Enhancing Software Requirements Analysis Using Machine Learning Techniques

Grant Guernsey

November 15, 2024

Abstract

In software engineering, requirements analysis is crucial for the successful development of software systems. This study explores the potential of machine learning techniques to improve the accuracy and efficiency of identifying and categorizing software requirements from textual data. Using a dataset of 977 entries classified into functional and non-functional requirements, natural language processing methods were employed to automate the labeling process. Different machine learning algorithms were compared, such as linear and deep learning models to determine their effectiveness in classifying requirements. The impact of various feature extraction techniques on classification performance is also investigated. Our results demonstrate that machine learning can enhance the requirements engineering process, reducing the time and effort needed for accurate requirements gathering and improving software quality.

1 Introduction

The process of requirements analysis is a fundamental aspect of software engineering that significantly influences the success of software development projects. Traditional methods of identifying and categorizing software requirements are often manual, time-consuming, and susceptible to human error. This research aims to explore the potential of machine learning techniques to automate

the identification and classification of software requirements from textual data, thereby enhancing efficiency and accuracy.

2 Dataset Preview

A dataset comprising 977 software requirements entries was utilized, each classified as either a functional or non-functional requirement. A preview of the dataset is provided in Table 1.

Type	Requirement
PE	The system shall refresh the display every 60 seconds.
LF	The application shall match the color of the system theme.
US	If projected, the data must be readable. On small screens, it must adapt accordingly.
A	The product shall be available during normal business hours.
US	If projected, the data must be understandable without additional explanations.

Table 1: Dataset Preview

3 Missing Values Check

A check for missing values in the dataset revealed that there are no null entries in either the `Type` or `Requirement` columns, ensuring the integrity of the dataset for analysis.

4 Dataset Information

The dataset consists of 977 entries with two primary features:

- **Type**: Categorical data indicating the requirement type (functional or non-functional).
- **Requirement**: Textual descriptions of the software requirements.

Data types and memory usage have been validated to optimize processing efficiency.

5 Unique Requirement Types

The dataset contains the following unique requirement types:

- PE (Performance)
- LF (Look and Feel)
- US (Usability)
- A (Availability)
- SE (Security)
- F (Functional)
- L (Legal)
- O (Operational)
- PO (Portability)
- SC (Scalability)
- FT (Fault Tolerance)
- MN (Maintainability)
- FR (Functional Requirement)
- NFR (Non-Functional Requirement)

These categories provide a diverse set of classes for the classification task.

6 Requirement Length Statistics

Statistical analysis of the requirements' textual length was conducted to understand the data distribution. The average length of a requirement is approximately 127 characters, with lengths ranging from 13 to 1118 characters. This variation necessitates appropriate preprocessing steps to normalize the data for model training.

7 Visualization of Requirement Types

Figure 1 illustrates the distribution of requirement types within the dataset. This visualization highlights the prevalence of each type, which is essential for understanding class imbalances that may affect model performance.

