computers

MDPI

*Article*

# Zero-Shot Learning for Accurate Project Duration Prediction in Crowdsourcing Software Development

**Tahir Rashid [1,2], Inam Illahi [3], Qasim Umer [2], Muhammad Arfan Jaffar [1,4], Waheed Yousuf Ramay [5] and Hanadi Hakami [6,*]**

1 Department of Computer Science, The Superior University, Lahore 54000, Pakistan; tahir@cuivehari.edu.pk (T.R.); arfan.jaffar@superior.edu.pk (M.A.J.)
2 Department of Computer Sciences, COMSATS University Islamabad, Vehari 61000, Pakistan; qasimumer@cuivehari.edu.pk
3 Department of Computing and Emerging Technologies, Emerson University, Multan 60000, Pakistan; inam.illahi@eum.edu.pk
4 Intelligent Data Visual Computing Research (IDVCR), Lahore 54600, Pakistan
5 Department of Computer Science, Cholistan University of Veterinary and Animal Sciences, Bahawalpur 63100, Pakistan; waheedramay@cuvas.edu.pk
6 Department of Software Engineering, College of Engineering, University of Business and Technology, Jeddah 21361, Saudi Arabia
* Correspondence: h.hakami@ubt.edu.sa

**Abstract:** Crowdsourcing Software Development (CSD) platforms, i.e., TopCoder, function as intermediaries connecting clients with developers. Despite employing systematic methodologies, these platforms frequently encounter high task abandonment rates, with approximately 19% of projects failing to meet satisfactory outcomes. Although existing research has focused on task scheduling, developer recommendations, and reward mechanisms, there has been insufficient attention to the support of platform moderators, or copilots, who are essential to project success. A critical responsibility of copilots is estimating project duration; however, manual predictions often lead to inconsistencies and delays. This paper introduces an innovative machine learning approach designed to automate the prediction of project duration on CSD platforms. Utilizing historical data from TopCoder, the proposed method extracts pertinent project attributes and preprocesses textual data through Natural Language Processing (NLP). Bidirectional Encoder Representations from Transformers (BERT) are employed to convert textual information into vectors, which are then analyzed using various machine learning algorithms. Zero-shot learning algorithms exhibit superior performance, with an average accuracy of 92.76%, precision of 92.76%, recall of 99.33%, and an f-measure of 95.93%. The implementation of the proposed automated duration prediction model is crucial for enhancing the success rate of crowdsourcing projects, optimizing resource allocation, managing budgets effectively, and improving stakeholder satisfaction.

**Keywords:** classification; BERT; machine learning; crowdsourcing; crowdsourcing software development; TopCoder



**Citation:** Rashid, T.; Illahi, I.; Umer, Q.; Jaffar, M.A.; Ramay, W.Y.; Hakami, H. Zero-Shot Learning for Accurate Project Duration Prediction in Crowdsourcing Software Development. *Computers* **2024**, *13*, 266. https://doi.org/10.3390/computers13100266

Academic Editor: Yan Liu

Received: 5 September 2024
Revised: 27 September 2024
Accepted: 28 September 2024
Published: 12 October 2024



## 1. Introduction

Crowdsourcing software development (CSD) platforms serve as a marketplace between clients (companies) and the crowd (developers). The CSD platform's basic role is to address clients' needs as demands and the crowd as suppliers [1]. From the client's perspective, they require maximum assurance that the crowdsourcing project must receive significant crowd participation. From the crowd's perspective, they must see tangible or intangible values in involvement, i.e., winning prize money, gaining popularity, or building technical skills [2,3].

TopCoder, the world's largest CSD platform with over 1.2 million registered developers, is trusted by tech giants, e.g., Adobe, Microsoft, and Zurich. Despite its systematic

approach from requirement extraction to project delivery, TopCoder faces a high task abandonment rate due to its dynamic and competitive environment, resulting in around 19% of projects failing to achieve satisfactory solutions [4]. Research to improve the success rate of crowd-sourced projects has focused on task scheduling [1], developer recommendations [5], and improving the reward system [6]; others have explored motivational factors [2] to increase the participation of the crowd. However, little attention has been paid to support platform moderators [7], known as TopCoder copilots, who play a crucial role in project success by managing project duration, adjusting prices, extracting and articulating requirements, answering forum questions, and fixing submissions.

Copilots are advised to consider a critical performance evaluation factor, that is, estimating the duration of the project. The project size (small, medium, large, and extra-large) must align with the CSD platform budget and be consistent with similar projects. To aid in this, TopCoder provides a set of instructions and reusable specification templates (https://docs.google.com/document/d/13szQLm1dbl5WAMDe-ddm2EHZ25ThavsI70Us7vo7lAI/edit?usp=sharing, accessed on 25 July 2024) that help estimate the duration and size of the project. For example, a small code development challenge typically takes 1 to 6 days, a medium one takes 7 to 8 days, a large one takes 9 to 10 days, and an extra-large one takes more than 10 days for final submission.

Despite these guidelines, many projects still fail, particularly small-sized projects. That is why predicting project duration is vital for preventing failure in crowdsourcing projects. It helps mitigate delays, efficiently allocate resources, comply with budgets, manage stakeholder expectations, and reduce rework. By improving predictive accuracy over time, organizations can enhance their project management practices and increase the likelihood of successful project outcomes. Investing in robust duration prediction methods is essential for minimizing project failure risk and ensuring crowdsourcing initiatives' sustained success. It aids in efficient planning, successful project management, resource allocation, budget management, risk mitigation, and stakeholder satisfaction. In addition, it improves the accuracy of future predictions, contributing to the overall reliability of crowdsourcing as a project development approach.

While copilots currently strive to predict project duration manually, this approach's limitations necessitate the adoption of automatic duration prediction models. Manual adjustments, however, often lead to inconsistent and inaccurate duration estimations, causing delays and inefficiencies. If a competition fails, copilots must re-run the development competition, impacting the overall project duration. Therefore, automatic duration prediction ensures consistency, reliability, and efficiency in managing project timelines. Such models offer greater consistency, accuracy, and efficiency, significantly reducing the risk of project failure. Investing in developing and implementing these automatic models is crucial for enhancing the success rate of crowdsourcing projects and ensuring optimal project management.

For this purpose, this study introduces a novel ML approach to predict the duration of a given CSD project automatically. The approach is tested using historical data from real software projects on TopCoder. It begins by extracting relevant attributes, i.e., requirements, technologies, platforms, project duration, and prize money. Then, it applies thorough preprocessing on the textual data using natural language processing techniques. Next, BERT [8] word embeddings are used to convert the textual information of each CSD project into vectors. These vectors and their corresponding project durations are fed into various ML algorithms. However, the zero-shot algorithms achieved the best performance.

This approach makes the following main contributions:

- To address the limitations of manual duration estimation by ensuring greater consistency, accuracy, and efficiency in managing project timelines, this paper introduces a novel ML approach to automatically predict the duration of Crowdsourcing Software Development (CSD) projects by leveraging BERT word embeddings to process and convert project-related textual information into vectors.

- Application of various ML algorithms to BERT vectors, with zero-shot algorithms demonstrating superior performance that achieves high performance with averages *A* of 92.76%, *P* of 92.76%, *R* of 99.33%, and *FM* of 95.93%.

The remaining sections of the paper are structured as follows: Section 2 gives an overview of the recent Crowdsourced Project (CSP) classification methods. The working of the proposed model is explained in Section 3. The evaluation process, analysis of results, and threats to validity are given in Section 4. Finally, conclusions and suggestions for future research are presented in Section 5.

## 2. Related Work

CSD has gained popularity as a strategy for organizations to utilize a global talent pool to complete software projects efficiently. In this perspective, many guidelines are provided. Despite these guidelines, many projects still fail, particularly small-sized projects. Figure 1 shows that out of 2438 failed projects, 2389 are small-sized. Research in this field has increased significantly over the past decade. This section reviews current state-of-the-art (SOTA) studies on CSD, including success prediction, decision recommendations, success factors, task scheduling, and quality assessment.

