Acceptance Criteria Validation in Agile Projects Using AI and NLP Techniques

Ana Carla Gomes da Silva[1][a], Afonso Sales[1][b] and Fabio Gomes Rocha[2][c]
[1]*School of Technology, PUCRS, Porto Alegre, RS, Brazil*
[2]*Federal University of Sergipe, UFS, Aracaju, SE, Brazil*

Keywords: Artificial Intelligence, Software Requirements, Machine Learning Models, Software Development.

Abstract: In agile software development, user stories and their acceptance criteria play a critical role in ensuring alignment between stakeholder expectations and system functionality. However, the manual validation of these criteria is often labor-intensive and prone to bias. This study investigates the application of Artificial Intelligence (AI) techniques, particularly Natural Language Processing (NLP) and Machine Learning (ML), to automate the analysis and validation of user stories. Using a dataset of user stories collected from academic and industry projects, we trained and evaluated four ML algorithms: Multilayer Perceptron (MLP), Support Vector Machine (SVM), Naive Bayes, and Random Forest. The models were assessed for their ability to classify acceptance criteria accurately and efficiently. Our findings demonstrate the potential of AI to enhance the validation process, achieving over 60% accuracy in certain cases, with SVM standing out as the most robust algorithm. This research highlights the transformative role of AI in improving software requirements analysis and lays the foundation for future innovations in automated validation and quality assurance in agile environments.

1 INTRODUCTION

In the context of software development, an accurate and comprehensive understanding of user requirements is fundamental to the success of a project (Johnson et al., 2023; Smith and Jones, 2022). A popular approach to capturing and describing these requirements in a contextualized and accessible manner is the use of user stories (North, 2006; Erdogmus et al., 2005). User stories are short and simple descriptions of a system's functionality written from the perspective of the person who desires the new capability, usually in a standard format such as "*As a [type of user], I want [goal] so that I can [benefit]*". These stories are often used in agile methodologies, such as *Scrum*, to guide development and ensure that the final product meets stakeholders' expectations.

Each user story includes acceptance criteria, which are necessary conditions for the story to be considered complete. These criteria serve to validate whether the functionality's behavior meets the proposed objectives and are often used as a basis for automation. The analysis and validation of these criteria are crucial, as they directly impact the quality and efficiency of the developed software (Smith and Jones, 2022; Melegati et al., 2020).

However, the manual analysis of these stories and their acceptance criteria can be labor-intensive and prone to bias, justifying the search for more efficient and objective approaches. In this context, Artificial Intelligence (AI) emerges as a promising ally, offering advanced capabilities to understand and process large volumes of data in an automated and accurate manner.

This study aims to explore the potential of AI in the analysis of user stories, investigating how AI models can be trained and applied to identify patterns, predict quality, and suggest improvements in these stories, with a focus on acceptance criteria as the main reference. In this context, this study seeks to answer the following research question: *"How can the application of machine learning models and natural language processing improve the accuracy and efficiency in validating acceptance criteria in user stories within agile software development projects?"*.

Using a dataset composed of user stories extracted

[a] https://orcid.org/0000-0002-5185-0481
[b] https://orcid.org/0000-0001-6962-3706
[c] https://orcid.org/0000-0002-0512-5406

Gomes da Silva, A. C., Sales, A. and Rocha, F. G.
Acceptance Criteria Validation in Agile Projects Using AI and NLP Techniques.
DOI: 10.5220/0013276400003929

In *Proceedings of the 27th International Conference on Enterprise Information Systems (ICEIS 2025) - Volume 2*, pages 176-184
ISBN: 978-989-758-749-8; ISSN: 2184-4992

from real projects, the study employs machine learning techniques to develop and evaluate AI models (Silva et al., 2020; Li et al., 2021). Additionally, works such as *"Understanding Software Requirements"* (Wiegers, 2003) and *"How to Evaluate BDD Scenarios' Quality?"* (Oliveira et al., 2019) provided a comprehensive overview of the topic and served as important references for this study.