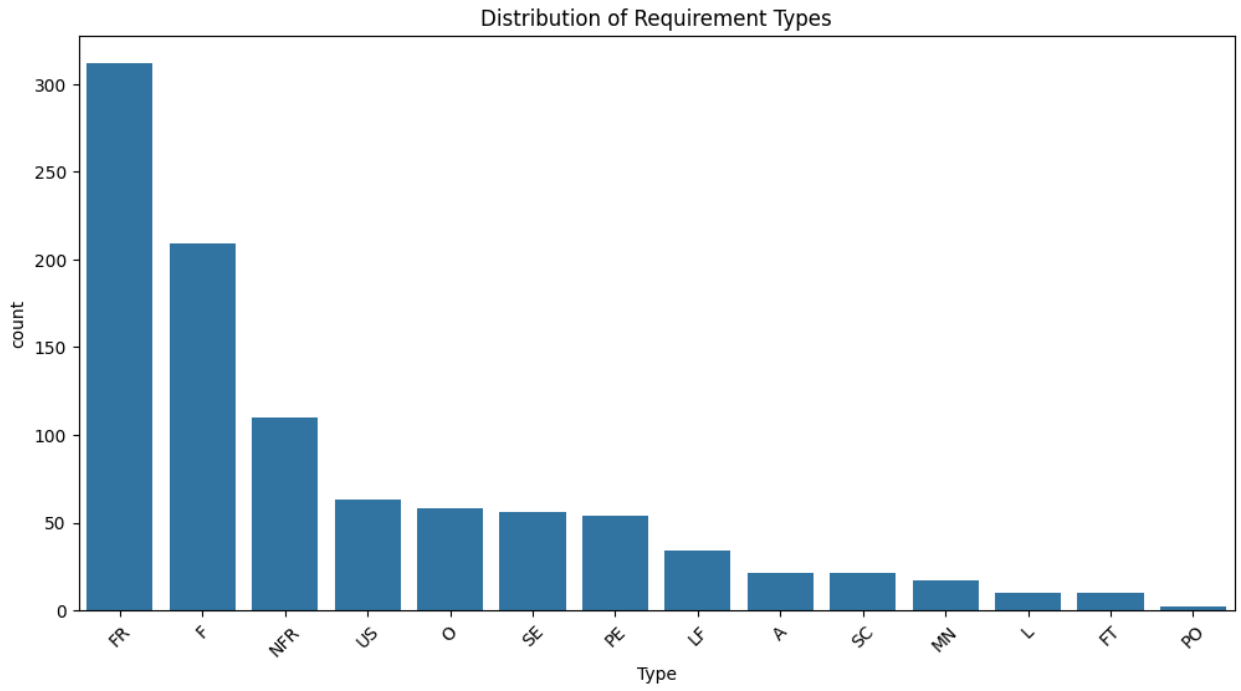

Figure 1: Distribution of Requirement Types

8 Methodology

To classify software requirements as either functional or non-functional, a combination of natural language processing (NLP) and machine learning techniques were applied, supported by a deep learning model for text classification and a ChromaDB-based semantic search for requirement retrieval.

8.1 Data Preprocessing

Each requirement text was preprocessed to improve model performance:

- **Tokenization**: Requirements were split into individual tokens using the NLTK library's word tokenizer.
- **Stop-word Removal**: Common stop-words, which add little semantic value, were removed.
- **Text Normalization**: All tokens were converted to lowercase, and non-alphabetic characters were filtered out.

8.2 Feature Extraction

Two feature extraction methods were tested:

- **TF-IDF (Term Frequency-Inverse Document Frequency)**: Used as an initial method for traditional machine learning classifiers, transforming text data into a sparse matrix representation.
- **Word Embeddings**: Using Sentence Transformers, each requirement was embedded into a 384-dimensional vector space, capturing semantic similarity. These embeddings improved downstream model performance.

8.3 Classification Models

We implemented multiple models to classify requirements:

- **Traditional Machine Learning Models**: A linear model was used as a basline classifier.
- **Deep Learning Model**: A neural network classifier with one hidden layer, using 128 hidden units, ReLU activation, and a dropout layer for regularization, was implemented using PyTorch. The model outputs were fed into a softmax layer to predict the requirement type.
- **Semantic Search Model**: This technique uses a semantic similarity search within a Vector Database to classify based on semantically similar,

The neural network was trained for 300 epochs with the Adam optimizer, a learning rate of 0.001, and cross-entropy loss. The training process was monitored to ensure convergence and avoid overfitting.

8.4 Embedding Storage and Semantic Search Using ChromaDB

To enable requirement retrieval based on semantic similarity:

- **ChromaDB Integration**: The Sentence Transformer embeddings were stored in ChromaDB. Each entry was indexed with its embedding, requirement text, and type.
- **Requirement Retrieval**: A querying function was implemented that compares a given requirement's embedding to those stored in ChromaDB, returning the closest match based on cosine similarity.