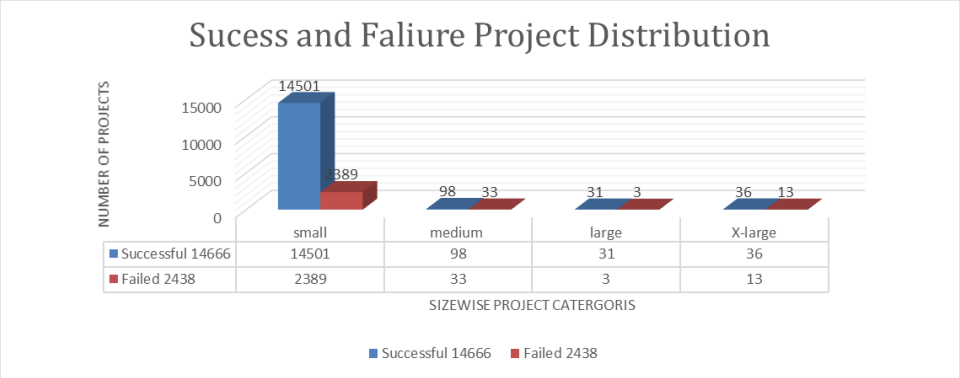


**Figure 1.** Success and failure distribution of CCSD projects.

### 2.1. Success Prediction

Illahi et al. [4,9] addressed inefficiencies in software development with a machine learning model for predicting project success. Their model achieved an average *P* of 82.64%, an *R* of 86.16%, and an *FM* of 84.36% using CNN classifiers, significantly reducing the time and effort needed to evaluate project viability. Erica Mazzola et al. [10] examined the influence of network positions, such as central and structural hole positions, on success in crowdsourcing challenges, identifying an inverted U-shaped pattern among 2479 participants. Rashid et al. [11] utilized the BERT model to predict the success of CSP, trained on historical data from TopCoder. This model showed substantial improvements in *P*, *R*, and *FM*, with increases of 13.46%, 8.83%, and 11.13%, respectively. Patel et al. [9] studied the impact of monetary rewards on crowdsourcing dynamics, finding that larger rewards increase participation but may hinder contribution appropriateness and contest success. Data from over 93,000 contest interactions on eYeka demonstrated the need to balance participant skills and experience when adjusting reward sizes.

### 2.2. Developer Recommendation

Selecting feasible developers for new project tasks or recommending appropriate jobs for workers is crucial for Competitive Crowdsource Software Development (CCSD) decision support. X. Yin et al. [12] proposed a task suggestion system for platforms like TopCoder, MTurk, CrowdFlower, and Taskcn. This system, using a probabilistic matrix factorization model, aligns tasks with developers, enhancing client satisfaction. However, gathering comprehensive worker data for quality assurance and bias correction continues to be difficult.

Yongjun et al. [13] improved task selection efficiency by using a crowdsourcing decision support framework that recommends crowds based on project characteristics. Xiaojing Yin et al. [14] proposed a task allocation strategy for cooperative software development in diverse crowdsourcing environments using HMM and GANs, achieving promising results but requiring additional empirical data. Junjie et al. [15] introduced a context-sensitive task recommendation method for testing of crowd, which significantly enhanced *P* and *R* and reduced exploration effort.

Yuen et al. [16] developed a time-sensitive task recommendation system that considers temporal changes in worker preferences, outperforming previous methods. Wang et al. [17] addressed biases in worker recommendations while crowd testing, incorporating context, and fairness elements, which significantly decreased non-productive sessions and improved bug detection. He et al. [18] designed a sustainability assessment framework for CCSD, though it lacked specific environmental metrics.

### 2.3. CCSD Project Success Factors

Dubey et al. [19] identified factors such as task category and worker rating that affect task completion in crowdsourcing, finding that well-organized tasks attract skilled developers. Messinger [20] highlighted the importance of quality communities, incentives, openness, and trust for successful software crowdsourcing. Yang et al. [21] introduced a decision-support strategy that achieved a high *P* and reduced abandonment rates. Borst et al. [22] emphasized the role of rewards in attracting participants, noting that projects can succeed without monetary compensation. Yang and Saremi [23] proposed an ideal balance between the number of contributors and prize values.

Kamar and Horvitz [24] discussed reliability concerns in task results on CCSD platforms. Machado et al. [25] highlighted the positive effects of collaboration on crowdsourcing success. Sultan et al. [26] suggested automated selection of project managers for effective crowdsourcing. Mansour et al. [27] found that CCSD benefits postgraduate students, recommending customized e-learning environments. Xu et al. [28] emphasized the importance of credibility and monetary incentives for crowdsourcers. Feng et al. [29] linked gamification with motivation, proposing self-determination as a framework. Xiaoxiao et al. [30] presented a model showing how individual thought processes influence contributions. Denisse et al. [31] found that tasks with 60–80% similarity are most likely to succeed using k-means clustering. Wang et al. [3] explored how media exposure, project length, partner count, and cross-sector collaborations affect crowd involvement in crowdsourced social innovation, identifying pathways to high and low contributions.

### 2.4. Task Scheduling

The University of Michigan's CrowdSim model [32] forecasts the failure of crowdsourced software projects. Abdullah et al. [33] identified 13 critical factors contributing to CCSD failures and developed a model for CCSD that addresses minimum task progress control. Razieh et al. [34] employed neural networks to decrease task failure by 4%, thereby improving efficiency and success rates in crowdsourced software development (CCSD). They also worked on task scheduling [35] with a multi-objective genetic framework, which significantly reduced project duration through NN-based task failure predictions.

### 2.5. CCSD Quality Assessment

Zhenghui et al. [36] worked a project rating metric for CCSD, with their clustering-based model on TopCoder. Hyun Joon Jung [37] developed a matrix factorization algorithm for project routing, predicting developer performance using SVD and PMF models, which exceeded benchmark results. Wu et al. [38] created an assessment model for software crowdsourcing, utilizing a weak min–max mechanism to ensure high-quality product delivery while promoting participation and learning within software ecosystems.

Although the reviewed research in Section 2 provides insights into various aspects of CSD, Table 1 provides the strengths and weaknesses of the existing studies. The limitations

presented in Table 1 indicates that there remains a significant gap in accurately predicting project duration. Existing studies primarily focus on task allocation, quality assessment, and success prediction but lack comprehensive solutions for duration prediction. The main goal of this study was to develop a predictive model that accurately estimates the duration of crowdsourcing software projects, thereby improving efficiency and success rates.

**Table 1.** Strengths and limitations of existing studies.

| Study | Method Used | Result | Strengths | Limitations |
| --- | --- | --- | --- | --- |
| Illahi et al. [4] | CNN Classifiers | $P = 82.64\%$, $R = 86.16\%$, $FM = 84.36\%$ | Efficient prediction of software project success | Limited generalizability beyond specific datasets |
| Mazzola et al. [10] | Network Position Analysis | Inverted U-shaped relationship identified among 2479 participants | Focus on network effects in crowdsourcing | Only addresses network position; lacks broader scope |
| Rashid et al. [11] | BERT Model | $P$, $R$, $FM$ improvements of 13.46%, 8.83%, and 11.13%, respectively | Strong performance in CSP success prediction | Limited scalability for other crowdsourcing tasks |
| X. Yin et al. [12] | Probabilistic Matrix Factorization | Enhanced task alignment with developers on platforms like TopCoder | Effective task-to-developer matching | Difficulty in gathering comprehensive worker data for bias correction |
| Yongjun et al. [13] | Crowdsourcing Decision Framework | Improved task selection efficiency | Effective crowd selection based on project characteristics | Requires more empirical data for validation |
| Junjie et al. [15] | Context-Sensitive Task Recommendation | Enhanced $P$ and $R$, and reduced exploration effort | Effective at task recommendation in testing environments | Lack of testing outside the testing crowds |
| Yuen et al. [16] | Temporal Task Recommendation | Improved worker preferences prediction | Considers time-based worker preferences | Performance dependent on accurate worker history data |
| Dubey et al. [19] | Analysis of Task Category and Worker Ratings | Well-organized tasks attract skilled developers | Insights into task categorization and skill alignment | Focuses only on small task pools |
| Sultan et al. [26] | Automated Project Manager Selection | Effective crowdsourcing management using automation | Strong project success association with manager selection | Lacks in-depth metrics for task progress monitoring |
| Razieh et al. [34] | Neural Networks | Task failure reduced by 4% | Improved efficiency and task completion rate | Focuses only on specific CCSD platforms |
| Zhenghui et al. [36] | Clustering-Based Quality Metric | Project rating on TopCoder based on clustering | Effective quality assurance mechanism | Focuses on TopCoder, limiting generalization |
| Hyun Joon Jung [37] | SVD and PMF Models | Exceeded benchmarks in developer performance prediction | Accurate project routing and developer selection | Requires significant computational resources |

