ArtificialGate Ltd. Research Preprint

Improving Requirements Classification with SMOTE-Tomek Preprocessing

Barak Or
ArtificialGate Ltd., Israel
barak@artificialgate.ai

Abstract

This study emphasizes the domain of requirements engineering by applying the SMOTE-Tomek preprocessing technique, combined with stratified K-fold cross-validation, to address class imbalance in the PROMISE dataset. This dataset comprises 969 categorized requirements, classified into functional and non-functional types. The proposed approach enhances the representation of minority classes while maintaining the integrity of validation folds, leading to a notable improvement in classification accuracy. Logistic Regression achieved 76.16% ± 2.58%, substantially exceeding its baseline of 59.85% ± 2.52%. These results highlight the applicability and efficiency of machine learning models as scalable and interpretable solutions.

Keywords. Requirements Engineering; Class Imbalance; SMOTE-Tomek; Imbalanced Datasets; Requirements Classification; Natural Language Processing; Synthetic Oversampling; Text-Based Classification; Supervised Learning

1 Introduction

Requirements engineering (RE) is a cornerstone of the software development lifecycle, translating stakeholder needs into actionable system specifications [1–3]. Accurate and efficient requirement classification is essential to ensure project clarity, prioritization, and goal alignment. Requirements are typically categorized into functional, non-functional, and subtypes [4–8]. A survey revealed that over 60% of failed projects neglected non-functional requirements, underscoring the critical importance of effective classification [9]. Despite its significance, requirement classification remains a largely manual process, prone to inconsistencies, inefficiencies, and scalability challenges [10].

Over the years, researchers have explored various approaches to automate requirement classification. Early rule-based systems were among the first attempts, offering interpretable but inflexible solutions that were labor-intensive to maintain [11]. The advent of machine learning (ML) brought more adaptive models, demonstrating moderate success, particularly in structured datasets [4, 7, 12]. However, these methods often face challenges with class imbalance, a prevalent issue in real-world datasets, including the widely used PROMISE dataset for requirements engineering [13]. This imbalance skews model predictions toward majority classes, undermining performance on minority classes.

Over the past decade, deep learning (DL) has achieved transformative breakthroughs across multiple domains, including natural language processing (NLP), computer vision, and time series analysis. These advancements are primarily driven by the unprecedented representational power of deep neural networks, which leverage millions, and in some cases billions, of trainable parameters to extract and model intricate, high-dimensional patterns from large-scale datasets [14].

Equally significant are the advances in time series analysis, where DL models have demonstrated remarkable capabilities in tasks such as motion sensing classification and physical quantity estimation [15–18]. These innovations highlight DL's ability to capture temporal dependencies and complex properties in sequential data.

In computer vision, architectures such as ResNet [19] and Vision Transformers (ViT) [20] have revolutionized image classification, object detection, and semantic segmentation, achieving state-of-the-art accuracy while enabling applications that were previously unattainable.

In NLP, DL models such as BERT [21] and GPT-3 [22] have set new standards in tasks such as language

This research was conducted under ArtificialGate Ltd. The author serves as Academic Director at the Google & Reichman Tech School, and as an External Lecturer at Technion - Israel Institute of Technology and Reichman University.

translation, text summarization, and question answering, surpassing traditional methods and enabling more nuanced understanding and generation of human language [23–27]. These models excel in NLP tasks due to their ability to capture contextual relationships within text using self-attention mechanisms. They are particularly effective at modeling semantic intricacies in textual data, enabling strong performance in complex classification tasks. However, these methods require not only substantial computational resources for fine-tuning and inference but also access to massive amounts of high-quality data to achieve their full potential. This reliance on extensive datasets and powerful hardware often makes them less practical for small-scale or resource-constrained projects, where data availability and computational capacity may be limited.

To address these challenges, this study integrates the Synthetic Minority Oversampling Technique (SMOTE) [28] with Tomek Links (SMOTE-Tomek) [29], providing a robust preprocessing solution for imbalanced datasets. SMOTE oversamples minority classes by generating synthetic samples through interpolation, effectively enhancing the diversity of underrepresented classes while mitigating overfitting, a common issue with random oversampling methods. Complementing this, Tomek Links identifies and removes noisy or borderline samples, improving class separability by reducing overlapping data points between classes. This dual-action preprocessing strategy results in cleaner and more balanced training datasets, enabling ML models to better learn decision boundaries for minority classes.

