our other models (as shown in Figure 5.5) and was even unanimous for some datasets such as SpringXD and Talendesb (shown in tables 5.12, 5.14). Our model did not outperform when compared to the most similar approaches to the one presented in the work of this thesis (Choetkiertikul et al. 2018 and Marapelli et al. 2020). We can note that our best MAE and MdAE results (1.29, 0.94) that we obtained with the Bamboo dataset (as shown in Table 5.1) are lower when compared to the MAE and MdAE results (0.74, 0.61) returned by the model presented by Choetkiertikul et al., 2018. The same scenario occurs when comparing our same MAE results (1.29) with the best MAE results (0.72) presented by Marapelli et al., 2020. We believe that the author's approach is better accurate because they combined more than one deep learning algorithm such as RNN and CNN (Marapelli et al. 2020). We aim to implement this similar approach to future works with the backdrop presented in the conclusion section.

Chapter 6

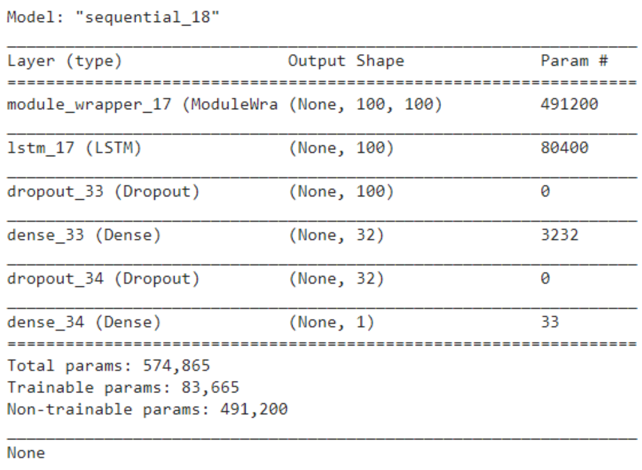
Conclusion and Future Works

Since accuracy in estimating software development efforts can represent considerable costs, many studies on deep learning and natural language processing applied to agile environments are in the interest of researchers worldwide [3]. However, most of the studies bases on syntax or word-level approaches. Likewise, with NLP research in general, the semantics and context approach remains a challenge [49]. Our experiments state that the Deep Learning and pre-trained word embeddings techniques are ambitious compared to traditional regressors for effort estimation like Random Forest. However, these methods are susceptible to the data used for training. We can say that the greater the volume and diversity of samples in the corpus used, the better the performance of the models and story point estimates can be. From the experiments presented, we can see the effectiveness of using deep learning models for some datasets and others not. We believe that this variance is because user stories or issues usually have short texts and short vocabulary, which may mean that many expressions are familiar to software engineering and shared among agile environments. Against this backdrop, it is essential to identify different semantic and contexts in a more advanced way. Fine-tuning the word embedding method (GloVe or Word2Vec) used can be a solution. Recently, word embedding methods considered contextualized like BERT promise to solve polysemy and ambiguity problems by self-attention mechanism. In future work, we would like to implement combined deep learning architecture and word embeddings finetuning. We also aim to study auto-machine learning (AutoML) to automatically feed DL models by taking the textual requirements of user stories or issues generated in the initial development phases. Finally, we hope to build a suitable model for deployment in agile environments.

Appendix A

Deep Learning Models Architectures

A.1 Summary and Model Architectures



```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper_17 (ModuleWra (None, 100, 100)          491200
_________________________________________________________________
lstm_17 (LSTM)               (None, 100)               80400
_________________________________________________________________
dropout_33 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_33 (Dense)             (None, 32)                3232
_________________________________________________________________
dropout_34 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_34 (Dense)             (None, 1)                 33
=================================================================
Total params: 574,865
Trainable params: 83,665
Non-trainable params: 491,200
_________________________________________________________________
None
```