## 3. Methodology

### *3.1. Overview*

The proposed approach takes a CSD project as an input and predicts the size/project duration of a given project, i.e., small/medium/large/extra-large. Notably, the size/project

duration of the projects is adopted from TopCoder [39], which has a methodology based on a number of screens. The co-pilots, who are technically strong and also take care of the projects, do this classification. The outline of the proposed method is illustrated in Figure 2. Moreover, the proposed approach is described below:

- First, CSD projects are extracted by implementing a Python script that utilizes the *TopCoder* public API (https://tcapi.docs.apiary.io/, accessed on 25 July 2024).
- Second, NLP technologies are employed to preprocess the available data, with a particular focus on the requirement documents of each project.
- Third, each CSD project is labeled as *small*, *medium*, *large*, *or extra-large* based on the duration calculated by subtracting the last submission date from the posting date. This labeling follows standard project-duration settings as instructed in Section 1.
- Fourth, the extracted attributes are utilized, and the BERT model is applied for embedding calculations to represent each CSD project as a vector.
- Fifth, it trains the ML classifiers for project duration prediction and provides the best optimal model for the project duration prediction.

The following sections offer a comprehensive explanation of the proposed approach.

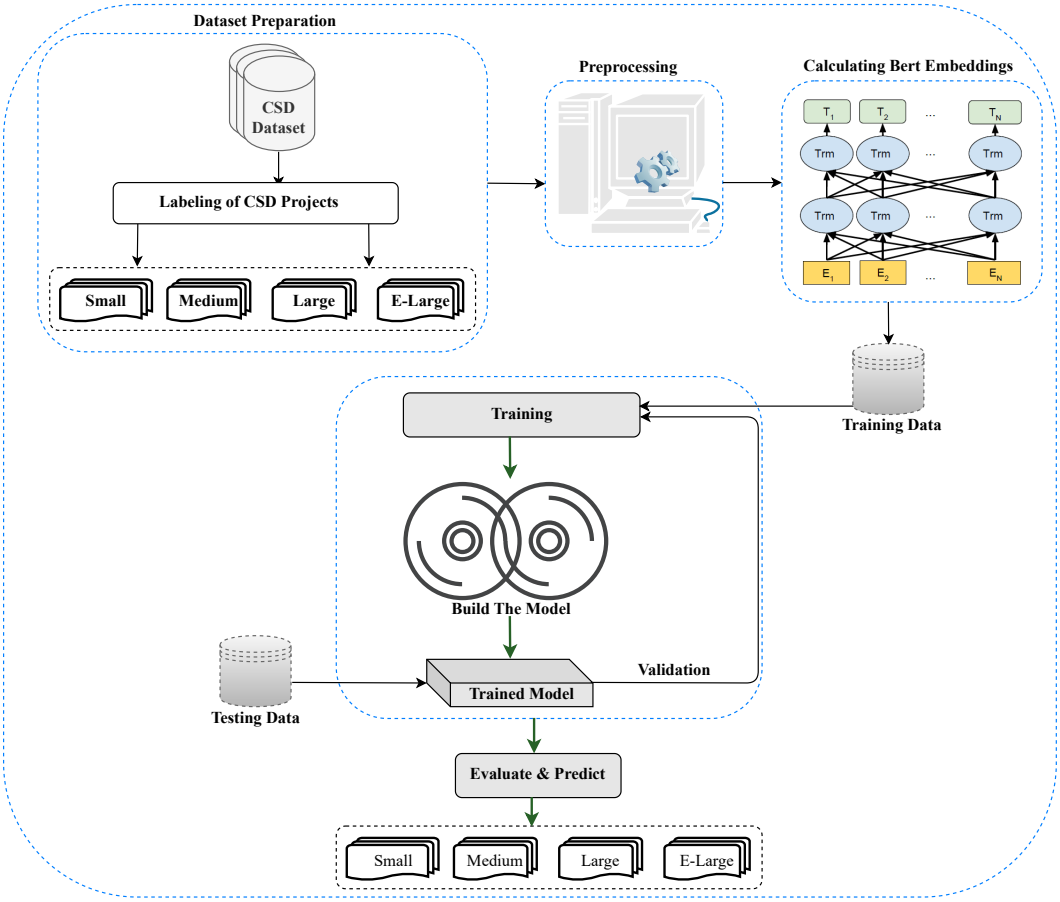


**Figure 2.** Proposed model.

### *3.2. Detailed Example*

This section provides a step-by-step guide to the proposed approach and how it performs project size/duration prediction from a real-time example from the CSD platform. The software development project (30043295) example is taken from TopCoder, which was started on (6 June 2014) and finished on (11 June 2014). Following are the details of the project.

- Project name: "*Swift-iOS8-User Notification Actions*".

- Posting date: "6 June 2014" is the date of posting of the open call for the development competition.
- Last submission date: "11 June 2014" is the last date for submitting the software solution.
- Detailed Description: (a snippet from the detailed description of the project) "*Hone your iOS development skills by implementing a new iOS 8 API in the new programming language Swift. We are challenging you to implement the new UI User Notification Action API, referred to as Quick Actions in the WWDC Keynote. You should showcase two types of actions; foreground actions (ones that take you into the application for further action) and background actions (ones that perform the action and let the user get back to what they were currently doing) with an authenticated service of some sort, like Salesforce Chatter. For this challenge, the following features are required: Native/Universal/iOS 8+, bundle ID is (deleted for privacy purpose). Implementation of background actions, implementation of an authenticated service that will be leveraged in the quick actions. UI Login Should be functional for authenticating with the service you choose to implement. Primary Screen The screen that will be used for performing your foreground action, e.g., replying to a post. User Notification Implement using the Minimal or Default Action context, this isn't very customizable, but it is where the actual functionality of the challenge happens*".
- Required technologies: "*iOS, REST, Swift, R*" are the technology constraints that are specified for developing the project.
- Prize money: "*1000, 500*" are the first and second winners prize monies.
- Platforms: "*iOS*" are the platform constraints (potentially multiple) on which the finished project will operate.
- Status: "*Cancelled-Failed Review*" is the ultimate status of the project that signifies whether the project was completed or failed to meet the deadline (the specifics of the various statuses are detailed in Section .....).

### *3.3. Problem Definition*

In this section, the example taken into account is used to illustrate how the proposed approach works. Let us take a project $p$ from a set of projects $\mathbb{P}$ and formalize it as in Equation (9).

$$p =< r, m, pt, s, rt, d, hr, t > \quad (1)$$

where $r$ is the detailed description of project $p$, $m$ is the prize money, $pt$ is the required platform, and $s$ is the project status. The required technology constraints are denoted as $rt$. The number of days needed to complete the project is represented as $d$, the number of hours required to develop the project is shown as $hr$, and $t$ indicates the type of project, categorized as small, medium, large, or extra-large. We select the variables based on the related work and baseline approaches. Notably, $d$ is computed by subtracting the starting date from the closing date of $p$, and $hr$ is converted to $d$ into hours. However, the date of the last submission of the project is provided by TopCoder.