The proposed approach is particularly suited to text-based datasets such as the PROMISE dataset, where class imbalance and noise frequently undermine classification performance. By addressing these issues, SMOTE-Tomek creates a foundation for training classical ML models, which can achieve useful performance without the computational demands of deep learning approaches. This study builds on the strengths of SMOTE-Tomek by integrating it into a stratified K-fold cross-validation framework, preserving the integrity of validation folds and ensuring rigorous evaluation.

In RE classification, SMOTE-Tomek addresses both class imbalance and noise, making it particularly suitable for the PROMISE dataset, where minority classes are significantly underrepresented. While previous studies have demonstrated the efficacy of SMOTE-Tomek for addressing class imbalance in the PROMISE dataset [30], this research extends its application by integrating it into a stratified K-fold cross-validation framework. This ensures that the validation folds remain untouched by resampling, preserving the original class distribution and enabling evaluation of the model's generalization capabilities. Furthermore, this study explores several classical ML models, evaluates their performance on the PROMISE dataset, and highlights the potential of lightweight solutions for efficient requirement classification.

This study makes the following key contributions:

- A systematic application of SMOTE-Tomek to address class imbalance in the PROMISE dataset, within a 10-fold cross-validation framework that ensures the validation set remains unaffected by resampling.
- A comparative evaluation of classical ML models, demonstrating their potential for scalable and efficient requirement classification.
- An analysis of Logistic Regression (LR) coefficients to interpret the relationship between features and requirement types, offering insights into the most frequent and influential words for each type.

The experimental results demonstrate that integrating SMOTE-Tomek with classical ML frameworks improves classification performance. LR, for instance, achieves an accuracy improvement from 59.85% ± 2.52% to 76.16% ± 2.58% under cross-validation, an absolute gain of 16.31 percentage points.

The remainder of this paper is structured as follows: Section 2 outlines the methodology, including an overview of the dataset, preprocessing steps, the class imbalance challenge, the SMOTE-Tomek approach, and the classical ML models utilized. Section 3 presents the results and their implications, and Section 4 concludes with directions for future research.

2 Learning Method

This section provides an overview of the methodology employed in this study, detailing the dataset, class imbalance challenges, and the preprocessing techniques applied to address them. It outlines the evaluated ML models, the implementation of the SMOTE-Tomek approach for data balancing, and the use of K-fold cross-validation to support robust performance assessment.

2.1 Dataset

The study utilizes the expanded PROMISE dataset, a diverse collection of 969 software requirements extracted from Software Requirements Specification (SRS) documents using the Google search engine [13].

The dataset contains 444 functional requirements (45.8%) and 525 non-functional requirements (54.2%), distributed across 12 distinct categories. These categories represent a spectrum of requirements in software engineering, such as Security (SE), Usability (US), Portability (PO), and Performance (PE). Despite its comprehensiveness, the dataset exhibits notable class imbalance. For example, the Portability category is severely underrepresented, comprising only 12 requirements (1.24%), while the most prevalent non-functional class, Security, includes 125 requirements (12.9%). This disparity affects the performance and generalizability of ML models. Figure 1 illustrates the distribution of the dataset across all requirement types.

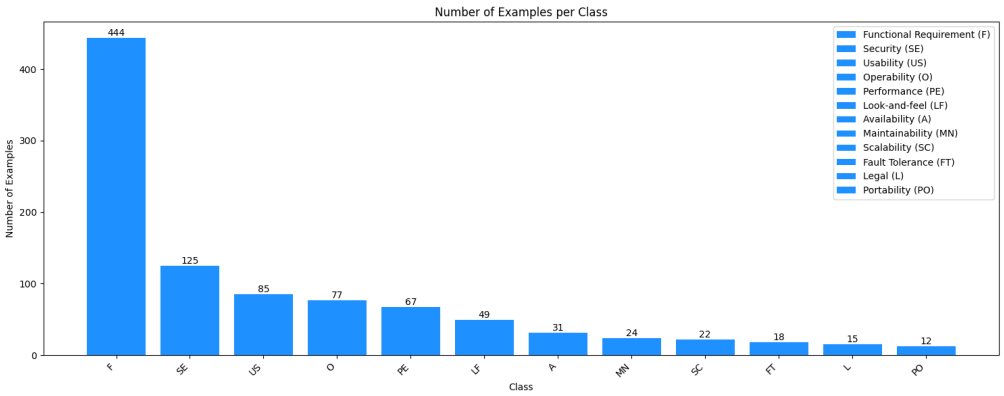

Figure 1. Requirement type distribution in the PROMISE dataset.