8.5 Model Evaluation

Each model's performance was evaluated on an 80/20 train-test split using:

- **Accuracy**: Calculated as the proportion of correctly classified requirements.

- **Confusion Matrix**: Used to visualize classification performance across all requirement types.

8.6 Results Visualization

The final confusion matrix and accuracy scores were plotted using Seaborn, allowing us to assess each model's effectiveness in handling class imbalances and identify potential misclassification patterns.

9 Results and Discussion

The classification models were evaluated using accuracy metrics and confusion matrices to determine their effectiveness in categorizing software requirements as either functional or non-functional. Three primary models were assessed: the linear classifier, the neural network classifier, and the embedding-based retrieval system using ChromaDB.

9.1 Linear Classifier Performance

The linear classifier achieved the highest accuracy at 74.49%, outperforming other models. Figure 2 shows the confusion matrix for the linear classifier, highlighting its strong performance in classifying both functional and non-functional requirements, with relatively few misclassifications. This result suggests that, despite its simplicity, the linear classifier effectively captures key patterns in the dataset.

9.2 Neural Network Classifier Performance

The neural network achieved an accuracy of 72.96%, demonstrating its ability to handle the complexities of textual data. Figure 3 shows the confusion matrix for the neural network classifier, where it correctly classified a majority of the functional and non-functional requirements. How-

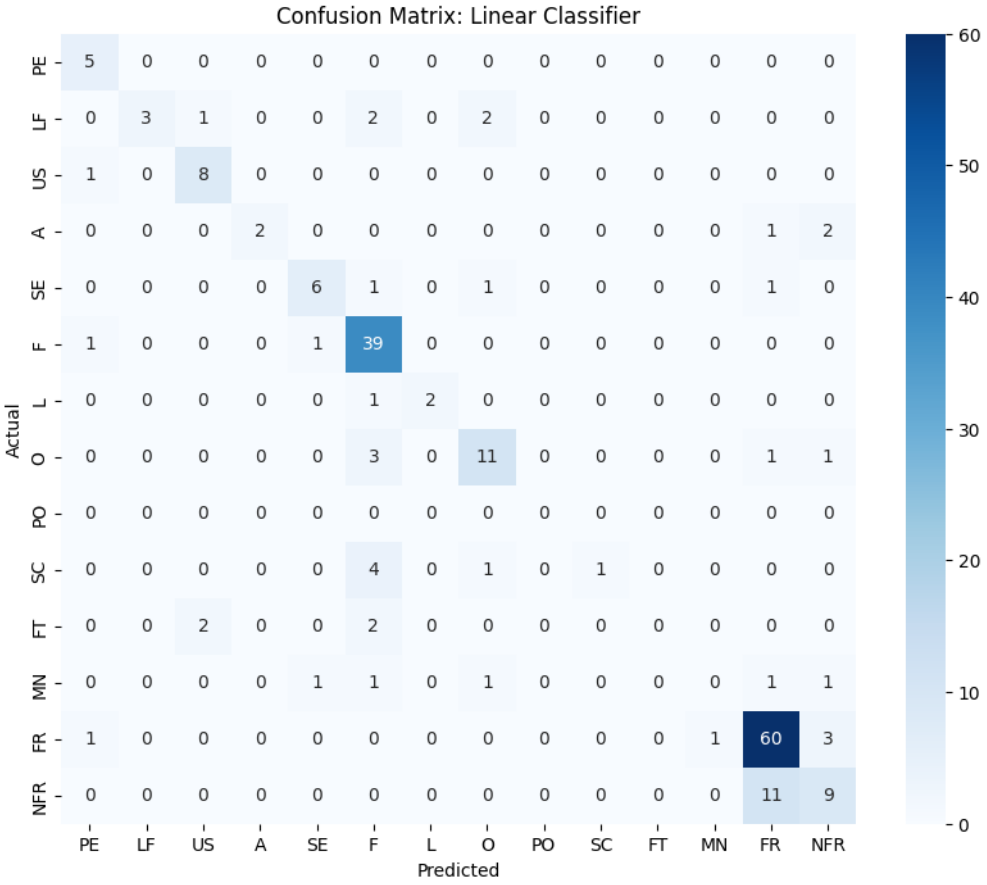

Figure 2: Confusion Matrix for Linear Classifier

ever, misclassifications were observed across certain requirement types, indicating potential areas for improvement, such as refining feature extraction techniques or adjusting model architecture.