For the example taken into account and elaborated in Section 3.2, we have

$$p_e =< r_e, m_e, pt_e, s_e, rt_e, d_e, hr_e, t_e > \quad (2)$$

where

- $r_e$ = "*Hone your iOS development skills by implementing a new iOS 8 API in the new programming language Swift. We are challenging you to implement the new UI User Notification Action API, referred to as Quick Actions in the WWDC Keynote. You should showcase two types of actions; foreground actions (ones that take you into the application for further action) and background actions (ones that perform the action and let the user get back to what they were currently doing) with an authenticated service of some sort, like Salesforce Chatter. For this challenge, the following features are required: Native/Universal/iOS 8+, bundle ID is (deleted for privacy purpose). Implementation of background actions, implementation of an authenticated service that will be leveraged in the quick actions.UI Login Should be functional for authenticating with the service you choose to implement. Primary Screen The screen that*

*will be used for performing your foreground action, e.g., replying to a post. User Notification Implement using the Minimal or Default Action context, this isn't very customizable, but it is where the actual functionality of the challenge happens*". It is a snippet of the detailed description of the project.

- $m_e$ = "*1000, 500* = 1500" is the sum of prize money of the first and second winners.
- $pt_e$ = 1 is number of platform's constrains.
- $s_e$ = 0 is the project's status, where 0 indicates the project has failed due to different reasons, e.g., zero submission, failed review, or failed screening, and 1 indicates the project is successfully completed.
- $rt_e$ = 4 is the number of technology constraints required to build the software project.
- $d_e$ = 5 is a number of days to complete the project (the project duration changes according to the size of the project).
- $hr_e$ = 120 are the number of hours calculated from number of days $d_e$.
- $t_e$ = "*small*" is the size of the project labeled based on the given project duration, i.e., between 5 and 7 days.

The suggested method categorizes the given project $p$ into *small* or *medium* or *large* or $extra - large$. The classes are indications of the project size of a given project $p$, where *small* project duration lies between 1 and 6 days, *medium* project duration is 7 to 8 days, *large* project duration is between 9 and 10 days, and $extra - large$ can take more than 10 days for its completion. The prediction of project duration might be depicted as a function $f$:

$$t = f(p) \tag{3}$$

$$t \epsilon \{small, medium, large, extra - large\}, \;\; p \epsilon \mathbb{P} \tag{4}$$

where $t$ is the outcome of the classification (e.g., *small* or *medium* or *large* or $extra - large$) and $f$ is a classification function.

### *3.4. Preprocessing*

TopCoder's API provides raw data containing many unwanted structural components, words, and symbols. Therefore, to clean the data, NLP technologies are applied to preprocess extracted data. As a first step, all the *htm/xmll* tags are removed. Furthermore, the preprocessing methods of tokenization, elimination of stop words, parts-of-speech (POS) tagging, handling negations, correcting spelling errors, identifying modifier words, inflecting words, and lemmatization are implemented. The subsequent preprocessing stages are applied to each of the extracted projects.

- Tokenization: in this step, the text is split into words and each word is called a token. Special characters like *punctuation marks* are also decomposed and converted into lowercase.
- Spell correction: in this step, the spell correction is performed using *textblob* module (https://github.com/sloria/TextBlob, accessed on 25 July 2024).
- Stop-words removal: Stop-words are commonly used words, i.e., *the*, *a*, *an*, *in*, and *are*. NLTK in Python has a list of stopwords, and in this step, all the stopwords are removed.
- POS tagging: a process of categorizing words to correspond to a particular part of speech called POS tag. Each tokenized word is assigned a POS tag in this step, especially from requirement documents.
- Replacing emails, phone numbers, and URL: *clean-text* (https://pypi.org/project/clean-text/, accessed on 25 July 2024) library is applied to remove and replace emails, phone numbers, and URL (if any) to blank spaces.
- Word morphology and lemmatization: word morphology is a method of transforming words into their singular forms. For instance, *problems* changes into *problem*. Additionally, lemmatization transforms nouns and adjectives into their base forms. For example, the term *glasses* changes into *glass*.

After preprocessing, a project $p$ can be represented as

$$p' =< rs, m, pt, s, rt, d, hr, t > \tag{5}$$

$$rs =< r_1, r_2, r_3, ....., r_n > \tag{6}$$

where $rs$ represents the terms (tokens) extracted from the requirement document of $p$ post-preprocessing. For the exemplar scenario delineated in Section 3.2, we have

$$\begin{aligned} p_e = \langle &\text{Hone, your, iOS, ..., taken, project, 1500, 1,} \\ &\text{0, 4, 5, 120, small}\rangle \end{aligned} \tag{7}$$

where *Hone*, *iOS*, ....., *project* constitute the processed terms from $p_e$.

### *3.5. Word Embeddings*

In a broader term, ML and almost all DL algorithms cannot process textual data. They require digits as inputs for classification, regression, etc. Word embedding converts the textual data to numeric from such that a machine can automatically learn the different patterns and understand the semantic and syntactic context of the data. Recent investigations [40,41] propose various word representation methods employed in NLP, such as Word2Vec [41], GloVe [42], and FastText [43]. Nevertheless, BERT [8], unveiled by Google in 2018, stands out as a robust NLP model pre-trained on an extensive text corpus. Unlike older methods like Word2Vec or FastText, BERT captures bidirectional context by pre-training on vast amounts of text data, leading to richer word representations. Its transformer architecture allows it to understand entire sentence contexts rather than just nearby words. By predicting missing words during training, BERT learns intricate semantic relationships, enhanced by its attention mechanism, which assigns importance to words. Overall, BERT offers superior performance and versatility, making it a preferred choice in project duration prediction. The embeddings calculation from Figure 2 presents the process for generating word embeddings from $p' = \langle rs, m, pt, s, rt, d, hr\rangle$. *Bert-base-uncased* "BertTokenizer" from the *Transformers* library is utilized for this purpose to generate embeddings for CSP.

### *3.6. Zero-Shot Learning Classifier*

The training and prediction processes for project duration prediction involve zero-shot models with BERT embeddings leveraging the rich semantic representations captured by BERT. In many real-life situations, gathering and labeling vast amounts of data for every potential category or idea a model might face is not practical. Allowing models to deal with fresh and unfamiliar categories with little or no extra labeled data can boost scalability and cut down the expenses linked to labeling and annotating data.

The zero-shot learning classifier leverages pre-trained data from the BERT layer to predict project duration on CSD platforms. The implementation of the zero-shor learning model is mathematically represented as follows:

$$y_i = f_\theta(x_i) \quad x_i \epsilon X, \quad y_i \epsilon Y \tag{8}$$

where $X$ and $Y$ represent the set of input samples and possible output labels (duration categories, i.e., small, medium, large, or extra-large). $f_\theta$ represents the zero-shot learning model with parameter $\theta$ that predicts the label $y_i$ for each input sample $x_i$, indicating the predicted project duration category.

The zero-shot learning classifier is trained to classify instances without direct exposure to all class instances (small, medium, large, extra-large). Using pre-trained data from the BERT layer, the algorithm effectively handles unseen project descriptions, making the model robust for various project types. This capability is crucial for CSD platforms where project attributes can vary widely and new types of projects continuously emerge.

The process begins with extracting relevant attributes from the project description, including requirements, technologies, platforms, project duration, and prize money. These attributes are denoted as

$$p = \langle r, m, pt, s, rt, d, hr, t \rangle \tag{9}$$

where $r$ is the detailed description of a project $p$, $m$ is the prize money, $pt$ is the required platform, $s$ is the project status, $rt$ represents the required technology constraints, $d$ is the number of days needed to complete the project, $hr$ is the number of hours required, and $t$ indicates the project type (small, medium, large, extra-large).

These project attributes are processed using natural language processing techniques (mentioned in Section 3.4) and converted into vectors using BERT embeddings (mentioned in Section 3.5). The vectors and their corresponding project durations are then fed into various ML algorithms. By leveraging the zero-shot learning classifier, the proposed model ensures greater consistency, accuracy, and efficiency in predicting project durations, significantly reducing project failure risk. This automated prediction model enhances overall project management in crowdsourcing initiatives, ensuring optimal resource allocation, budget management, and stakeholder satisfaction. The zero-shot learning classifiers's ability to handle unseen project descriptions without extensive retraining makes it an invaluable tool for CSD platforms like TopCoder, supporting dynamic and diverse project environments.