The following is a concise description of the requirement types used in this study:

Functional Requirement (F)	Defines the specific functionalities and behaviors the system must perform to meet stakeholder needs.
Availability Requirement (A)	Ensures that the system remains operational and accessible under defined conditions and constraints.
Legal Requirement (L)	Addresses compliance with legal, regulatory, and contractual obligations relevant to the system's deployment and use.
Look-and-Feel Requirement (LF)	Specifies the aesthetic and user interface characteristics to align with user expectations and branding.
Maintainability Requirement (MN)	Focuses on the system's ability to be easily modified, updated, or repaired during its lifecycle.
Operability Requirement (O)	Ensures that the system can be operated efficiently and effectively within its intended environment.
Performance Requirement (PE)	Defines the system's capability to deliver expected throughput, response time, and efficiency under specified conditions.
Scalability Requirement (SC)	Addresses the system's ability to adapt to increased workloads or expanded functionalities without degradation.
Security Requirement (SE)	Protects the system and its data against unauthorized access, breaches, and vulnerabilities.
Usability Requirement (US)	Ensures the system is intuitive and user-friendly, enabling stakeholders to interact with it effectively.
Fault Tolerance Requirement (FT)	Ensures the system can continue functioning correctly even in the event of hardware or software failures.
Portability Requirement (PO)	Specifies the system's ability to be deployed or operated across various platforms, environments, or configurations.

Table 1. Example requirements and their corresponding types from the PROMISE dataset

Requirement Example	Requirement Type
The product shall have audit capabilities. The product shall store messages for a minimum of one year for audit and transaction tracking purposes.	Security (SE)
The recycled parts search results provided to the estimator shall be retrieved by the system.	Functional (F)
Users should be able to access their streaming movies in under two clicks after logging into the website.	Usability (US)
The product shall be available 24 hours per day, seven days per week.	Availability (A)
The system shall help the user avoid making mistakes while scheduling classes and clinicals for the nursing students.	Usability (US)
The system shall display a detailed invoice for the current order once it is confirmed.	Functional (F)
The system will use a secured database.	Security (SE)
The leads washing functionality will store any potential lead duplicates returned by the enterprise system.	Functional (F)
The system shall interface with the faculty central server.	Operability (O)
The product shall be internet browser independent. The product must run using Internet Explorer and Netscape Navigator.	Maintainability (MN)

2.2 Problem Definition: Multiclass Classification

While some studies in requirements engineering focus on a binary classification task, distinguishing only between functional and non-functional requirements, the present study addresses a more complex multiclass classification problem.

The objective is to automatically categorize each requirement into one of the 12 distinct classes available in the PROMISE dataset, comprising Functional plus 11 non-functional subtypes. This formulation is more challenging due to the extreme class imbalance, where minority classes such as Portability or Fault Tolerance represent less than 2% of the dataset. By defining the task as multiclass, we aim to demonstrate the robustness of the SMOTE-Tomek approach in learning fine-grained semantic boundaries across a diverse spectrum of requirement types, rather than identifying only broad categories.

2.3 Pre-Processing

Transforming unstructured text into a numerical representation suitable for ML algorithms is a fundamental preprocessing step in many NLP tasks. The PROMISE dataset comprises individual sentences rather than full documents. Term Frequency-Inverse Document Frequency (TF-IDF) remains a highly effective vectorization technique [31, 32]. TF-IDF quantifies the importance of a term within a sentence while considering its prevalence across the entire dataset of sentences. By emphasizing terms that are frequent within a specific sentence but rare across the dataset, TF-IDF captures features that are contextually meaningful and relevant to the classification task. Despite the brevity of the textual units, TF-IDF preserves the semantic significance of terms.