Among the specific objectives of this study, it is expected not only to demonstrate the effectiveness of these models but also to highlight their ability to understand and interpret user requirements, particularly the acceptance criteria. Furthermore, the study aims to compare the performance of various machine learning algorithms (such as *Multilayer Perceptron* - MLP (Mohanty, 2019), *Support Vector Machine* - SVM (Cortes and Vapnik, 1995), *Naive Bayes* (Wang and Manning, 2012), and *Random Forest* (Breiman, 2001)) to determine which model offers the best balance between accuracy and generalization capability across different datasets of user stories. Additionally, the study seeks to investigate and implement methods to minimize bias and errors in the automated validation of user stories, ensuring that the models are fair and effective. Through this research, the aim is to enhance software development practices, providing valuable insights to the academic and professional communities on the transformative role of AI in software requirements analysis.

The remainder of this paper is organized as follows: Section 2 presents the related work, discussing previous studies and positioning the research within the current literature; Section 3 describes the methodology used, detailing the stages of data collection, preprocessing, and the application of machine learning models; Section 4 presents the results obtained, comparing the effectiveness of the different algorithms tested; Section 5 discusses the possible threats to the validity of the study, addressing limitations and points of attention; and finally, Section 6 offers the final considerations, summarizing the conclusions and suggesting directions for future work.

2 RELATED WORKS

The accurate interpretation of user stories and their associated criteria plays a central role in the success of software development projects (Pressman, 2015). User stories, commonly structured in brief narrative formats, serve as bridges between the needs of end-users and the technical functionalities that developers must implement. These stories help ensure that the developed software effectively aligns with the expectations of the *stakeholders*, facilitating planning and task prioritization during agile development cycles (Conrado, 2012).

In this work, we utilize the research of Sabrina Marczak (Oliveira and Marczak, 2018), which conducts a detailed study to identify the main challenges faced by development teams. The study highlights that, although agile methods are effective in improving collaboration and flexibility, adapting requirements to constant market changes remains a significant problem. It emphasizes the importance of robust communication and documentation practices, as well as tools that facilitate requirements traceability throughout the project life cycle. These findings provide a solid foundation for enhancing requirements management strategies in agile environments, contributing to the delivery of software products more aligned with *stakeholder* expectations.

Complementing our approach, the automated generation of test inputs from user stories and acceptance criteria, as presented by Nguyen *et al.* (Nguyen et al., 2020), emerges as an innovative technique. While our research focuses on applying machine learning algorithms, such as *Multilayer Perceptron*, *Support Vector Machine*, *Naive Bayes*, and *Random Forest*, to evaluate the compliance of user stories with predefined criteria, Nguyen *et al.* (Nguyen et al., 2020) propose a methodology for inferring behavioral models from these stories and subsequently generating automated test scenarios. This combination of techniques can potentially offer a more robust solution for software development, where validation and automated testing complement each other to ensure quality and adherence to *stakeholder* requirements.

3 METHODOLOGY

The methodology used in this research was delineated following the steps shown in Figure 1: *Initial Data Collection*, *Sample Expansion*, *Data Preprocessing*, and *Application of Machine Learning Models*.

1. **Initial Data Collection.** We gathered a set of 28 user stories and their acceptance criteria to validate the possibility of analysis. These stories were obtained from projects carried out by students in undergraduate courses at the university. The collection of data from academic projects highlights the importance of collaborative approaches, such as Challenge-Based Learning, in preparing students for real-world scenarios (Chanin et al., 2018).
2. **Sample Expansion.** Then, we expanded our database to include a sample of 176 user stories

Figure 1: Methodological stages of the research.

with one or more acceptance criteria, resulting in a total of 204 acceptance scenarios in the training set. These data were collected from real industry projects and academic projects in a technological and scientific park. To ensure the consistency and quality of the data used, the user stories were labeled according to previously established criteria. The labeling process was conducted by me, the author of this work, which provided me with familiarity with both the project requirements and agile methodology, ensuring the accuracy of the labels.