## 4. Evaluation

### *4.1. Research Questions*

- **RQ1** Does the proposed approach work for the project-duration prediction of CCSD projects? If yes, to what extent?
- **RQ2** Does embedding influence the proposed method of CCSD project-duration prediction?
- **RQ3** Does preprocessing influence the proposed method of CCSD for project-duration prediction?
- **RQ4** Does the proposed method outperform the machine learning classifiers for CCSD project-duration prediction?

The main research question (RQ1) assesses the accuracy of the proposed method. We compare this method with two baseline prediction algorithms: the "random prediction algorithm" and the "zero rule algorithm". These baseline algorithms are typically used when there is no established standard for comparison, especially in addressing rare or unconventional problems. The random prediction algorithm requires different actual outcome values from the training data and generates random predictions for the test data. In contrast, the zero rule algorithm predicts the most frequently occurring classification within the dataset.

To address the second research question (RQ2), we examine the influence of embedding. To achieve this, we evaluate the effectiveness of the proposed method with embedding against TF/IDF. In response to the third research question (RQ3), we explore the influence of preprocessing on the provided inputs. We contrast the performance of the proposed method with and without preprocessed text from requirement documents. The fourth research question (RQ4) assesses the effectiveness of various machine learning algorithms. This examination may indicate whether SVM surpasses other machine learning algorithms in predicting the project duration of CCSD projects.

### *4.2. Dataset*

The data about software development projects are sourced from *TopCoder*, employing the *TopCoder Rest API* (https://tcapi.docs.apiary.io, accessed on 25 July 2024) to gather essential attributes. These attributes encompass project commencement and conclusion dates, requisite technologies and platforms, project specifications, and the project's scale, which could be classified as small, medium, large, or extra-large. This information is

collected from publicly available projects up to July 2018 and stored locally. In total, there are 16,190 projects, segmented according to their sizes into small, medium, large, or extra-large categories. Out of 16,190, 5667 (35%), 4695 (29%), 3400 (21%), and 2428 (15%) are small, medium, large, and extra-large, respectively.

### *4.3. Process*

The assessment of the proposed method proceeds as follows. Initially, we retrieve the projects $\mathbb{P}$ from TopCoder, employing NLP technologies for initial processing and executing word embedding as detailed in Section 3.4. Subsequently, we employ the 10-fold cross-validation technique on $\mathbb{P}$, dividing $\mathbb{P}$ into ten subsets. Each iteration involves training on a randomly selected 90% portion of the total training set $K_{train}$, with the remaining 10% used as the testing set $K_{test}$. This process is repeated ten times to ensure a comprehensive evaluation. Notably, other evaluation techniques (i.e., 80%:20%) create dataset bias for the training and testing of the proposed approach. Consequently, we consider ten-fold cross-validation to avoid biases. Algorithm 1 trains and tests classifiers in each iteration of ten-fold cross-validation.

**Algorithm 1** Training and Evaluation of Classifiers

1: **for** each $m_i$ **do**
2: Select the training set $K_{train}$.
3: Train a Linear Regression classifier (LR) on $K_{train}$.
4: Train a Random Forest classifier (RF) on $K_{train}$.
5: Train a Support Vector Machine (SVM) classifier on $K_{train}$.
6: Train a Zero-Shot classifier (SZC) on $K_{train}$.
7: **for** each $i^{th}$ permutation of the testing set $K_{test}$ **do**
8: Use the trained classifiers (LR, RF, SVM, SZC) to predict the duration of each project from $K_{i_{test}}$.
9: **end for**
10: Compute Precision ($P$), Recall ($R$), and F-Measure ($FM$) for each classifier to compare their performances.
11: **end for**

### *4.4. Metrics*

To assess the methodology, we employ the most commonly utilized metrics for classification, namely $P$, $R$, and $FM$, defined as follows:

$$P = \frac{TP}{TP + FP} \tag{10}$$

$$R = \frac{TP}{TP + FN} \tag{11}$$

$$FM = \frac{2 * P * R}{P + R} \tag{12}$$

Here, $P$, $R$, and $FM$ represent the $P$, $R$, and $FM$ of the methodology concerning the prediction of project durations in $\mathbb{P}$. $TP$ stands for the count of projects accurately predicted with respect to a specified project duration, while $TN$ represents the count of projects correctly predicted as being out of range. $FP$ signifies the count of incorrectly predicted projects, and $FN$ denotes the count of projects incorrectly predicted as being out of range.

### *4.5. Results*

Effectiveness of the Proposed Approach

To address research question **RQ1**, we contrast the *suggested methodology* with two foundational algorithms (*random forecasting* and *zero rule*). These algorithms are employed interchangeably as benchmarks to assess the efficacy of the suggested methodology. Since there are no established methods for comparison with the proposed approach, it stands

as the inaugural method for predicting project duration, to the best of our understanding. Hence, we opt for these algorithms to evaluate the performance of the suggested approach.

The average evaluation outcomes of the *suggested methodology*, *random forecasting*, and *zero rule* are displayed in Table 2, which is evaluated across *P*, *R*, and *FM*. The Proposed Approach has the highest scores across all metrics, showcasing its robustness and balanced performance. Random Prediction comes in second, performing consistently well but slightly below the Proposed Approach in each metric. Figure 3 shows that the zero Rule shows the lowest performance among the three, indicating areas needing improvement. This visual comparison underscores the effectiveness of the proposed approach, with random prediction being a strong alternative and zero rule requiring enhancements to compete with the other methods.

**Table 2.** Comparison against Alternative Approaches.

| Approach | *P* | *R* | *FM* |
|---|---|---|---|
| **Proposed Approach** | 92.76% | 99.33% | 95.93% |
| **Random prediction** | 65.23% | 65.64% | 65.43% |
| **Zero Rule** | 82.58% | 100.00% | 90.46% |

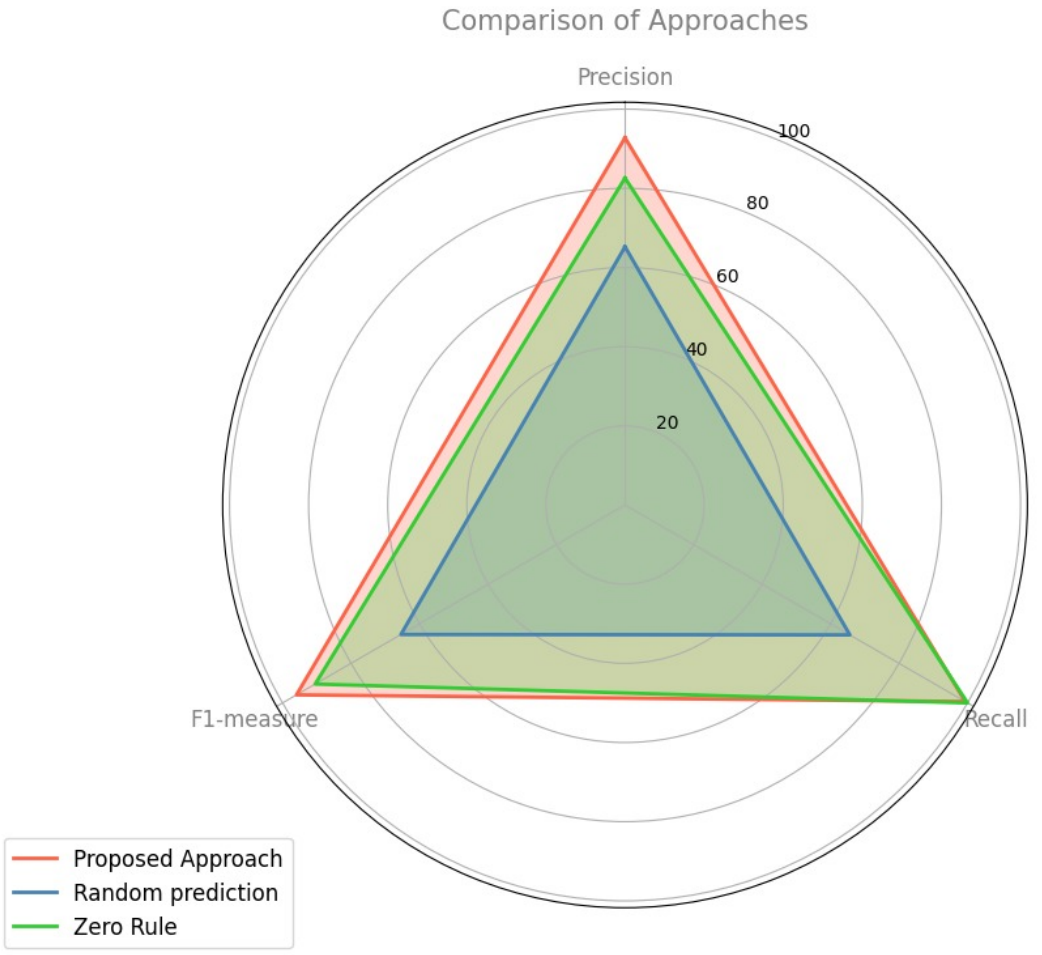


**Figure 3.** Spider Graph: Performance comparison against alternative approaches.

From Table 2, we draw the following conclusions:

- The mean *P*, *R*, and *FM* of the *proposed methodology*, *random forecasting*, and *zero rule* are (92.76%, 99.33%, 95.93%), (65.23%, 65.64%, 65.43%), and (82.58%, 100.00%, 90.46%), respectively.
- The *suggested methodology* surpasses the *random forecasting* and *zero rule* classifiers.

- Concerning *P*, the enhancement in performance of the *suggested methodology* over *random forecasting* and *zero rule* is *42.20% = (92.76% − 65.23%)/65.23%* and *12.33% = (92.76% − 82.58%)/82.58%*, respectively.
- In terms of *R*, the performance enhancement of the *suggested methodology* over *random forecasting* and *zero rule* is *51.33% = (99.33% − 65.64%)/65.64%* and *(0.67)% = (99.33% − 100.00%)/100.00%*, respectively. The reason for the decline in performance of the *suggested methodology* in *R* compared to *zero rule* is that *zero rule* consistently predicts the majority class.
- Regarding *FM*, the performance enhancement of the *suggested methodology* over *random forecasting* and *zero rule* is *46.61% = (95.93% − 65.43%)/65.43%* and *6.05% = (95.93% − 90.46%)/90.46%*, respectively.

### *4.6. Importance of Embedding*

Word embedding transforms tokens into feature vectors. To explore its influence, we contrast the effectiveness of the proposed methodology with two distinct input methodologies: embedding and TF-IDF.

The evaluation outcomes of the proposed methodology against both input methodologies are detailed in Table 3. The initial column showcases the input configurations. Columns 2–4 display the performance metrics of *P*, *R*, and *FM* under different input conditions, respectively. Each row illustrates the mean performance across each input setup.

The mean *P*, *R*, and *FM* of the proposed methodology with embedding and TF-IDF are (*92.76%*, *99.33%*, and *95.93%*) and (*90.45%*, *95.32%*, and *92.82%*), respectively. Table 3 indicates that employing the embedding technique as input enhances the overall performance (*P*, *R*, and *FM*) of the proposed methodology by *2.56%*, *4.21%*, and *3.35%*, respectively.

The performance improvement of the proposed methodology when using word embeddings over TF-IDF can be attributed to several key factors. Word embeddings capture semantic relationships between words by encoding them as dense vectors in a continuous vector space, allowing the model to understand contextual similarities and improve generalization. This results in a more efficient input data representation, reducing noise and irrelevant features. Additionally, embeddings handle synonyms and polysemy effectively by mapping semantically similar words closer together, whereas TF-IDF treats words independently. The continuous nature of embeddings enhances the model's ability to generalize in prediction tasks, leading to significant gains in performance metrics such as Precision (*P*), Recall (*R*), and F-Measure (*FM*), as evidenced by the observed improvements of 2.56%, 4.21%, and 3.35%, respectively.

**Table 3.** Importance of embeddings.

| Input | *P* | *R* | *FM* |
|---|---|---|---|
| **Embedding** | 92.76% | 99.33% | 95.93% |
| **TF-IDF** | 90.45% | 95.32% | 92.82% |

### *4.7. Importance of Preprocessing*

The textual content within projects often contains unwanted elements such as URLs, hexadecimal codes, stop-words, and punctuation. This noise is devoid of meaning and can directly impede the learning process of any M/D learning model. Hence, preprocessing textual information becomes a crucial step in machine learning, enhancing performance and reducing computational overhead.

To address research question RQ3, we conduct a comparison between the performance outcomes of the proposed methodology with and without the preprocessing step. The evaluation results of the proposed methodology under both preprocessing scenarios are presented in Table 4. From the data provided in Table 4, the following observations can be made:

- Enabling preprocessing leads to enhanced performance. It boosts the average $P$, $R$, and $FM$ of the proposed methodology by $0.34\% = (92.76\% - 92.45\%)/92.45\%$, $0.38\% = (99.33\% - 98.95\%)/98.95\%$, and $0.36\% = (95.93\% - 95.59\%)/95.59\%$, respectively.
- The likely rationale behind this enhancement is the presence of extraneous and irrelevant content within the textual data of projects, such as stop-words and punctuation. Consequently, feeding such data into the proposed methodology poses an additional burden. Hence, implementing preprocessing may enhance performance and reduce computational expenses.

Based on the preceding analysis, we deduce that the preprocessing step is critical for project-duration prediction. Disabling preprocessing would consequently diminish the proposed methodology's performance.

**Table 4.** Importance of preprocessing.

| Preprocessing | *P* | *R* | *FM* |
|---|---|---|---|
| **Enable** | 92.76% | 99.33% | 95.93% |
| **Disable** | 92.45% | 98.95% | 95.59% |

### *4.8. Proposed Classifier Verses Other Machine Learning Classifiers*

To address *RQ4*, we compare the performance of the proposed methodology with that of other ML classifiers. The evaluation outcomes of *SVM*, *LR*, *RF*, and *MNB* are displayed in Table 5.

The ridge graph (Figure 4) also presents a comparison of four machine learning algorithms (ZSC, SVM, LR, and RF) across $P$, $R$, and $FM$. Each ridge plot represents the distribution of one metric's performance for each algorithm, where the x-axis indicates the percentage values of the metric and the y-axis illustrates the density of these values. By examining the ridges, we can discern notable patterns and variations in algorithm performance. Algorithms with taller and narrower ridges signify higher $P$ and $R$ values, suggesting superior overall performance. Conversely, wider ridges with lower peaks may indicate comparatively lower $P$ and $R$, highlighting potential areas for enhancement. The positioning and spread of the ridges provide valuable insights into algorithm performance across different metrics, aiding in the identification of algorithms best suited for specific tasks, ZSC in our case.