2.4 Class Imbalance Challenge

Class imbalance poses a critical challenge in training ML models, as they tend to prioritize majority classes, resulting in poor generalization and suboptimal performance on minority classes [33]. Such biases can affect the usability and reliability of classification models in practical applications, where correctly identifying minority classes is often essential. Balancing the training set helps the model give greater attention to underrepresented classes and reduces bias toward majority classes.

2.5 SMOTE-Tomek

Among various methods to address data imbalance, SMOTE-Tomek is a hybrid technique that combines oversampling and data cleaning. SMOTE addresses the imbalance by generating synthetic samples for minority classes, producing a more equitable class distribution in the training data. However, SMOTE alone can introduce noise by oversampling borderline or overlapping samples. Tomek Links complement SMOTE by identifying and removing borderline or ambiguous samples that may hinder model training. Together, SMOTE-Tomek enhances data quality while balancing the class distribution during training. Unlike random oversampling, which duplicates existing minority class samples, SMOTE creates new synthetic samples based on linear interpolation between existing samples.

In NLP applications, where textual data can be complex and imbalanced, the synthetic examples

generated by SMOTE are not direct textual entities. They are feature vectors representing characteristics of the minority class, as illustrated in Table 2. These vectors serve as abstract representations, enabling the model to learn structural properties of underrepresented classes without requiring additional annotated textual data.

The SMOTE-Tomek process is described in the following steps:

1. **Generate Synthetic Samples for Minority Classes:** For each sample x_i belonging to the minority class, synthetic samples are generated using linear interpolation:

$$x_{\text{synthetic}} = x_i + \lambda(x_{\text{nn}} - x_i), \tag{1}$$

   where x_{nn} is a randomly selected nearest neighbor of x_i from the same minority class, and $\lambda \in [0, 1]$ is a random scalar.

2. **Identify Tomek Links:** A Tomek Link is a pair of samples x_i and x_j that are mutual nearest neighbors, meaning that x_i is the closest sample to x_j and vice versa. The two samples belong to different classes, with one from the minority class and the other from the majority class. The nearest-neighbor condition can be written as

$$d(x_i, x_j) = \min_{x_k}\{d(x_i, x_k)\}, \tag{2}$$

   with the reciprocal condition defined analogously for x_j.

3. **Remove Majority Class Samples:** For each identified Tomek Link, remove the sample belonging to the majority class to enhance class separation and reduce noise.

Table 2. Examples of resampled requirement representations using SMOTE-Tomek

Requirement Representation	Requirement Type
based, established, lead, operate, physical, process, product, run, server, service, shall, structure, washing, web	Operability (O)
changes, edit, information, just, modify, normal, personal, read, securely, server, transmission, transmitted, users	Security (SE)
entry, shown	Functional (F)
initiator, mediator, messages, request, response, shall, user	Functional (F)
accommodate, compensatory, data, failures, fault, items, large, product, recovery, robust, routing, shall, technique, tolerance, transaction, using	Fault Tolerance (FT)
explanatory, intuitive, self, shall	Usability (US)
control, details, employees, site, view	Functional (F)
browsers, ce, compatible, environments, expected, interface, major, operating, order, palm, product, run, standards, systems, usable, variety, web, wide, windows	Portability (PO)
800x600, 95, appropriately, correct, corrected, display, feel, higher, implementation, incorrect, intranet, look, notification, pages, prior, provide, remaining, resolutions, shall, uniform, web, week	Look-and-Feel (LF)
able, alongside, days, environment, function, installed, java, operating, product, runtime, server, shall, software	Operability (O)

2.6 Stratified K-fold Cross-Validation Method

Stratified K-fold cross-validation provides advantages over a single train-test split, particularly when classes are imbalanced. By dividing the dataset into K folds and using each fold for validation exactly once, the method makes broad use of the available data. Stratified sampling maintains the proportional representation of classes in each fold, reducing the risk that minority classes are omitted from training or validation subsets.

To maintain the integrity of the cross-validation process, SMOTE-Tomek resampling is applied solely to the training folds during each iteration of the 10-fold stratified cross-validation. Specifically, in each iteration, the resampling technique is applied to the nine training folds, while the validation fold remains untouched. This preserves the original distribution of the validation data, avoids data leakage from synthetic samples into validation, and enables a more realistic evaluation of model performance. As illustrated in Figure 2, SMOTE-Tomek is applied only to the training folds.