Figure A.1: Summary - LSTM model

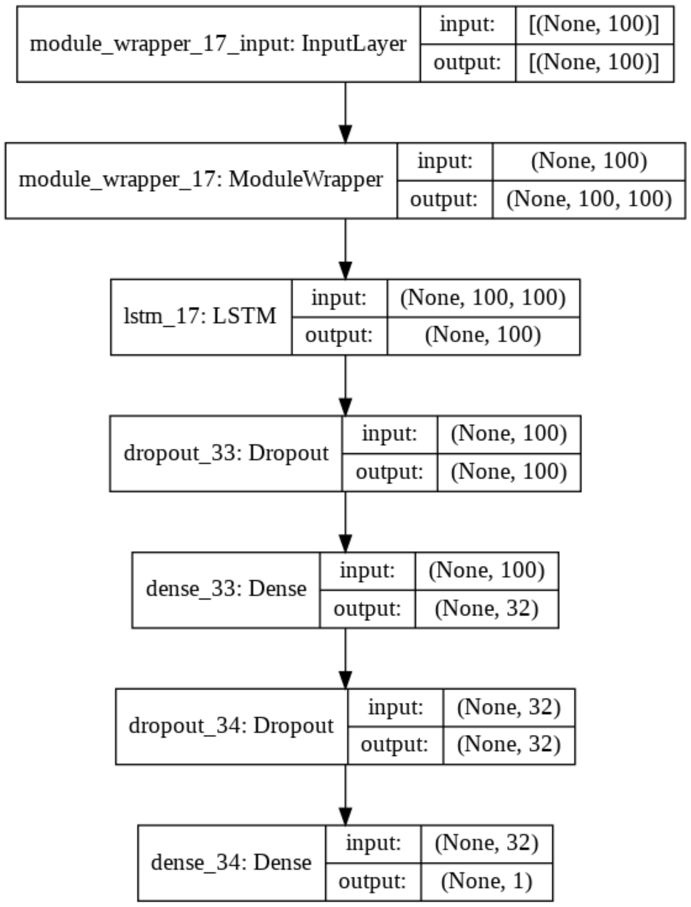


Figure A.2: Architecture - LSTM model

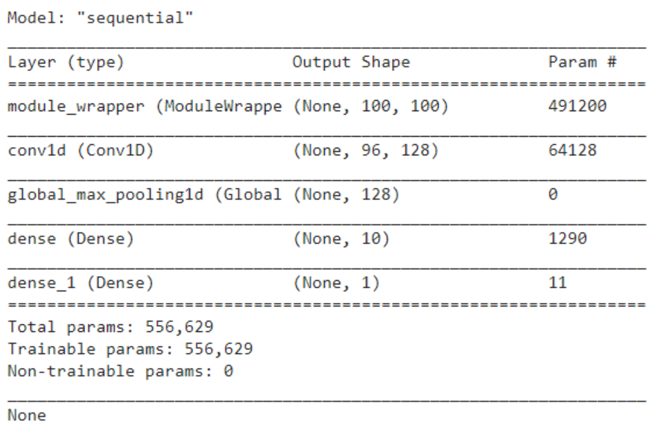

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper (ModuleWrappe (None, 100, 100)          491200
_________________________________________________________________
conv1d (Conv1D)              (None, 96, 128)           64128
_________________________________________________________________
global_max_pooling1d (Global (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 10)                1290
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 11
=================================================================
Total params: 556,629
Trainable params: 556,629
Non-trainable params: 0
_________________________________________________________________
None
```

Figure A.3: Summary - CNN model

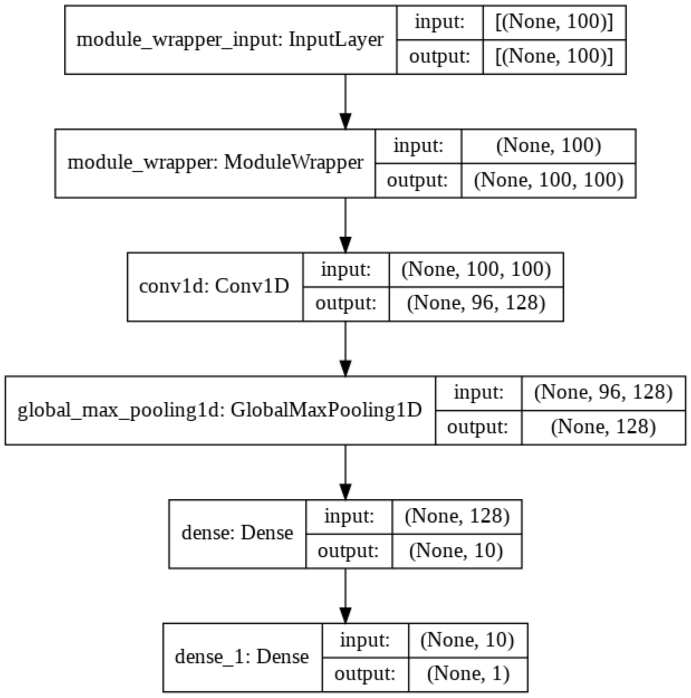
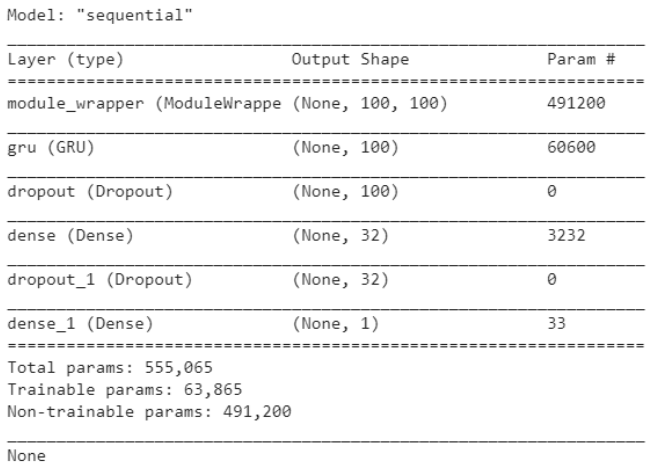


Figure A.4: Architecture - CNN model



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper (ModuleWrappe (None, 100, 100)          491200
_________________________________________________________________
gru (GRU)                    (None, 100)               60600
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense (Dense)                (None, 32)                3232
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 555,065
Trainable params: 63,865
Non-trainable params: 491,200
_________________________________________________________________
None
```

Figure A.5: Summary - GRU model

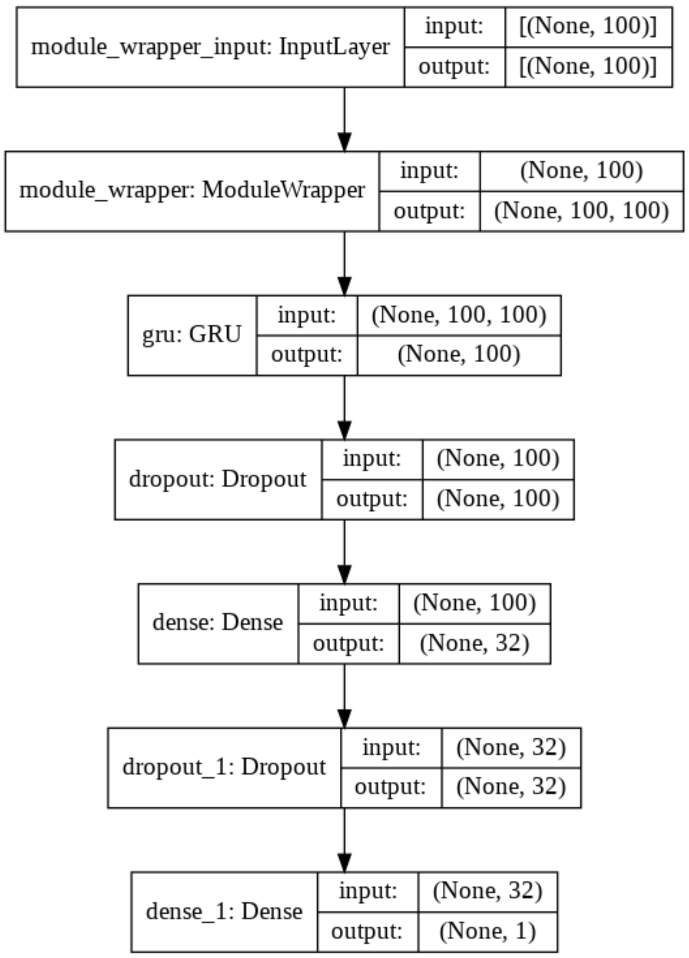
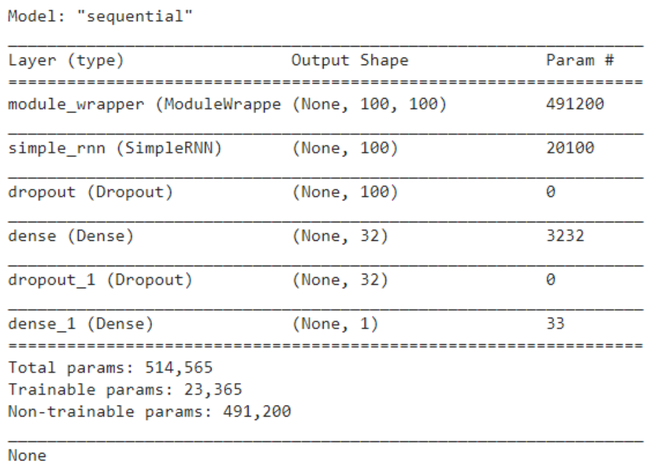