9.3 Embedding-based Retrieval System Performance

The embedding-based retrieval system, leveraging ChromaDB for semantic similarity matching, obtained an accuracy of 66.33%. As shown in Figure 4, the system was able to capture broad semantic similarities between requirements but struggled with finer-grained distinctions between closely related types. This result suggests that while embeddings provide valuable context for requirement retrieval, they may benefit from additional tuning or the inclusion of more advanced semantic modeling techniques to enhance classification precision.

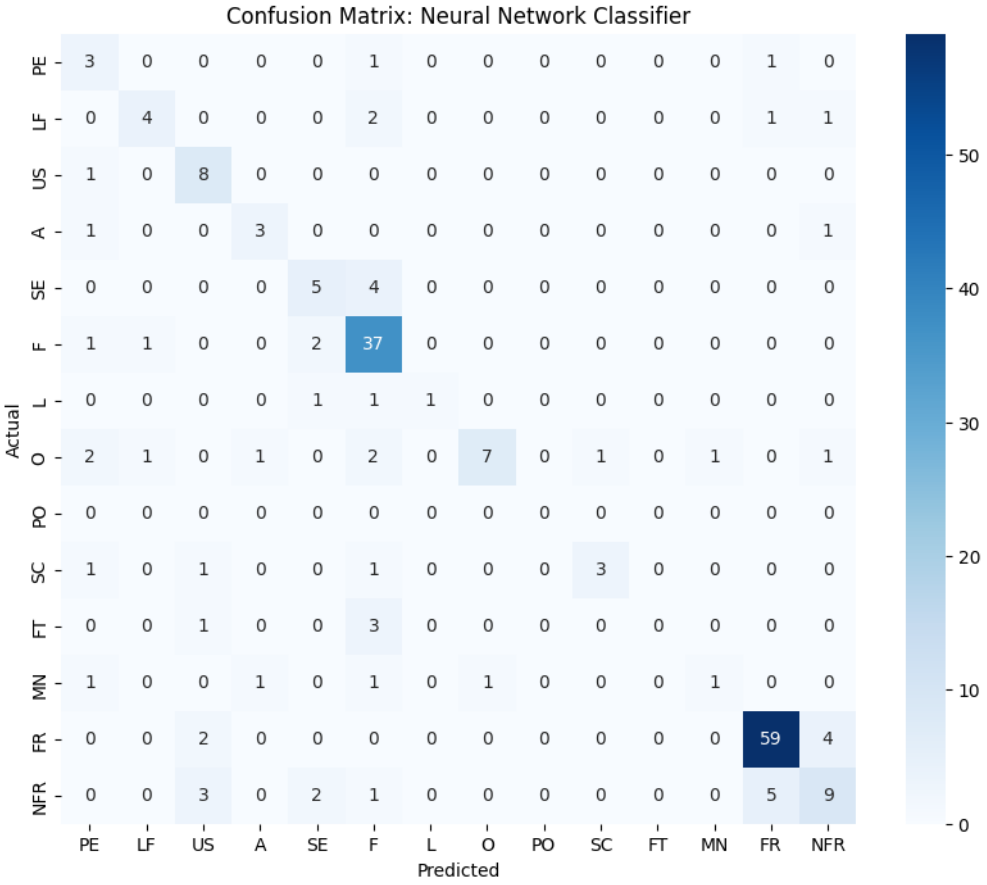

Figure 3: Confusion Matrix for Neural Network Classifier

9.4 Comparison and Analysis

The linear classifier outperformed the neural network and embedding-based approaches, underscoring the effectiveness of traditional machine learning methods in this classification task. While the neural network performed well, achieving an accuracy of 72.96%, the simpler linear model captured essential patterns more reliably. The embedding-based approach, while lower in accuracy, provided valuable insights into the semantic relationships between requirements.