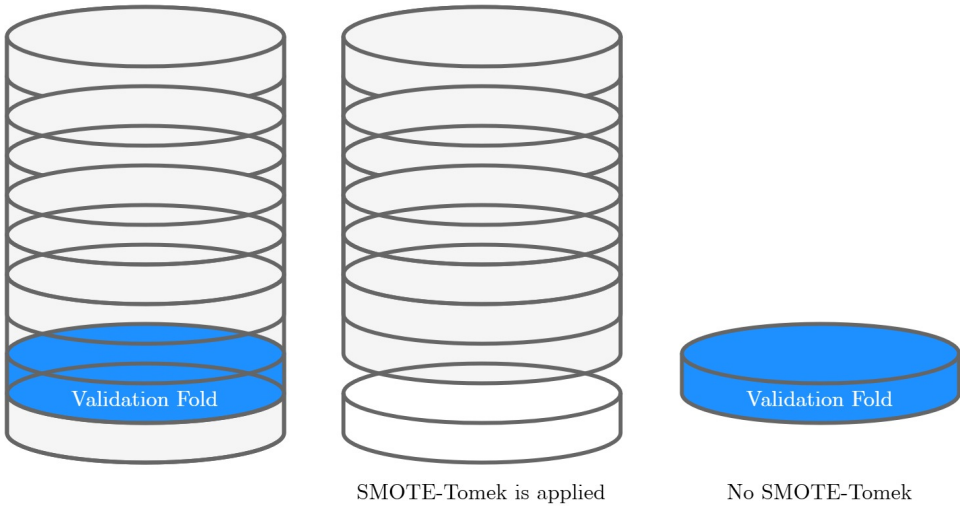

Figure 2. Stratified cross-validation with SMOTE-Tomek applied only to the training folds; the validation fold is not resampled.

2.7 Training Algorithm

The proposed method, outlined in Algorithm 1, integrates stratified K-fold cross-validation to preserve class proportions across folds. SMOTE-Tomek is applied solely to the training folds to prevent data leakage and preserve the integrity of validation metrics. The algorithm also includes final training on the complete balanced dataset after cross-validation.

Table 3. Notations and definitions

Notation	Definition
$\mathcal{D}$	Complete dataset used for training and evaluation
K	Number of folds in stratified K-fold cross-validation
$\mathcal{F}_i$	Fold i used as the validation set in the i-th iteration
$\mathcal{D}_{\text{train}}$	Training set formed by $K-1$ folds
$\mathcal{D}_{\text{val}}$	Validation set corresponding to fold i
$\mathcal{M}_i$	Model trained in the i-th cross-validation iteration

Algorithm 1 Training with SMOTE-Tomek and Stratified K-Fold Cross-Validation

Require: $\mathcal{D}$, K, $\mathcal{M}$

1: Split $\mathcal{D}$ into K stratified folds $\{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_K\}$, preserving class proportions.
2: For fold $i = 1$ to K:
  2:1. $\mathcal{D}_{\text{train}} \leftarrow \bigcup_{j \neq i} \mathcal{F}_j$.
  2:2. $\mathcal{D}_{\text{val}} \leftarrow \mathcal{F}_i$.
  2:3. $\mathcal{D}_{\text{train,balanced}} \leftarrow$ apply SMOTE-Tomek only to $\mathcal{D}_{\text{train}}$.
  2:4. Train $\mathcal{M}_i$ on $\mathcal{D}_{\text{train,balanced}}$.
  2:5. Evaluate $\mathcal{M}_i$ on $\mathcal{D}_{\text{val}}$ and compute the metrics.
3: Compute average metrics across all K folds.
4: Train $\mathcal{M}_{\text{final}}$ on $\mathcal{D}$ after applying SMOTE-Tomek to the complete training data.
5: Return $\mathcal{M}_{\text{final}}$ and the average cross-validation metrics.