Figure A.6: Architecture - GRU model



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper (ModuleWrappe (None, 100, 100)          491200
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 100)               20100
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense (Dense)                (None, 32)                3232
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 514,565
Trainable params: 23,365
Non-trainable params: 491,200
_________________________________________________________________
None
```

Figure A.7: Summary - RNN model

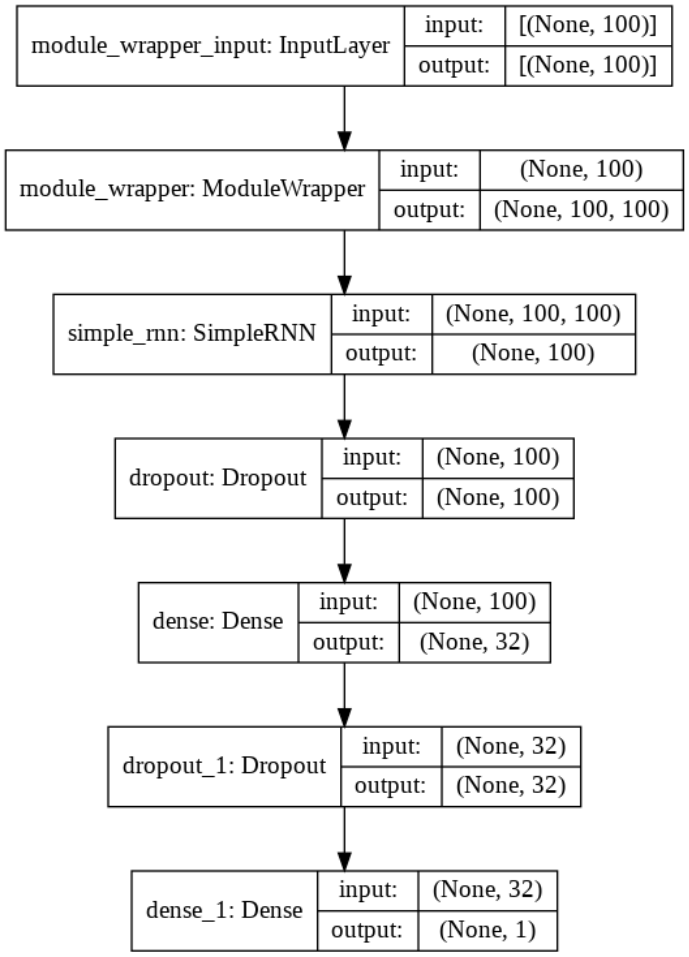
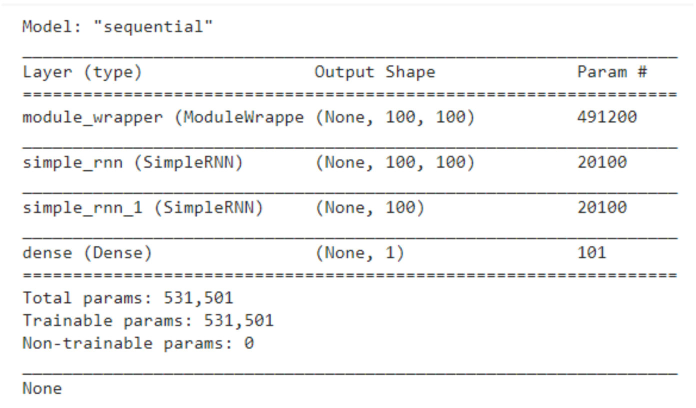


Figure A.8: Architecture - RNN model



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper (ModuleWrappe (None, 100, 100)          491200
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 100, 100)          20100
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 100)               20100
_________________________________________________________________
dense (Dense)                (None, 1)                 101
=================================================================
Total params: 531,501
Trainable params: 531,501
Non-trainable params: 0
_________________________________________________________________
None
```