3. **Data Preprocessing.** We applied preprocessing techniques to improve data quality, including text normalization and removal of irrelevant information. This step is crucial, as the quality of the data set directly impacts the accuracy of the results. Then, the data were divided into training and test sets.
4. **Application of Machine Learning Models.** We used four machine learning algorithms (*Multilayer Perceptron* - MLP; *Support Vector Machine* - SVM; *Naive Bayes*; and *Random Forest*) to analyze the user stories. It is important to note that these models are classifiers, designed to categorize the acceptance criteria, not to interpret them. These models were trained and evaluated for their ability to interpret software requirements.

3.1 Model Configurations

In this section, we present the configurations and specific parameters used to train each machine learning model. The configurations were selected based on the nature of the classification problem and preliminary experiments conducted to optimize the models' performance.

To analyze the acceptance criteria, the models were trained and evaluated for their ability to classify software requirements. The configurations and parameters for each model were deliberately kept simple to facilitate an initial assessment.

These configurations aim to provide consistent results and facilitate comparisons among the models.

1. **Multilayer Perceptron (MLP):**
   - **Hidden Layers (70, 80, 100).** Different layer sizes were defined to enhance the model's capacity to capture complex variations in the data without an excessive number of neurons, thus avoiding overfitting.
   - **Maximum Number of Iterations (700).** Selected to ensure the model had sufficient time to converge and find a stable solution during training.
   - **L2 Regularization (alpha=1e-8).** Used to prevent overfitting by smoothing the impact of high weights in the network.
   - **Learning Rate invscaling.** This configuration reduces the learning rate as iterations progress, allowing for finer adjustments in the later stages of training.
2. **Support Vector Machine (SVM):**
   - **Linear Kernel.** Chosen to ensure simplicity in the model's initial tuning and to establish a baseline for performance evaluation.
3. **Naive Bayes (MultinomialNB):**
   - **Default Parameters (alpha=1.0).** Kept to assess the algorithm's performance in its most common configuration, serving as a comparison point with the other models.
4. **Random Forest:**
   - **Maximum Depth (2).** Limiting the tree depth was chosen to avoid excessive specialization of the model to the training data.
   - **Random Seed (random_state=0).** Used to ensure experiment reproducibility, generating the same results under identical conditions.

In order to ensure a robust evaluation, we applied the *k-fold Cross-Validation* technique to the trained models. This technique involves splitting the dataset into multiple parts and repeatedly training and testing the model using different data combinations for each iteration.

We adopted a standardized framework for accuracy calculation, which can be adapted to each algorithm. In Algorithm 1, we present an example using SVM to illustrate the accuracy calculation process.

1. **Accuracy Evaluation with k-Fold.** The *"cross_val_score"* function applies the *"mlp"* model (or any other chosen algorithm) to the training dataset *"x_train"* and *"y_train"* across multiple splits (*k-folds*). Each fold evaluates the model's accuracy, enabling a more robust assessment and reducing the likelihood of overfitting.
2. **Accuracy Metrics Summary.** After computing the accuracies for each fold, the code displays the individual accuracies, the mean and the standard deviation, providing a detailed view of the model's behavior across different training set samples.

To adapt the code to other algorithms, simply replace the *"mlp"* model with the desired model, such as *"svm"*, *"naive_bayes"*, or *"random_forest"*. This way, the pseudocode serves as a reusable framework, requiring only the algorithm's name to be changed.

In addition, to clarify the process used in this study, we followed the methodology as follow:

1. **Data Splitting:**
   - The data is divided into k equal parts, with k being a parameter chosen by the researcher that can vary depending on the size of the dataset and the specific problem, with $k = 5$ or $k = 10$ being commonly used values.
   - Each part is used as a test set once, while the other parts are used to train the model, ensuring that each subset is used exactly once as a test set.
2. **Training and Testing:**
   - The model is trained k times, once for each different test part.
   - The accuracy is calculated for each test round.
3. **Mean of Accuracies:**
   - The accuracies of all the rounds are measured to obtain the final accuracy.
   - After performing the k iterations, as detailed by Bishop (Bishop, 2006), the accuracy is calculated as the mean of the accuracies obtained in each iteration. Mathematically, if we denote the accuracy in the i-th iteration as A_i, the final accuracy A_T is given by:

$$A_T = \frac{1}{k} \sum_{i=1}^{k} A_i$$

   - This method ensures that all observations in the dataset are used for both training and testing, providing a more reliable assessment of the model's performance.

```
# Importing the required libraries
from sklearn.model_selection import
    train_test_split
from sklearn.model_selection import
    cross_val_score

# Accuracy eval. of the chosen alg.
# (replace 'mlp' to apply other model)

# Using cross-validation to measure
    accuracy across k-folds
val_scores = cross_val_score(mlp, x_train,
     y_train, cv=5)

# Display the final accuracy and
    performance across k-folds

# Overall accuracy on the test set
print("Accuracy:", mlp.score(x_test,
    y_test))

# Accuracies of each fold
print('Accuracy across k-folds:',
    val_scores)

# Mean and standard deviation of
    accuracies
print('Mean: {:.2f} | Standard Deviation:
    {:.2f}'.format(np.mean(val_scores), np
    .std(val_scores)))
```

Algorithm 1: Pseudocode for accuracy calculation.

This method ensures that all observations in the dataset are used for both training and testing, providing a more reliable evaluation of the model's performance.

The *k-fold Cross-Validation* is an effective technique for assessing the performance of Machine Learning models, offering a more stable measure of their effectiveness. While accuracy is an important metric, others such as precision and recall could also be calculated to provide additional insights into the model's behavior. Table 1 (see Section 4) summarizes the accuracies obtained in this study, offering an overview of the relative performance of each algorithm.

For exemplification purposes, we will show step by step how the accuracy of the model used with the SVM technique was calculated:

1. We divided the data into k equal parts. In this study, we used $k = 5$, meaning the data was split into 5 parts (*folds*).
2. In each iteration, we used 4 of the 5 parts to train the model and the remaining part for testing. In each iteration, a different part is used for testing, ensuring that all parts are used for testing once.

3. After each iteration, we calculate the accuracy, which represents the proportion of correct predictions made by the model. This results in 5 accuracy values, one for each training and testing performed.
4. Finally, we calculate the mean of these 5 accuracies. The final accuracy is the mean of these accuracies, which is the indicator presented in Table 1 and used as the basis for validating the model.

3.2 User Stories Standardization

The user should have the option to add notes or attachments to a ticket. Each note can have visibility, either PUBLIC (accessible by anyone accessing the ticket) or TECHNICAL (viewable only by Administrators).

After data collection, it is checked whether the data have the necessary pattern for the AI to perform the analysis. If they are correct, the next steps are followed. Otherwise, the stories must follow the pattern demonstrated in Figure 2, so the correct writing would be as described in the following subsection.

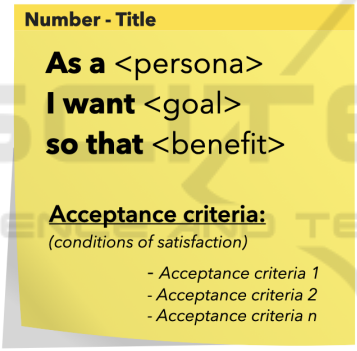

Figure 2: Structure of all criteria to be analyzed by AI.

As a user, I want to have the option to add notes or attachments to a ticket so that I can have visibility.

- *Criterion 1*: The note must have 2 visibility options: PUBLIC or TECHNICAL.
- *Criterion 2*: PUBLIC notes are accessible by anyone accessing the ticket.
- *Criterion 3*: TECHNICAL notes are viewable only by Administrators.

The subsequent step involves preprocessing the text using the techniques outlined in the following subsection.

3.3 Lowercase Transformation

All characters in the user stories were converted to lowercase. This approach standardizes the text, avoiding discrepancies between words written in uppercase and lowercase, providing uniformity to the dataset.

3.4 Stopwords Removal

Stopwords are common words that usually do not significantly contribute to the meaning of a sentence and can, therefore, be removed without impairing text comprehension. Examples include articles, prepositions, and conjunctions. For example, “the”, “of”, and “and” are stopwords in English.