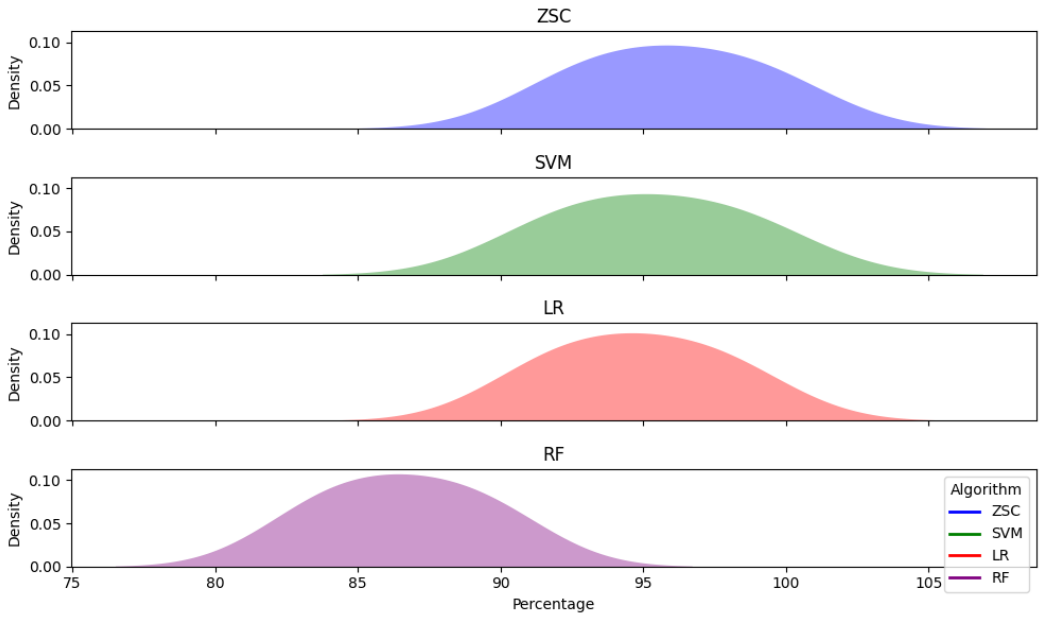


**Figure 4.** Ridge Graph: Comparison against different machine learning algorithms.

**Table 5.** Comparison against Different Machine Learning Algorithms.

| | ***P*** | ***R*** | ***FM*** |
|---|---|---|---|
| **ZSC** | 92.76% | 99.33% | 95.93% |
| **SVM** | 91.93% | 98.72% | 95.21% |
| **LR** | 91.68% | 97.94% | 94.71% |
| **RF** | 89.57% | 83.64% | 86.51% |

From Table 5 and Figure 4, the following observations can be made:

- The mean *P*, *R*, and *FM* of *ZSC*, *SVM*, *LR*, and *RF* are (92.76%, 99.33%, and 95.93%), (99.93%, 98.72%, and 95.21%), (91.68%, 97.94%, and 94.71%), and (89.57%, 83.64%, and 86.51%), respectively. The application of these classifiers indicates that *ZSC* provides the most accurate results on the given dataset.
- The ZSC algorithm outperformed SVM, LR, and RF in terms of *P*, *R*, and *FM*. Importantly, we did not use boosting algorithms to correct classification errors due to their additional computational cost. ZSC excels for several reasons. Firstly, it has superior generalization capabilities, allowing it to perform well across domains and datasets without extensive tuning. Secondly, ZSC requires less labeled data for training than SVM, LR, and RF, making it ideal for scenarios where annotated data is limited or expensive. Thirdly, ZSC is more adaptable to new or unseen classes, which is beneficial for tasks involving rapidly changing data environments.
- SVM surpasses LR and RF because it constructs a hyperplane in the feature space that maximizes the margin for most projects, except for outliers. This characteristic helps SVM generalize better on test data compared to distance-based and similarity-based algorithms like RF. Furthermore, linear SVM efficiently explores different feature combinations and performs classification with lower computational complexity than other SVM kernels. SVM also excels in long text classification scenarios, outperforming classifiers like LR, RF, and MNB.
- LR also shows better performance than RF, primarily due to its rapid training capability and effectiveness with sparse features. Although the performance difference between LR and RF is small, LR's ability to handle high-dimensional data can significantly enhance its performance on larger datasets. In contrast, RF's complexity makes it less suitable for high-dimensional features, particularly in project-duration prediction tasks.

Based on the above analysis, we conclude that the results obtained from the proposed approach are significant when compared to other machine learning classifiers.

### *4.9. Threats to Validity*

The evaluation of the proposed approach introduces potential threats to construct, internal, and external validity. While *P*, *R*, and *FM* are widely accepted metrics for assessing performance, their usage alone may overlook nuances in model effectiveness. For instance, our reliance on default parameter settings instead of exploring optimal configurations introduces uncertainty in results, as different settings could yield varied outcomes. To address internal validity concerns regarding implementation, rigorous cross-checking of results was conducted; however, there remains a possibility of undetected errors. External validity threats arise from the approach's reliance on English-language projects, potentially limiting its applicability to projects written in other languages. Moreover, the small number of projects analyzed may restrict the generalizability of findings. To mitigate these limitations, future research could explore the utilization of deep learning algorithms, which offer greater parameter flexibility and may better accommodate larger training datasets.

## 5. Conclusions

Automation has become a necessity in software project development as it can significantly reduce human errors. Scientists have developed various automated solutions

for every stage of traditional software development. However, the Collaborative Crowdsourced Software Development (CCSD) paradigm is still in its early stages and lacks automated solutions for its different stages compared to traditional software development. For example, the paradigm struggles to attract a sufficient number of developers for many CCSD projects. Moreover, many projects fail to attract proper registrants, leading to wasted time, money, and effort, ultimately threatening the success rate of CCSD projects.

TopCoder, one of the world's most famous CCSD platforms, provides a systematic approach from requirement extraction to the final deliverable project. However, project-duration estimation remains a manual task performed by the copilots of CCSD projects. Copilots on TopCoder are primarily responsible for managing the project duration, adjusting its price, responding to forum questions, and evaluating submitted solutions to ensure they meet the requirements. Copilots typically follow a traditional rule-based template to determine the size of the CCSD project (small, medium, large, or extra-large) and then assign a duration accordingly. This task consumes a significant amount of time and effort for copilots.

To address this issue, this study provided a novel machine learning-based approach that automatically predicts the duration of a given CCSD project. The proposed approach is evaluated using historical data from real software projects. The results of 10-fold cross-validation suggest that the average precision, recall, and F-measure are up to 92.76%, 99.33%, and 95.93%, respectively.

**Author Contributions:** Conceptualization, Q.U. and M.A.J.; Methodology, Q.U. and T.R.; Software, I.I. and H.H., Formal Analysis, I.I. and W.Y.R., Data Curation, T.R. and W.Y.R.; Visualization, Q.U. and I.I.; Supervision, Q.U. and H.H.; Writing—original draft, T.R. and Q.U. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Data available on request from the authors.

**Acknowledgments:** The authors acknowledge that this project was funded by the Deanship of Scientific Research (DSR), University of Business and Technology, Jeddah 21361, Saudi Arabia. The authors, therefore, gratefully acknowledge the DSR technical and financial support.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Urbaczek, J.; Saremi, R.; Saremi, M.; Togelius, J. Greedy Scheduling: A Neural Network Method to Reduce Task Failure in Software Crowdsourcing. In Proceedings of the 23rd International Conference on Enterprise Information Systems—Volume 1: ICEIS,. INSTICC, Virtual, 26–28 April 2021; SciTePress: Setúbal, Portugal, 2021; pp. 410–419. [CrossRef]
2. Illahi, I.; Liu, H.; Umer, Q.; Zaidi, S.A.H. An Empirical Study on Competitive Crowdsource Software Development: Motivating and Inhibiting Factors. *IEEE Access* **2019**, *7*, 62042–62057. [CrossRef]
3. Wang, R.; Chen, B. A Configurational Approach to Attracting Participation in Crowdsourcing Social Innovation: The Case of Openideo. *Manag. Commun. Q.* **2023**, *37*, 340–367. [CrossRef]
4. Illahi, I.; Liu, H.; Umer, Q.; Niu, N. Machine learning based success prediction for crowdsourcing software projects. *J. Syst. Softw.* **2021**, *178*, 110965. [CrossRef]
5. Zhang, Z.; Sun, H.; Zhang, H. Developer recommendation for Topcoder through a meta-learning based policy model. *Empir. Softw. Eng.* **2020**, *25*, 859–889. [CrossRef]
6. Afridi, H.G. Empirical investigation of correlation between rewards and crowdsource-based software developers. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, Argentina, 20–28 May 2017; pp. 80–81. [CrossRef]
7. de Souza, C.R.B.; Machado, L.S.; Melo, R.R.M. On Moderating Software Crowdsourcing Challenges. *Proc. ACM Hum.-Comput. Interact.* **2020**, *4*, 1–22. [CrossRef]
8. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
9. Patel, C.; Husairi, M.A.; Haon, C.; Oberoi, P. Monetary rewards and self-selection in design crowdsourcing contests: Managing participation, contribution appropriateness, and winning trade-offs. *Technol. Forecast. Soc. Chang.* **2023**, *191*, 122447. [CrossRef]
10. Mazzola, E.; Piazza, M.; Perrone, G. How do different network positions affect crowd members' success in crowdsourcing challenges? *J. Prod. Innov. Manag.* **2023**, *40*, 276–296. [CrossRef]