Figure A.9: Summary - RNN Stack model

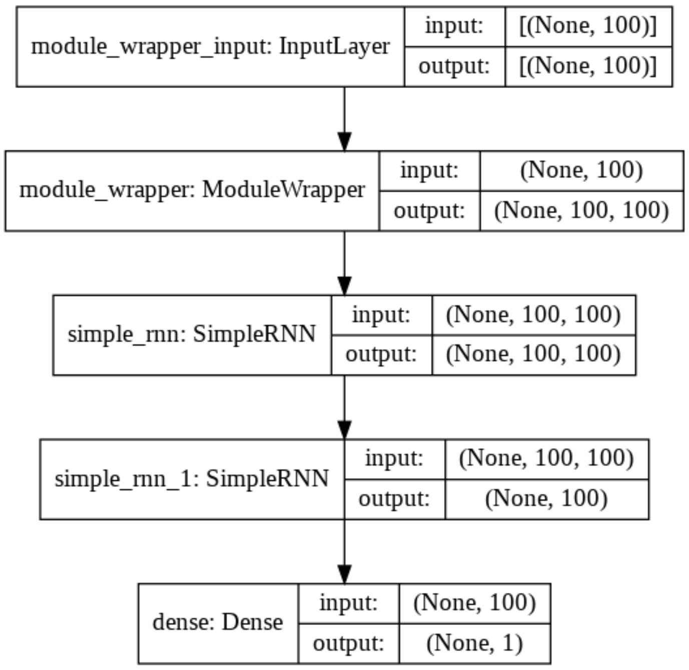
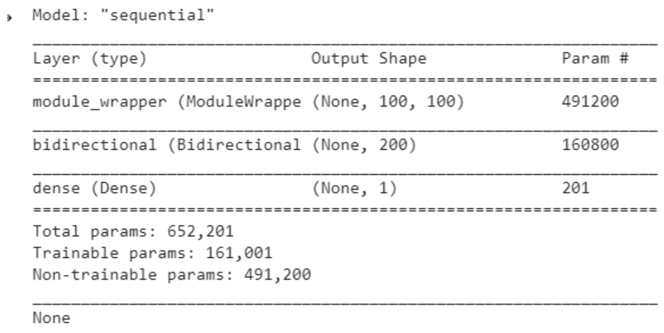


Figure A.10: Architecture - RNN stack model



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper (ModuleWrappe (None, 100, 100)          491200
_________________________________________________________________
bidirectional (Bidirectional (None, 200)               160800
_________________________________________________________________
dense (Dense)                (None, 1)                 201
=================================================================
Total params: 652,201
Trainable params: 161,001
Non-trainable params: 491,200
_________________________________________________________________
None
```

Figure A.11: Summary - BiLSTM model

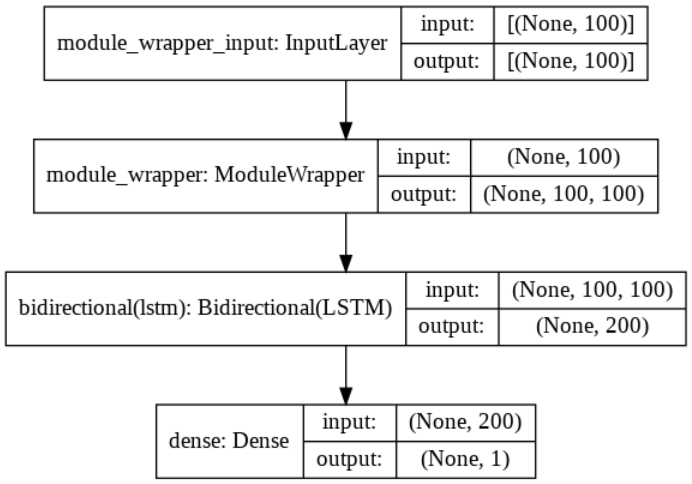
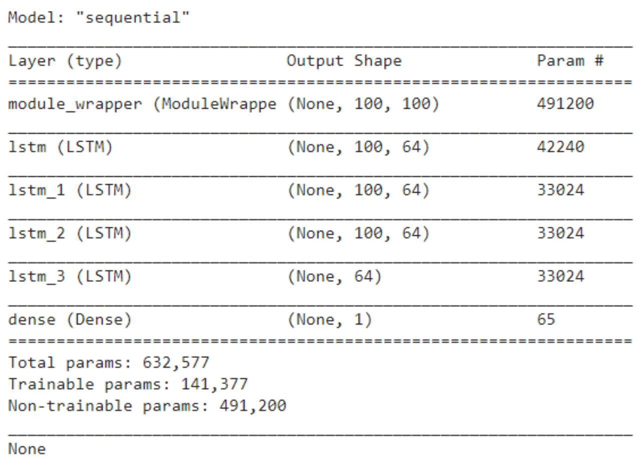


Figure A.12: Architecture - BiLSTM model



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
module_wrapper (ModuleWrappe (None, 100, 100)          491200
_________________________________________________________________
lstm (LSTM)                  (None, 100, 64)           42240
_________________________________________________________________
lstm_1 (LSTM)                (None, 100, 64)           33024
_________________________________________________________________
lstm_2 (LSTM)                (None, 100, 64)           33024
_________________________________________________________________
lstm_3 (LSTM)                (None, 64)                33024
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 632,577
Trainable params: 141,377
Non-trainable params: 491,200
_________________________________________________________________
None
```

Figure A.13: Summary - LSTM Stack model

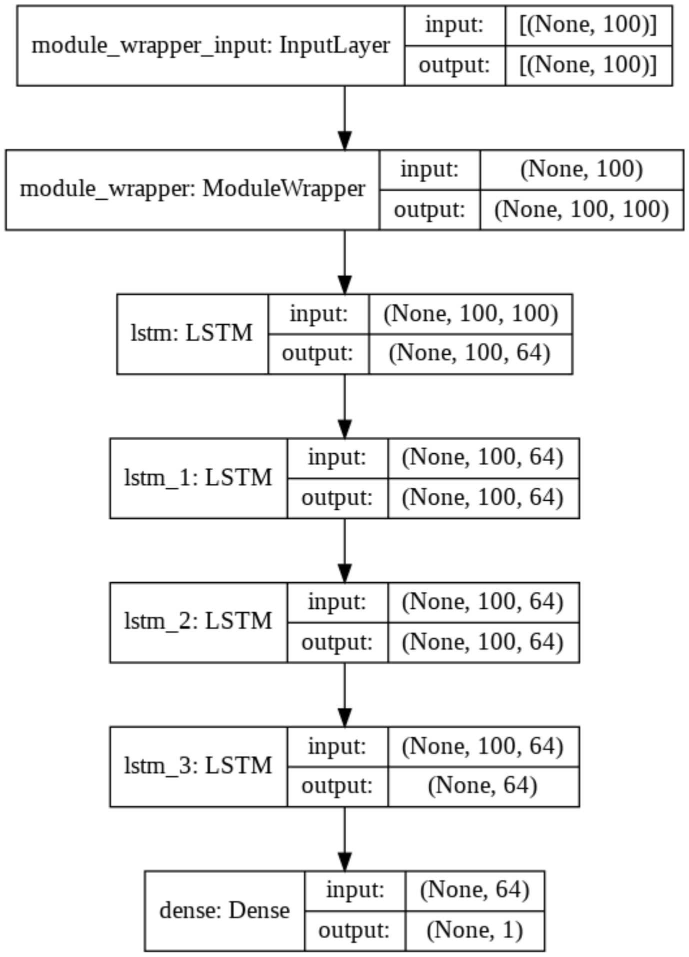


Figure A.14: Architecture - LSTM Stack model

Bibliography

[1] N. Fenton, P. Hearty, M. Neil, and L. Radlinski, "Software Project and Quality Modelling Using Bayesian Networks," pp. 1–25, 2011. DOI: 10.4018/978-1-60566-758-4.ch001.