3.5 Special Character Cleaning

Accents, special characters, and unnecessary punctuation were removed from the processed texts. This cleaning aims to simplify the text, facilitating comparison and subsequent analysis.

3.6 Tokenization and TF-IDF Vectorization

Tokenization divides the text into smaller units, such as words or phrases, while TF-IDF vectorization assigns values to these units based on their importance in the global context of the dataset. For example, the expression “TF-IDF” stands for “*Term Frequency-Inverse Document Frequency*”. It highlights the relevance of a word in relation to a specific document within a broader collection of documents.

As a final step, a machine learning model is applied to perform the training and validation of the model. As explained earlier in this research, the *k-fold Cross-Validation* algorithm was used to validate the trained model.

This methodological approach allowed us to understand the performance of machine learning models in software requirements validation, providing excellent indicators for selecting the most suitable model for this task.

After presenting an example of how user stories and their acceptance criteria are processed by AI, we applied all the demonstrated steps in this research. For the application of Machine Learning models, we trained four different algorithms to validate the user stories: *Multilayer Perceptron* (MLP) (Mohanty, 2019); *Support Vector Machine* (SVM) (Cortes and Vapnik, 1995); *Naive Bayes* (Wang and Manning, 2012); and *Random Forest* (Breiman, 2001). These algorithms were selected due to their relevance and applicability to classification problems, such as the one addressed in this research. We present the justifications for the selection of each algorithm as follow:

1. **Multilayer Perceptron (MLP).** It has the ability to learn complex relationships in non-linear data, which is particularly useful for classification problems involving multiple acceptance criteria.
2. **Support Vector Machine (SVM).** It is efficient in finding optimal separating hyperplanes, making it effective for binary classification tasks, such as determining whether acceptance criteria are met or not.
3. **Naive Bayes.** It is a simple yet efficient algorithm for probability-based classification, making it useful in scenarios where variables are independent — a reasonable assumption for some characteristics of acceptance criteria.
4. **Random Forest.** Its ability to handle complex feature interactions makes it ideal for validating user stories with intricate acceptance criteria. Additionally, Random Forest's feature importance analysis offers insights into which criteria are most influential, enhancing both model interpretability and performance.

Initially, we used the processed training data to allow the model to learn the characteristics to be considered. The output of this process is a vector of 10 positions, where each position represents a classification: 1 indicates that the criterion was met, and 0 indicates that it was not. The validation criteria, as defined by Oliveira and Marczak (Oliveira and Marczak, 2018), encompass various aspects and are described below, along with examples of how each criterion is applied:

- **Criterion 1 - Identification of the Value of the Feature File or Result by Description.** This criterion analyzes whether the description meets the expected business outcome, represented by the format of the user story.

  Example: If the user story states, *"As a user, I want to log in to access my account"*, this should correspond to a feature that allows logging in, reflecting the expected outcome.

- **Criterion 2 - Verification of the Absence of any Scenario in the Feature File.** Evaluates whether the user story covers all necessary scenarios for the system being developed.

  Example: If the story does not mention error scenarios, such as *"What happens if the user enters an incorrect password?"*, this may indicate a gap.

- **Criterion 3 - Ensuring that the Scenario Contains All Necessary Information.** Verifies whether the scenarios require additional information for complete understanding, allowing any team member to follow the steps independently.

  Example: A scenario that states, *"The user clicks the button and receives a message"* should include details about what the message says and which button it is.

- **Criterion 4 - Verification of Comprehensible Steps in the Scenario.** Analyzes whether there is excessive essential information to validate the behavior of the acceptance criteria.

  Example: A very long and complicated scenario that mixes multiple actions can hinder understanding. Ideally, each scenario should address a single action clearly.

- **Criterion 5 - Ensuring that the Scenario Represents a Uniquely Identifiable Action by the Title.** Focuses on the uniqueness of the action in the scenario represented by *"When"*, aligning with the title.

  Example: If the title is *"User Login"*, the scenario should focus only on the steps describing the login and not mix in other functionalities.