2.8 Classical ML Models

The following ML models were evaluated in this study to capture diverse classification perspectives:

- **Decision Tree (DT):** A tree-based algorithm that splits the data into subsets based on feature values, creating a hierarchy of decisions to classify data.
- **Random Forest:** An ensemble method combining multiple decision trees, where each tree votes and the majority decision is taken.
- **Support Vector Machine (SVM) with Linear Kernel:** A linear classifier that finds the hyperplane maximizing the margin between classes.
- **SVM with RBF Kernel:** A nonlinear classifier that maps data into a higher-dimensional space using the radial basis function kernel.

- **Naive Bayes:** A probabilistic model based on Bayes' theorem, assuming feature independence, and commonly used for text classification.
- **Logistic Regression:** A statistical model that predicts class probabilities using the logistic function.
- **K-Nearest Neighbors (KNN):** A non-parametric algorithm that classifies samples based on the majority class of their nearest neighbors in feature space.
- **Gradient Boosting:** An iterative ensemble method that builds sequential trees, correcting errors of previous trees to minimize classification error.
- **AdaBoost:** An adaptive boosting algorithm that combines weak classifiers and gives higher weights to misclassified samples.
- **CatBoost:** A gradient boosting algorithm optimized for categorical features.

3 Results and Discussion

This section evaluates the performance of the ML models for requirement type classification, with a focus on the impact of SMOTE-Tomek preprocessing. It outlines the error metrics, compares baseline performance with the performance obtained after SMOTE-Tomek integration, highlights the interpretability of Logistic Regression, and discusses the observed improvements and limitations.

3.1 Error Metrics

Let TP, FP, TN, and FN denote true positives, false positives, true negatives, and false negatives, respectively. Precision quantifies the proportion of correctly predicted positive cases among all predicted positive cases:

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{3}$$

Recall evaluates the proportion of actual positive cases correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{4}$$

The F1-score is the harmonic mean of precision and recall:

$$\text{F1} = \frac{2TP}{2TP + FP + FN}. \tag{5}$$

Accuracy assesses overall correctness:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{6}$$

Matthews Correlation Coefficient (MCC) provides a summary of classification quality under imbalance. For a binary confusion matrix, it is defined as

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \tag{7}$$

The reported experiment uses the corresponding multiclass extension. MCC ranges from -1 to $+1$, where $+1$ represents perfect classification, 0 indicates random or uninformed classification, and -1 represents complete disagreement.

3.2 Baseline Performance Without SMOTE-Tomek

The baseline experimental results are summarized in Table 4 and visualized in Figure 3. Among all evaluated classifiers, the linear SVM achieved the strongest overall performance, attaining a mean accuracy of 71.10%, precision of 68.25%, recall of 71.10%, F1-score of 66.47%, and an MCC of 0.5977. This indicates a favorable balance between predictive accuracy and robustness under class imbalance.

KNN classifiers with $k = 3$ and $k = 5$ also demonstrated competitive performance, with mean accuracies of 67.69% and 68.42%, respectively, and MCC values close to that of the linear SVM. These results suggest that local neighborhood-based methods can capture discriminative structure in the PROMISE dataset despite class imbalance. Gradient Boosting achieved a mean accuracy of 66.98% and an MCC of 0.5344.

In contrast, Random Forest and AdaBoost showed weaker results. AdaBoost, in particular, produced a mean precision of 28.21% and an MCC of 0.1963. Logistic Regression and Decision Trees achieved

moderate performance but were outperformed by SVM and KNN variants without imbalance-aware preprocessing.