[2] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Information and Software Technology*, vol. 54, no. 1, pp. 41–59, 2012. DOI: 10.1016/j.infsof.2011.09.002. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2011.09.002.

[3] M. Fernandez-Diego, E. R. Mendez, F. Gonzalez-Ladron-De-Guevara, S. Abrahao, and E. Insfran, "An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review," *IEEE Access*, vol. 8, pp. 166 768–166 800, 2020. DOI: 10.1109/access.2020.3021664.

[4] W. Rosa, B. K. Clark, R. Madachy, and B. Boehm, "Empirical Effort and Schedule Estimation Models for Agile Processes in the US DoD," *IEEE Transactions on Software Engineering*, vol. 5589, no. c, pp. 1–1, 2021. DOI: 10.1109/tse.2021.3080666.

[5] S. Bilgaiyan, S. Sagnika, S. Mishra, and M. Das, "A systematic review on software cost estimation in Agile Software Development," *Journal of Engineering Science and Technology Review*, vol. 10, no. 4, pp. 51–64, 2017. DOI: 10.25103/jestr.104.08.

[6] K. Srinivasan and D. Fisher, "Estimating Software Development Effort," vol. 21, no. 2, 1995.

[7] C. Prasada Rao, P. Siva Kumar, S. Rama Sree, and J. Devi, "Effort Estimation Based on Story Points in Agile Approaches : a Systematic Literature," *International Journal of Latest Trends in Engineering and Technology*, pp. 007–011, 2016.

[8] M. Jorgensen, "Relationships between Project Size, Agile Practices, and Successful Software Development: Results and Analysis," *IEEE Software*, vol. 36, no. 2, pp. 39–43, 2019. DOI: 10.1109/MS.2018.2884863.

[9] S. D. Vishnubhotla, E. Mendes, and L. Lundberg, "Understanding the perceived relevance of capability measures: A survey of Agile Software Development practitioners," *Journal of Systems and Software*, vol. 180, 2021. DOI: 10.1016/j.jss.2021.111013.

[10] K. Beck, *Manifesto for Agile Software Development, Agil*, 2021. [Online]. Available: http://www.agilemanifesto.org/.

[11] S. Wanjala Munialo and G. Muchiri Muketha, "A Review of Agile Software Effort Estimation Methods," *International Journal of Computer Applications Technology and Research*, vol. 5, no. 9, pp. 612–618, 2016. [Online]. Available: www.ijcat.com.

[12] M. Usman, E. Mendes, F. Weidt, and R. Britto, "Effort estimation in Agile Software Development: A systematic literature review," *ACM International Conference Proceeding Series*, no. September, pp. 82–91, 2014. DOI: 10.1145/2639490.2639503.

[13] StateOfAgile, "14th annual STATE OF AGILE REPORT," *Annual Report for the STATE OF AGILE*, vol. 14, no. 14, pp. 2–19, 2020. [Online]. Available: https://stateofagile.com/%7B%5C#%7Dufh-i-615706098-14th-annual-state-of-agile-report/7027494.

[14] M. Cohn, *Agile Estimating and Planning*. USA: Prentice Hall PTR, 2005.

[15] S. Dragicevic, S. Celar, and M. Turic, "Bayesian network model for task effort estimation in agile software development," *Journal of Systems and Software*, vol. 127, pp. 109–119, 2017. DOI: 10.1016/j.jss.2017.01.027. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2017.01.027.

[16] N. K. Singh, *Story Point vs Ideal Hour — It is a generation gap issue*, 2021. [Online]. Available: https://www.scrum.org/resources/blog/story-point-vs-ideal-hour-it-generation-gap-issue (visited on 03/20/2021).

[17] Atlassian, *Jira — Issue & Project Tracking Software — Atlassian*. [Online]. Available: https://www.atlassian.com/software/jira.

[18] R. Popli and N. Chauhan, "Cost and effort estimation in agile software development," *ICROIT 2014 - Proceedings of the 2014 International Conference on Reliability, Optimization and Information Technology*, vol. 3, no. 7, pp. 57–61, 2014. DOI: 10.1109/ICROIT.2014.6798284.

[19] A. Idri, M. Hosni, and A. Abran, "Systematic literature review of ensemble effort estimation," *Journal of Systems and Software*, vol. 118, pp. 151–175, 2016. DOI: 10.1016/j.jss.2016.05.016. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2016.05.016.

[20] R. Malhotra, S. Gupta, and T. Singh, "A Systematic Review on Application of Deep Learning Techniques for Software Quality Predictive Modeling," *2020 International Conference on Computational Performance Evaluation, ComPE 2020*, pp. 332–337, 2020. DOI: 10.1109/ComPE49325.2020.9200103.

[21] S. Porru, A. Murgia, S. Demeyer, M. Marchesi, and R. Tonelli, "Estimating story points from issue reports," *ACM International Conference Proceeding Series*, 2016. DOI: 10.1145/2972958.2972959.

[22] K. Moharreri, A. V. Sapre, J. Ramanathan, and R. Ramnath, "Cost-Effective Supervised Learning Models for Software Effort Estimation in Agile Environments," *Proceedings - International Computer Software and Applications Conference*, vol. 2, pp. 135–140, 2016. DOI: 10.1109/COMPSAC.2016.85.