- **Criterion 6 - Identification of Results or Verifications in the Scenario Titles and Markings.** Assesses whether the scenario checks, present in *"Then"*, align with the purpose expressed in the title.

  Example: If the scenario ends with *"Then the user should see the homepage"*, this should correspond to what the title indicates.

- **Criterion 7 - Adherence to Gherkin Keywords and Natural Order.** Validates the integrity of Gherkin rules, ensuring that the steps represent preconditions, action, and results in the established order.

  Example: In a scenario, there should be a *"Given"*, followed by a *"When"*, and finally a *"Then"*.

- **Criterion 8 - Correct Application of Business Terms, Including Appropriate Actors.** Ensures that business terms are consistent, facilitating understanding for both technical and non-technical members.

  Example: The story should clearly identify who the user is, such as *"As an administrator, I want to..."*.

- **Criterion 9 - Expression of *"What"* in a Declarative Manner in the Step.** Questions whether the *"what"* step is focused on the result rather than explaining *"how"* the result is achieved.

  Example: The phrase should focus on *"The user should see a success message"* instead of describing the process that leads to it.

- **Criterion 10 - Possibility of Different Interpretations Due to Vagueness or Misleading Statements.** Seeks to strike a balance between clearly expressing the action while avoiding ambiguities that may confuse the team.

  Example: Phrases like *"The system should behave correctly"* should be avoided as they are vague. Instead, one should specify *"The system should display an error message if the password is incorrect"*.

In the evaluation of the models, we adopted two different approaches. First, we assessed the overall performance of the models in predicting whether a user story meets all the established criteria. Then, we focused on identifying how many individual criteria each model correctly identified, allowing for a detailed analysis of performance in each aspect of the user stories.

This comprehensive methodological approach allows us to understand the performance of Machine Learning models in validating software requirements, providing valuable insights for selecting the most appropriate model for this task.

4 RESULTS

This study compared four Machine Learning algorithms to assess their effectiveness in predicting a test dataset. The models were selected based on their relevance and applicability in various Machine Learning scenarios, representing a wide range of modeling techniques.

Thus, in Table 1, after calculating the accuracy as previously discussed, we can observed that the SVM achieved the highest precision, indicating its effectiveness in classifying the test data. On the other hand, the MLP showed inferior performance, suggesting possible challenges related to the model's complexity and hyperparameter tuning.

Table 1: Overall accuracy of the applied techniques.

Technique	Accuracy (%)
MLP	5.5
SVM	66
Naive Bayes	60
Random Forest	62

In addition to analyzing Machine Learning algorithms, this study applied Artificial Intelligence (AI) to analyze user stories. By training the model with a set of acceptance criteria from university projects, the AI demonstrated the ability to understand and process information, anticipating the quality of the stories, identifying patterns, and suggesting improvements.

The capability of AI to interpret nuances in user requirements is highlighted, demonstrating its potential as a valuable tool in software development. We used metrics such as accuracy to evaluate performance, providing strong indicators of AI's effectiveness in analyzing user stories.

Hence, the obtained results not only outlined the performance of the Machine Learning algorithms but also highlighted the significant contribution of AI in interpreting functional and non-functional system requirements, opening doors for future advancements in Software Engineering. This integrated approach provides a comprehensive view of the transformative potential of these technologies in understanding and meeting user needs in software development projects.

5 THREATS TO VALIDITY

When investigating the applicability of AI models in the automated analysis of user stories, it is crucial to consider various threats to the validity of the obtained results. These threats can impact the generalization of the findings and the interpretation of their applicability in different software development contexts, especially in small-scale projects such as startups, which face specific barriers to experimentation (Melegati et al., 2019). Inconsistencies, typos, and ambiguities in user stories can lead to poorly trained models that fail to adequately capture the essence of the requirements. Moreover, there is a risk of overfitting, especially with the use of the *Multilayer Perceptron* (MLP), where the model may overly adjust to the specificities of the training dataset, losing the ability to generalize to new data. The use of techniques such as *k-fold Cross-Validation* helps mitigate this risk, but it does not completely eliminate it.