Both models' confusion matrices provide insight into common misclassifications, which could be addressed in future work by integrating domain-specific embeddings or fine-tuning the neural network with additional domain data. Furthermore, exploring ensemble techniques or combining neural networks with embedding-based retrieval may offer a hybrid approach to enhance overall accuracy.

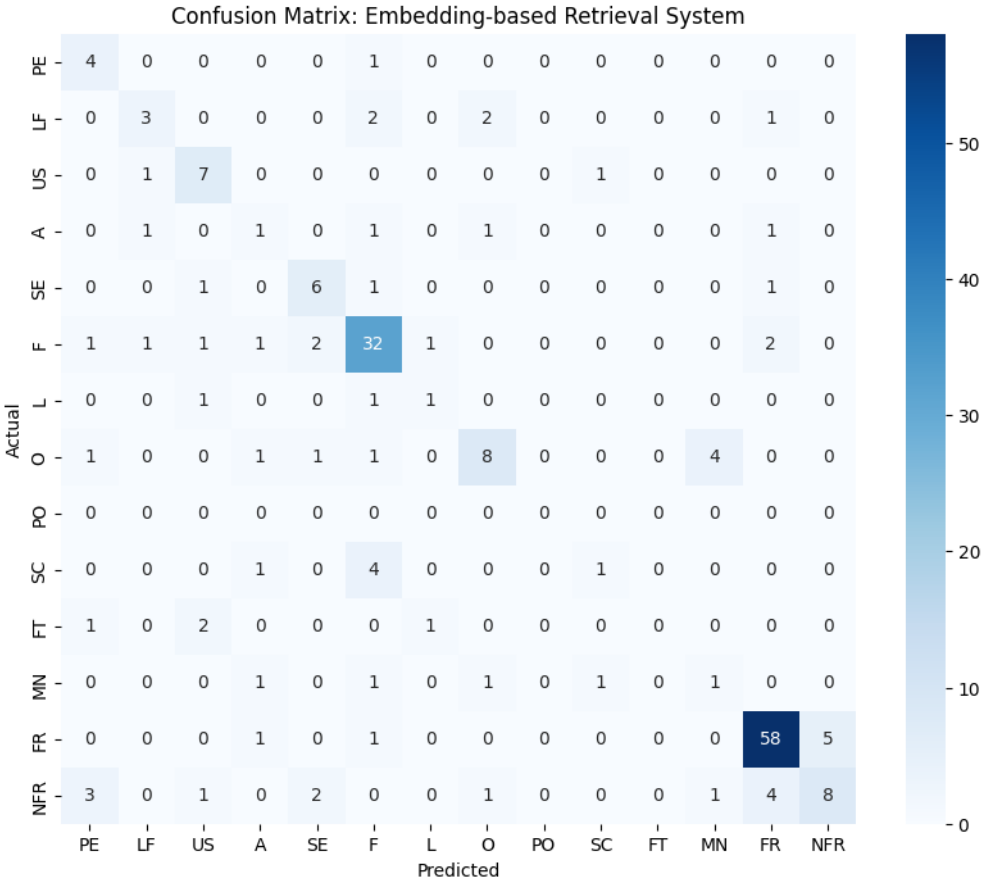

Figure 4: Confusion Matrix for Embedding-based Retrieval System

9.5 Use of Vector Database for Requirement Classification

Despite its lower accuracy compared to traditional models, the vector database approach provides a unique and valuable perspective in requirement classification by leveraging semantic similarity. This method, implemented with ChromaDB, allows requirements to be embedded into a high-dimensional space where the similarity function can retrieve requirements with related meanings or contexts, even if their wording or phrasing differs significantly.