[23] E. Scott and D. Pfahl, "Using developers' features to estimate story points," *ACM International Conference Proceeding Series*, pp. 106–110, 2018. DOI: 10.1145/3202710.3203160.

[24] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A Deep Learning Model for Estimating Story Points," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 637–656, 2018. DOI: 10.1109/TSE.2018.2792473. arXiv: 1609.00489.

[25] B. Marapelli, A. Carie, and S. M. Islam, "RNN-CNN MODEL: A bi-directional long short-term memory deep learning network for story point estimation," *CITISIA 2020 - IEEE Conference on Innovative Technologies in Intelligent Systems and Industrial Applications, Proceedings*, 2020. DOI: 10.1109/CITISIA50690.2020.9371770.

[26] S. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," *Informatica (Slovenia)*, vol. 31, pp. 249–268, 2007.

[27] L. S. Shigueoka *et al.*, "Automated algorithms combining structure and function outperform general ophthalmologists in diagnosing glaucoma," eng, *PloS one*, vol. 13, no. 12, e0207784–e0207784, Dec. 2018. DOI: 10.1371/journal.pone.0207784. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/30517157%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6281287/.

[28] 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 Computer Science*, vol. 57, pp. 772–781, 2015. DOI: 10.1016/j.procs.2015.07.474. [Online]. Available: http://dx.doi.org/10.1016/j.procs.2015.07.474.

[29] V. S. Ionescu, H. Demian, and I. G. Czibula, "Natural language processing and machine learning methods for software development effort estimation," *Studies in Informatics and Control*, vol. 26, no. 2, pp. 219–228, 2017. DOI: 10.24846/v26i2y201710.

[30] M. Choras *et al.*, "Measuring and improving agile processes in a small-size software development company," *IEEE Access*, vol. 8, pp. 78 452–78 466, 2020. DOI: 10.1109/ACCESS.2020.2990117.

[31] J. A. T. A. .-. T. T. .-. Highsmith, *Agile software development ecosystems LK - https://concordia.on.worldcat.org/oclc/5* English, 2002. [Online]. Available: http://books.google.com/books?id=EaBQAAAAMAAJ%20http://catalog.hathitrust.org/api/volumes/oclc/48906875.html%20http://proquest.safaribooksonline.com/0201760436%20http://proquestcombo.safaribooksonline.com/0201760436%20http://0-proquest.safaribooksonline.com.emu.l.

[32] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, 1st. USA: Prentice Hall PTR, 2001.

[33] K. Beck, *Extreme Programming Explained: Embrace Change*. USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[34] M. A. Ramessur and S. D. Nagowah, "Factors Affecting Sprint Effort Estimation," *Advances in Intelligent Systems and Computing*, vol. 1089, no. January, pp. 507–518, 2020. DOI: 10.1007/978-981-15-1483-8_43.

[35] M. Jørgensen, "Scope Creep or Embrace Change? A Survey of the Connections Between Requirement Changes ,Use of Agile, and Software Project Success," *Forthcoming*, 2018.

[36] E. Coelho and A. Basu, "Effort Estimation in Agile Software Development using Story Points," *International Journal of Applied Information Systems*, vol. 3, no. 7, pp. 7–10, 2012. DOI: 10.5120/ijais12-450574.

[37] L. Lindstrom and R. Jeffries, "Extreme programming and agile software development methodologies," *Information Systems Management*, vol. 21, no. 3, pp. 41–52, 2004. DOI: 10.1201/1078/44432.21.3.20040601/82476.7.

[38] L. Ben Othmane, P. Angin, H. Weffers, and B. Bhargava, "Extending the agile development process to develop acceptably secure software," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 6, pp. 497–509, 2014. DOI: 10.1109/TDSC.2014.2298011.

[39] C. Ladas, *Scrumban - Essays on Kanban Systems for Lean Software Development*. Seattle, WA, USA: Modus Cooperandi Press, 2009.

[40] K. S. Jeff Sutherland, *The 2020 Scrum Guide*, 2020. [Online]. Available: https://scrumguides.org/scrum-guide.html (visited on 11/07/2021).

[41] H. Takeuchi, I. Nonaka, and W. groups, "The new new product development game," *Harvard business review*, vol. 64, no. 1, p. 137, 1986. [Online]. Available: https://hbr.org/1986/01/the-new-new-product-development-game.

[42] I. Gat, "How BMC is scaling agile development," *Proceedings - AGILE Conference, 2006*, vol. 2006, pp. 315–320, 2006. DOI: 10.1109/AGILE.2006.33.

[43] B. Tanveer, A. M. Vollmer, and U. M. Engel, "Utilizing change impact analysis for effort estimation in agile development," *Proceedings - 43rd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2017*, pp. 430–434, 2017. DOI: 10.1109/SEAA.2017.64.

[44] D. G. Feitelson, E. Frachtenberg, and K. L. Beck, "Development and deployment at facebook," *IEEE Internet Computing*, vol. 17, no. 4, pp. 8–17, 2013. DOI: 10.1109/MIC.2013.25.

[45] M. T. Valente, "Engenharia de Software Moderna," p. 394, 2020. [Online]. Available: https://engsoftmoderna.info/.

[46] P. Pokharel and P. Vaidya, "A Study of User Story in Practice," *2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy, ICDABI 2020*, 2020. DOI: 10.1109/ICDABI51230.2020.9325670.

[47] M. Cohn, *User Stories Applied: For Agile Software Development*. USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[48] E.-M. Schön, J. Thomaschewski, and M. J. Escalona, "Agile Requirements Engineering: A systematic literature review," *Computer Standards & Interfaces*, vol. 49, pp. 79–91, 2017. DOI: https://doi.org/10.1016/j.csi.2016.08.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920548916300708.

[49] I. K. Raharjana, D. Siahaan, and C. Fatichah, "User Stories and Natural Language Processing: A Systematic Literature Review," *IEEE Access*, vol. 9, pp. 53 811–53 826, 2021. DOI: 10.1109/ACCESS.2021.3070606.