The data used in this study were collected from academic projects and some real-world projects, primarily in the domains of educational and enterprise software. The dataset distribution between training and testing included 28 student stories and their acceptance criteria, which were used to validate the initial analysis. Later, the sample was expanded to 176 stories, with one or more acceptance criteria, covering a total of 204 scenarios. However, this raises concerns about overfitting and representativeness. Measures such as preprocessing techniques were implemented to ensure data quality. It is crucial to carefully split the dataset and consider a more comprehensive dataset to ensure the model's effectiveness and generalization. The quality of the dataset is vital to ensure

the accuracy of the results, and stories from students may not adequately reflect real-world market scenarios, necessitating more rigorous analysis and validation. Among the threats to the validity of the study are the potential lack of representativeness of academic data compared to real-world scenarios, the risk of overfitting due to exclusive use of student stories for training, and the possibility of biases introduced during data preprocessing.

The choice of AI models may not be fully representative of best practices in the ML field. Although the selected models are widely used, other approaches, possibly more recent or specialized, could potentially offer superior results. Relying mainly on accuracy to evaluate models may not fully reflect other important dimensions, such as precision, which can provide more detailed insights into the models' performance on different types of user stories.

The possibility of dependence between the training and test data, especially if improperly divided, can lead to an optimistic estimate of the models' performance. The conclusions obtained about the models' effectiveness may be influenced by personal biases or researchers' expectations, which is a common limitation in experimental studies.

6 CONCLUSION

This study demonstrated how the application of machine learning models and natural language processing can effectively assist in validating user stories in agile software development environments. By comparing four distinct algorithms (Multilayer Perceptron - MLP, Support Vector Machine - SVM, Naive Bayes, and Random Forest), the focus was to identify the most promising path for training a machine learning model. This not only automated the analysis of acceptance criteria but also significantly increased the accuracy and efficiency of this critical process.

The results confirm that Artificial Intelligence (AI) can effectively interpret and enhance software requirements, achieving an accuracy exceeding 60% with the SVM model, which stood out for its robustness. This research positively answers the initial question of how AI can improve the validation of acceptance criteria in user stories, demonstrating that it is possible to reduce human errors and increase the reliability of software deliveries.

Furthermore, the study highlights the importance of continuing to develop and integrate AI technologies into the software development process. The implementation of these technologies not only accelerates the development cycle but also promotes greater consistency in the quality of the produced software. The models tested in this study can be integrated as standard tools in agile development platforms, helping software development teams improve requirements communication and project execution.

This work also sheds light on possible threats to the validity of the results, such as the risk of overfitting and the representativeness of the data. It is crucial that future studies address these issues by expanding the diversity of datasets and exploring more advanced modeling approaches to ensure that the proposed solutions are generalizable and applicable in various development contexts.

For future work, it is advisable to expand the dataset, including real projects from different domains, and implement more robust cross-validation techniques to avoid overfitting. Additionally, conducting detailed comparisons with manual validation methods could help quantify efficiency gains. Exploring new algorithms and analyzing additional metrics are promising areas. Investigations into the integration with development tools and the use of explainability techniques to improve model understanding and mitigate potential biases are also recommended.

Finally, the insights provided by this research have the potential to guide future innovations in software engineering, particularly in terms of adopting BDD (Behavior-Driven Development) techniques (North, 2006) and integrating AI more deeply into the development cycle. The partial automation of the user story validation process can offer several benefits, such as reducing human errors, increasing efficiency and consistency in validation, and freeing up developers' time for more complex tasks (Nascimento et al., 2020). Moreover, by demonstrating that AI can interpret and improve software requirements, this study paves the way for creating automated tools that not only validate but also suggest improvements to user stories, potentially resulting in higher quality software products that are better aligned with *stakeholder* expectations. This advancement in understanding the role of AI in software engineering significantly contributes to the field, indicating pathways for future investigations that could further optimize processes and outcomes in software development.