11. Rashid, T.; Anwar, S.; Jaffar, M.A.; Hakami, H.; Baashirah, R.; Umer, Q. Success Prediction of Crowdsourced Projects for Competitive Crowdsourced Software Development. *Appl. Sci.* **2024**, *14*, 489. [CrossRef]
12. Yin, X.; Wang, H.; Wang, W.; Zhu, K. Task recommendation in crowdsourcing systems: A bibliometric analysis. *Technol. Soc.* **2020**, *63*, 101337. [CrossRef]
13. Huang, Y.; Nazir, S.; Wu, J.; Hussain Khoso, F.; Ali, F.; Khan, H.U. An efficient decision support system for the selection of appropriate crowd in crowdsourcing. *Complexity* **2021**, *2021*, 5518878. [CrossRef]
14. Yin, X.; Huang, J.; He, W.; Guo, W.; Yu, H.; Cui, L. Group task allocation approach for heterogeneous software crowdsourcing tasks. *Peer-Peer Netw. Appl.* **2021**, *14*, 1736–1747. [CrossRef]
15. Wang, J.; Yang, Y.; Wang, S.; Chen, C.; Wang, D.; Wang, Q. Context-aware personalized crowdtesting task recommendation. *IEEE Trans. Softw. Eng.* **2021**, *48*, 3131–3144. [CrossRef]
16. Yuen, M.C.; King, I.; Leung, K.S. Temporal context-aware task recommendation in crowdsourcing systems. *Knowl.-Based Syst.* **2021**, *219*, 106770. [CrossRef]
17. Wang, J.; Yang, Y.; Wang, S.; Hu, J.; Wang, Q. Context-and Fairness-Aware In-Process Crowdworker Recommendation. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **2022**, *31*, 1–31. [CrossRef]
18. He, H.R.; Liu, Y.; Gao, J.; Jing, D. Investigating Business Sustainability of Crowdsourcing Platforms. *IEEE Access* **2022**, *10*, 74291–74303. [CrossRef]
19. Dubey, A.; Abhinav, K.; Taneja, S.; Virdi, G.; Dwarakanath, A.; Kass, A.; Kuriakose, M.S. Dynamics of software development crowdsourcing. In Proceedings of the 2016 IEEE 11th International Conference on Global Software Engineering (ICGSE), Orange County, CA, USA, 2–5 August 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 49–58.
20. Messinger, D. Elements of Good Crowdsourcing. In Proceedings of the 3rd International Workshop in Austin, Austin, TX, USA, 17 May 2016.
21. Yang, Y.; Karim, M.R.; Saremi, R.; Ruhe, G. Who should take this task? Dynamic decision support for crowd workers. In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Ciudad Real, Spain, 8–9 September 2016; pp. 1–10.
22. Borst, I. Understanding Crowdsourcing: Effects of Motivation and Rewards on Participation and Performance in Voluntary Online Activities. Number EPS-2010-221-LIS; 2010. Available online: https://repub.eur.nl/pub/21914/EPS2010221LIS9789058922625.pdf (accessed on 27 September 2024).
23. Yang, Y.; Saremi, R. Award vs. worker behaviors in competitive crowdsourcing tasks. In Proceedings of the 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Beijing, China, 22–23 October 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 1–10.
24. Kamar, E.; Horvitz, E. Incentives for truthful reporting in crowdsourcing. In Proceedings of the AAMAS. Citeseer, Valencia Spain, 4–8 June 2012; Volume 12, pp. 1329–1330.
25. Machado, L.; Melo, R.; Souza, C.; Prikladnicki, R. Collaborative Behavior and Winning Challenges in Competitive Software Crowdsourcing. *Proc. Acm -Hum.-Comput. Interact.* **2021**, *5*, 1–25. [CrossRef]
26. Al Haqbani, O.; Alyahya, S. Supporting Coordination among Participants in Crowdsourcing Software Design. In Proceedings of the 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), Las Vegas, NV, USA, 25–27 May 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 132–139.
27. Alabdulaziz, M.S.; Hassan, H.F.; Soliman, M.W. The effect of the interaction between crowdsourced style and cognitive style on developing research and scientific thinking skills. *Eurasia J. Math. Sci. Technol. Educ.* **2022**, *18*, em2162. [CrossRef]
28. Xu, H.; Wu, Y.; Hamari, J. What determines the successfulness of a crowdsourcing campaign: A study on the relationships between indicators of trustworthiness, popularity, and success. *J. Bus. Res.* **2022**, *139*, 484–495. [CrossRef]
29. Feng, Y.; Yi, Z.; Yang, C.; Chen, R.; Feng, Y. How do gamification mechanics drive solvers' Knowledge contribution? A study of collaborative knowledge crowdsourcing. *Technol. Forecast. Soc. Chang.* **2022**, *177*, 121520. [CrossRef]
30. Shi, X.; Evans, R.D.; Shan, W. What Motivates Solvers' Participation in Crowdsourcing Platforms in China? A Motivational–Cognitive Model. *IEEE Trans. Eng. Manag.* **2022**, *71*, 12068–12080. [CrossRef]
31. Mejorado, D.M.; Saremi, R.; Yang, Y.; Ramirez-Marquez, J.E. Study on patterns and effect of task diversity in software crowdsourcing. In Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Bari, Italy, 5–7 October 2020; pp. 1–10.
32. Saremi, R.; Yang, Y.; Vesonder, G.; Ruhe, G.; Zhang, H. Crowdsim: A hybrid simulation model for failure prediction in crowdsourced software development. *arXiv* **2021**, arXiv:2103.09856.
33. Khanfor, A.; Yang, Y.; Vesonder, G.; Ruhe, G.; Messinger, D. Failure prediction in crowdsourced software development. In Proceedings of the 2017 24th Asia-Pacific Software Engineering Conference (APSEC), Nanjing, China, 4–8 December 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 495–504.
34. Urbaczek, J.; Saremi, R.; Saremi, M.L.; Togelius, J. Scheduling tasks for software crowdsourcing platforms to reduce task failure. *arXiv* **2020**, arXiv:2006.01048.
35. Saremi, R.; Yagnik, H.; Togelius, J.; Yang, Y.; Ruhe, G. An evolutionary algorithm for task scheduling in crowdsourced software development. *arXiv* **2021**, arXiv:2107.02202.
36. Hu, Z.; Wu, W.; Luo, J.; Wang, X.; Li, B. Quality assessment in competition-based software crowdsourcing. *Front. Comput. Sci.* **2020**, *14*, 146207. [CrossRef]

37. Jung, H.J. Quality assurance in crowdsourcing via matrix factorization based task routing. In Proceedings of the 23rd International Conference on World Wide Web, Seoul, Republic of Korea, 7–11 April 2014; pp. 3–8.
38. Wu, W.; Tsai, W.T.; Li, W. An evaluation framework for software crowdsourcing. *Front. Comput. Sci.* **2013**, *7*, 694–709. [CrossRef]
39. Blohm, I.; Zogaj, S.; Bretschneider, U.; Leimeister, J.M. How to Manage Crowdsourcing Platforms Effectively? *Calif. Manag. Rev.* **2018**, *60*, 122–149. [CrossRef]
40. Sarzynska-Wawer, J.; Wawer, A.; Pawlak, A.; Szymanowska, J.; Stefaniak, I.; Jarkiewicz, M.; Okruszek, L. Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Res.* **2021**, *304*, 114135. [CrossRef]
41. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. *arXiv* **2013**, arXiv:1301.3781.
42. Pennington, J.; Socher, R.; Manning, C.D. Glove: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1532–1543.
43. Joulin, A.; Grave, E.; Bojanowski, P.; Mikolov, T. Bag of tricks for efficient text classification. *arXiv* **2016**, arXiv:1607.01759.