[50] N. Nunes, L. Constantine, and R. Kazman, "iUCP: Estimating Interactive-Software Project Size with Enhanced Use-Case Points," *IEEE Software*, vol. 28, no. 4, pp. 64–73, 2011. DOI: 10.1109/MS.2010.111.

[51] S. McConnell, *Software Estimation: Demystifying the Black Art*. USA: Microsoft Press, 2006.

[52] S. Boral, "Domain V: Adaptive Planning," in *Ace the PMI-ACP® exam: A Quick Reference Guide for the Busy Professional*. Berkeley, CA: Apress, 2016, pp. 201–261. DOI: 10.1007/978-1-4842-2526-4_6. [Online]. Available: https://doi.org/10.1007/978-1-4842-2526-4%7B%5C_%7D6.

[53] S. Z. Ziauddin, Shahid Kamal Tipu, "An Effort Estimation Model for Agile Software Development," *Advances in Computer Science and its Applications (ACSA)*, vol. 2, no. 1, pp. 314–324, 2012.

[54] M. Jorgensen, G. R. Bergersen, and K. Liestol, "Relations Between Effort Estimates, Skill Indicators, and Measured Programming Skill," *IEEE Transactions on Software Engineering*, 2020. DOI: 10.1109/TSE.2020.2973638.

[55] M. Usman, J. Börstler, and K. Petersen, *An Effort Estimation Taxonomy for Agile Software Development*, 4. 2017, vol. 27, pp. 641–674. DOI: 10.1142/S0218194017500243.

[56] M. Vyas, A. Bohra, C. S. Lamba, and A. Vyas, "A Review on Software Cost and Effort Estimation Techniques for Agile Development Process," *International Journal of Recent Research Aspects*, vol. 5, no. 1, pp. 1–5, 2018.

[57] E. Dantas, M. Perkusich, E. Dilorenzo, D. F. Santos, H. Almeida, and A. Perkusich, "Effort estimation in agile software development: An updated review," *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, vol. 2018-July, no. June, pp. 496–501, 2018. DOI: 10.18293/SEKE2018-003.

[58] J. Grenning, *Planning Poker or How to avoid Analysis Paralysis while Release Planning*, Vol. 3. Hawthorn Woods: Renaissance Software Consulting, 2002.

[59] O. Malgonde and K. Chari, *An ensemble-based model for predicting agile software development effort*, 2. 2019, vol. 24, pp. 1017–1055. DOI: 10.1007/s10664-018-9647-0.

[60] M. Arora, S. Chopra, and P. Gupta, "Estimation of regression test effort in agile projects," *Far East Journal of Electronics and Communications*, vol. SpecialVol, no. February 2020, pp. 741–753, 2016. DOI: 10.17654/ECSV3PII16741.

[61] R. Popli and N. Chauhan, "Estimation in agile environment using resistance factors," *Proceedings of the 2014 International Conference on Information Systems and Computer Networks, ISCON 2014*, pp. 60–65, 2014. DOI: 10.1109/ICISCON.2014.6965219.

[62] M. Owais and R. Ramakishore, "Effort, duration and cost estimation in agile software development," *2016 9th International Conference on Contemporary Computing, IC3 2016*, pp. 1–5, 2017. DOI: 10.1109/IC3.2016.7880216.

[63] W. Aslam, F. Ijaz, M. Ikramullah Lali, and W. Mehmood, "Risk Aware and Quality Enriched Effort Estimation for Mobile Applications in Distributed Agile Software Development," *Journal of Information Science and Engineering*, vol. 33, no. November, pp. 1–21, 2017. DOI: 10.6688/JISE.2017.33.6.6. [Online]. Available: http://web.a.ebscohost.com/ehost/pdfviewer/pdfviewer?vid=10%7B%5C&%7Dsid=213cf351-71b5-4f17-a717-f14e11a15065%7B%5C%%7D40sessionmgr4006.

[64] C. Prasada Rao, P. Siva Kumar, S. Rama Sree, and J. Devi, *An agile effort estimation based on story points using machine learning techniques*. Springer Singapore, 2018, vol. 712, pp. 209–219. DOI: 10.1007/978-981-10-8228-3_20. [Online]. Available: http://dx.doi.org/10.1007/978-981-10-8228-3%7B%5C_%7D20.

[65] E. Mendes, "Practitioner's Knowledge Representation," in *Springer Berlin Heidelberg*, 2014. DOI: 10.1007/978-3-642-54157-5.

[66] E. Ungan, N. Cizmeli, and O. Demirors, "Comparison of functional size based estimation and story points, based on effort estimation effectiveness in SCRUM projects," *Proceedings - 40th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2014*, pp. 77–80, 2014. DOI: 10.1109/SEAA.2014.83.

[67] A. E. D. Hamouda, "Using agile story points as an estimation technique in CMMI organizations," *Proceedings - 2014 Agile Conference, AGILE 2014*, pp. 16–23, 2014. DOI: 10.1109/AGILE.2014.11.

[68] S. M. Satapathy, A. Panda, and S. K. Rath, "Story point approach based agile software effort estimation using various SVR kernel methods," *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, vol. 2014-Janua, no. January, pp. 304–307, 2014.

[69] P. V. de Campos Souza, A. J. Guimaraes, V. S. Araujo, T. S. Rezende, and V. J. S. Araujo, "Incremental regularized Data Density-Based Clustering neural networks to aid in the construction of effort forecasting systems in software development," *Applied Intelligence*, vol. 49, no. 9, pp. 3221–3234, 2019. DOI: 10.1007/s10489-019-01449-w.

[70] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, pp. 1–12, 2013. arXiv: 1301.3781.

[71] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162.

[72] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, no. Mlm, pp. 4171–4186, 2019. arXiv: 1810.04805.

[73] J. Patihullah and E. Winarko, "Hate Speech Detection for Indonesia Tweets Using Word Embedding And Gated Recurrent Unit," *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, vol. 13, no. 1, p. 43, 2019. DOI: 10.22146/ijccs.40125.

[74] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting Similarities among Languages for Machine Translation," 2013. arXiv: 1309.4168. [Online]. Available: http://arxiv.org/abs/1309.4168.