A key advantage of this approach is its scalability and adaptability with larger datasets. With more data, the vector database's similarity function could become more precise, capturing subtle semantic distinctions that differentiate requirements. Many requirements are differentiated by nuanced semantics and specific wording, which makes this approach well-suited for applications that need to recognize contextually similar requirements, even when they are not lexically identical.

In scenarios with a larger dataset, the vector database would be better positioned to identify clusters of semantically similar requirements, enhancing retrieval accuracy. This could address the current model's challenge of distinguishing closely related requirement types. Furthermore, with fine-tuned or domain-specific embeddings, the vector database could be optimized to handle the specific semantic patterns prevalent in software requirements, offering a robust solution for requirement retrieval and classification in complex datasets.

Thus, the vector database remains a viable and promising option, particularly in cases where understanding subtle variations in requirement language is essential. As requirements datasets expand, this approach could provide a more flexible and context-aware solution that complements traditional classification methods.

10 Conclusion

This study illustrates the potential of machine learning techniques to automate the classification of software requirements, offering valuable support in the requirements engineering process. By leveraging both traditional classifiers and vector-based retrieval methods, we showed that these approaches can reduce manual effort, improve classification accuracy, and address semantic nuances in requirements text. Each model demonstrated unique strengths: the linear classifier provided reliable accuracy for general classification, while the vector database offered insights into semantic similarities, which can be further leveraged with larger datasets. Together, these methods represent a promising path toward enhancing software quality and development efficiency by streamlining requirements analysis and management.

References

[1] Alzubaidi, L., Zhang, J., Humaidi, A.J., et al. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data, 8*, 53. https://doi.org/10.1186/s40537-021-00444-8

[2] Dias Canedo, E., & Cordeiro Mendes, B. (2020). Software requirements classification using machine learning algorithms. *Entropy, 22(9)*, 1057. https://doi.org/10.3390/e22091057

[3] Görkem, G. (2021). A software engineering perspective on engineering machine learning systems: State of the art and challenges. *Journal of Systems and Software, 180*, C. https://doi.org/10.1016/j.jss.2021.111031

[4] Luitel, D., Hassani, S., & Sabetzadeh, M. (2023). Using language models for enhancing the completeness of natural-language requirements. In A. Ferrari & B. Penzenstadler (Eds.), *Requirements Engineering: Foundation for Software Quality (Vol. 13975)*. Springer, Cham. https://doi.org/10.1007/978-3-031-29786-1_7

[5] Liping Zhao, W., Alhoshan, W., Ferrari, A., Letsholo, K.J., Ajagbe, M.A., Chioasca, E.V., & Batista-Navarro, R.T. (2021). Natural language processing for requirements engineering: A systematic mapping study. *ACM Computing Surveys, 54(3)*, Article 55. https://doi.org/10.1145/3444689

[6] Sarker, I.H. (2021). Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science, 2*, 160. https://doi.org/10.1007/s42979-021-00592-x

[7] Schulze, S., & Li, Y. (2023). Feature and variability extraction from natural language requirements. In R.E. Lopez-Herrejon, J. Martinez, W.K. Guez Assunção, T. Ziadi, M. Acher, & S. Vergilio (Eds.), *Handbook of Re-Engineering Software Intensive Systems into Software Product Lines*. Springer, Cham. https://doi.org/10.1007/978-3-031-11686-5_2

[8] Shan, C. (2012). Research of support vector machine in text classification. In T. Zhang (Eds.), *Future Computer, Communication, Control and Automation (Vol. 119)*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-25538-0_79

[9] Sheth, V., Tripathi, U., & Sharma, A. (2022). A comparative analysis of machine learning algorithms for classification purpose. *Procedia Computer Science, 215*, 422-431. https://doi.org/10.1016/j.procs.2022.12.044

[10] Thakur, R.B.S. (2024). Associated requirements classification model and standardization for enhancing quality of software. *International Journal of Intelligent Systems and Applications in Engineering, 12(23s)*, 515–532. https://ijisae.org/index.php/IJISAE/article/view/6894