ACKNOWLEDGMENT

This study was partially supported by the Ministry of Science, Technology, and Innovations from Brazil, with resources from Law No. 8.248, dated October 23, 1991, within the scope of PPI-SOFTEX, coordinated by Softex.

REFERENCES

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Chanin, R., Sales, A., Santos, A. R., Pompermaier, L. B., and Prikladnicki, R. (2018). A collaborative approach to teaching software startups: findings from a study using challenge based learning. In Sharp, H., de Souza, C. R. B., Graziotin, D., Levy, M., and Socha, D., editors, *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 9–12. ACM.

Conrado, C. (2012). *Gerenciamento de requisitos de software: um guia prático para o desenvolvimento ágil*. Elsevier.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Erdogmus, H., Morisio, M., and Torchiano, M. (2005). On the effectiveness of the test-first approach to programming. *IEEE Trans. on Soft. Eng.*, 31(3):226–237.

Johnson, R., Lee, S., and Kim, E. (2023). Automating user story validation using natural language processing: A case study. *ACM Trans. on Soft. Eng. and Methodology*, 32(1):1–19.

Li, X., Zhang, W., Chen, M., and Wang, J. (2021). Leveraging machine learning for user story validation: A systematic literature review. *Journal of Systems and Software*, 181:111127.

Melegati, J., Chanin, R., Sales, A., Prikladnicki, R., and Wang, X. (2020). MVP and experimentation in software startups: a qualitative survey. In *46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020, Portoroz, Slovenia, August 26-28, 2020*, pages 322–325. IEEE.

Melegati, J., Chanin, R., Wang, X., Sales, A., and Prikladnicki, R. (2019). Enablers and inhibitors of experimentation in early-stage software startups. In Franch, X., Männistö, T., and Martínez-Fernández, S., editors, *Product-Focused Software Process Improvement - 20th International Conference, PROFES 2019, Barcelona, Spain, November 27-29, 2019, Proceedings*, volume 11915 of *Lecture Notes in Computer Science*, pages 554–569. Springer.

Mohanty, A. (2019). Multi layer Perceptron (MLP) Models on Real World Banking Data. Retrieved June, 2021 from https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f.

Nascimento, N., Santos, A. R., Sales, A., and Chanin, R. (2020). Behavior-driven development: A case study on its impacts on agile development teams. In *ICSE '20: 42nd International Conference on Software Engineering, Workshops, Seoul, Republic of Korea, 27 June - 19 July, 2020*, pages 109–116. ACM.

Nguyen, D.-M., Huynh, Q.-T., Ha, N.-H., and Nguyen, T.-H. (2020). Automated test input generation via model inference based on user story and acceptance criteria for mobile application development. *International Journal of Software Engineering and Knowledge Engineering*, 30(03):399–425.

North, D. (2006). Introducing bdd: The future of test automation. *Better Software*, 8(6):34–43.

Oliveira, G. and Marczak, S. (2018). On the understanding of BDD scenarios' quality: Preliminary practitioners' opinions. In *Proceedings of the Requirements Engineering: Foundation for Software Quality, Utrecht, The Netherlands, Springer, Cham, pp. 290–296*.

Oliveira, G., Marczak, S., and Moralles, C. (2019). How to evaluate bdd scenarios' quality? In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering, ACM, Salvador, Brazil, 2019, pp. 481–490*.

Pressman, R. S. (2015). *Software Engineering: A Practitioner's Approach*. McGraw Hill, 8 edition.

Silva, T. S. C., Marczak, S., and Rocha, F. G. (2020). On the understanding of how to measure the benefits of behavior-driven development adoption: Preliminary literature results from a grey literature study. In Viana, D. and Schots, M., editors, *19th Brazilian Symp. on Software Quality, (SBQS), São Luís, Brazil, December, 2020*, page 39. ACM.

Smith, J. and Jones, A. (2022). The impact of machine learning on agile software development: A review of recent advances. *IEEE Trans. on Soft. Eng.*, 48(6):890–905.

Wang, P. and Manning, C. D. (2012). Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers*, pages 90–94.

Wiegers, K. E. (2003). *Understanding Software Requirements*. Microsoft Press.