[75] K. P. Murphy, *Probabilistic Machine Learning: An introduction*. MIT Press, 2021. [Online]. Available: probml.ai.

[76] I. G. Courville, Y. Bengio, and Aaron, *Deep Learning*. MIT Press, 2016.

[77] Abhishek, A. Dhankar, and N. Gupta, "A systematic review of techniques, tools and applications of machine learning," *Proceedings of the 3rd International Conference on Intelligent Communication Technologies and Virtual Mobile Networks, ICICV 2021*, no. Icicv, pp. 764–768, 2021. DOI: 10.1109/ICICV50876.2021.9388637.

[78] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. 2021.

[79] L. Breiman, “Random Forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/A:1010933404324. [Online]. Available: https://doi.org/10.1023/A:1010933404324.

[80] P. Argentiero, R. Chin, and P. Beaudet, “An Automated Approach to the Design of Decision Tree Classifiers,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, pp. 51–57, 1982.

[81] L. Breiman, “Bagging predictors,” *Machine Learning*, vol. 24, pp. 123–140, 2004.

[82] S. M. Satapathy and S. K. Rath, “Empirical assessment of machine learning models for agile software development effort estimation using story points,” *Innovations in Systems and Software Engineering*, vol. 13, no. 2-3, pp. 191–200, 2017. DOI: 10.1007/s11334-017-0288-z.

[83] E. Kocaguneli, T. Menzies, and J. W. Keung, “On the value of ensemble effort estimation,” *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1403–1416, 2012. DOI: 10.1109/TSE.2011.111.

[84] N. Quadrianto and Z. Ghahramani, “A very simple safe-Bayesian random forest,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 6, pp. 1297–1303, 2015. DOI: 10.1109/TPAMI.2014.2362751.

[85] S. M. Aslam, A. K. Jilani, J. Sultana, and L. Almutairi, “Feature Evaluation of Emerging E-Learning Systems Using Machine Learning: An Extensive Survey,” *IEEE Access*, vol. 9, pp. 69 573–69 587, 2021. DOI: 10.1109/access.2021.3077663.

[86] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, “Software Defect Prediction via Attention-Based Recurrent Neural Network,” *Scientific Programming*, vol. 2019, 2019. DOI: 10.1155/2019/6230953.

[87] I. Karabayir, O. Akbilgic, and N. Tas, “A Novel Learning Algorithm to Optimize Deep Neural Networks: Evolved Gradient Direction Optimizer (EVGO),” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 2, pp. 685–694, 2021. DOI: 10.1109/TNNLS.2020.2979121.

[88] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, “How to construct deep recurrent neural networks,” *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, pp. 1–13, 2014. arXiv: 1312.6026.

[89] J. Lambert, “Stacked RNNs for Encoder-Decoder Networks : Accurate Machine Understanding of Images,” *Department of Computer Science, Stanford University*, pp. 1–9, 2014. [Online]. Available: https://cs224d.stanford.edu/reports/Lambert.pdf.

[90] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A Search Space Odyssey,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017. DOI: 10.1109/TNNLS.2016.2582924. arXiv: 1503.04069.

[91] S. T. A. .-. T. T. .-. Weidman, *Deep learning from scratch : building with Python from first principles LK - https://concordia.on.worldcat.org/oclc/1119738856*, English, Sebastopol, CA, 2019. [Online]. Available: http://proquest.safaribooksonline.com/?fpi=9781492041405.

[92] A. T. A. .-. T. T. .-. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems LK*, English, Sebastopol, CA, 2019. [Online]. Available: http://proquest.safaribooksonline.com/?fpi=9781492032632.

[93] C. Dyer, M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith, "Transition-based dependency parsing with stack long short-term memory," *ACL-IJCNLP 2015 - 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Proceedings of the Conference*, vol. 1, pp. 334–343, 2015. DOI: 10.3115/v1/p15-1033. arXiv: 1505.08075.

[94] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1724–1734, 2014. DOI: 10.3115/v1/d14-1179. arXiv: 1406.1078.

[95] E. Kiperwasser and Y. Goldberg, "Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations," *Transactions of the Association for Computational Linguistics*, vol. 4, no. August, pp. 313–327, 2016. DOI: 10.1162/tacl_a_00101. arXiv: 1603.04351.

[96] G. Charlyn Pushpa Latha and M. Mohana Priya, "A review on deep learning algorithms for speech and facial emotion recognition," *International Journal of Control Theory and Applications*, vol. 9, no. 24, pp. 183–204, 2016. DOI: 10.34306/csit.v1i3.55.

[97] L. Hou, D. Samaras, T. Kurc, Y. Gao, and J. Saltz, "ConvNets with Smooth Adaptive Activation Functions for Regression," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, A. Singh and J. Zhu, Eds., ser. Proceedings of Machine Learning Research, vol. 54, Fort Lauderdale, FL, USA: PMLR, 2017, pp. 430–439. [Online]. Available: http://proceedings.mlr.press/v54/hou17a.html.

[98] E. Lewinson, *Python for finance cookbook - https://concordia.on.worldcat.org/oclc/1139921653*, Inglês, Birmingham, UK, 2020. [Online]. Available: http://www.vlebooks.com/vleweb/product/openreader?id=none%7B%5C&%7Disbn=9781789617320.

[99] Atlassian, *Showing Advanced Roadmaps custom fields in Jira*, 2021. [Online]. Available: https://confluence.atlassian.com/advancedroadmapsserver0329/showing-advanced-roadmaps-custom-fields-in-jira-1021219170.html.

[100] P. R.
bibinitperiod H. S. Christopher D. Manning, *Introduction to Information Retrieval - NLP - Stemming and Lemmatization*, 2008. [Online]. Available: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html (visited on 10/07/2021).

[101] S. Suyanto, A. Arifianto, A. Sirwan, and A. P. Rizaendra, "End-to-End Speech Recognition Models for a Low-Resourced Indonesian Language," *2020 8th International Conference on Information and Communication Technology, ICoICT 2020*, no. ii, 2020. DOI: 10.1109/ICoICT49345.2020.9166346.