Table 4. Performance for each model without SMOTE-Tomek

Model	Mean Accuracy (%)	Mean Precision (%)	Mean Recall (%)	Mean F1-Score (%)	Mean MCC
Decision Tree	59.55 ± 2.72	54.71 ± 4.08	59.55 ± 2.72	53.48 ± 3.15	0.4116 ± 0.0473
Random Forest	48.71 ± 1.48	38.26 ± 8.61	48.71 ± 1.48	34.11 ± 2.53	0.1796 ± 0.0672
SVM (Linear)	71.10 ± 3.09	68.25 ± 4.18	71.10 ± 3.09	66.47 ± 3.73	0.5977 ± 0.0456
SVM (RBF)	58.31 ± 2.05	58.05 ± 5.33	58.31 ± 2.05	49.74 ± 2.88	0.4010 ± 0.0382
Naive Bayes	51.49 ± 2.04	45.57 ± 3.96	51.49 ± 2.04	38.80 ± 2.91	0.2566 ± 0.0510
Logistic Regression	59.85 ± 2.52	56.85 ± 5.00	59.85 ± 2.52	51.85 ± 3.65	0.4181 ± 0.0470
KNN ($k = 3$)	67.69 ± 3.84	66.11 ± 4.98	67.69 ± 3.84	64.34 ± 4.55	0.5478 ± 0.0586
KNN ($k = 5$)	68.42 ± 2.69	66.37 ± 3.29	68.42 ± 2.69	64.95 ± 3.12	0.5564 ± 0.0427
KNN ($k = 7$)	67.80 ± 2.93	65.07 ± 3.98	67.80 ± 2.93	63.84 ± 3.57	0.5468 ± 0.0452
Gradient Boosting	66.98 ± 2.92	66.37 ± 5.05	66.98 ± 2.92	63.13 ± 3.05	0.5344 ± 0.0446
AdaBoost	49.02 ± 1.14	28.21 ± 0.94	49.02 ± 1.14	33.71 ± 1.33	0.1963 ± 0.0472
CatBoost	58.82 ± 3.11	55.21 ± 5.54	58.82 ± 3.11	50.73 ± 4.32	0.3978 ± 0.0636

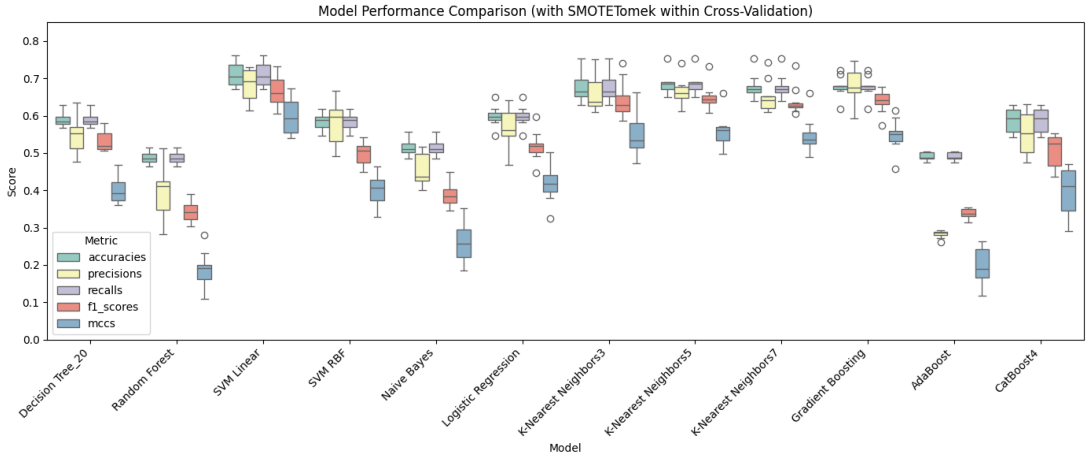

Figure 3. Cross-validation performance of machine learning models without SMOTE-Tomek preprocessing.

3.3 Enhanced Performance Using SMOTE-Tomek

The impact of SMOTE-Tomek preprocessing on classification performance is presented in Table 5 and Figure 4. Following the application of SMOTE-Tomek, Logistic Regression exhibited the most pronounced improvement across the reported evaluation metrics, achieving a mean accuracy of 76.16% ± 2.58%, precision of 75.31% ± 4.30%, recall of 76.16% ± 2.58%, F1-score of 74.34% ± 2.93%, and an MCC of 0.6736 ± 0.0370.

Relative to the baseline Logistic Regression results in Table 4, this corresponds to an absolute accuracy gain of 16.31 percentage points. MCC increased from 0.4181 ± 0.0470 to 0.6736 ± 0.0370, an absolute gain of 0.2555. Performance was further improved by approximately one percentage point through hyperparameter optimization using $C = 10$, an L2 regularization penalty, and the SAGA solver [34].

Several additional classifiers benefited from SMOTE-Tomek preprocessing. Naive Bayes and Gradient Boosting achieved mean accuracies exceeding 70%, with MCC values of 0.6286 and 0.5938, respectively. In contrast, AdaBoost remained highly sensitive to the resampled data distribution, achieving a mean accuracy of 16.10% and an MCC of 0.1422.

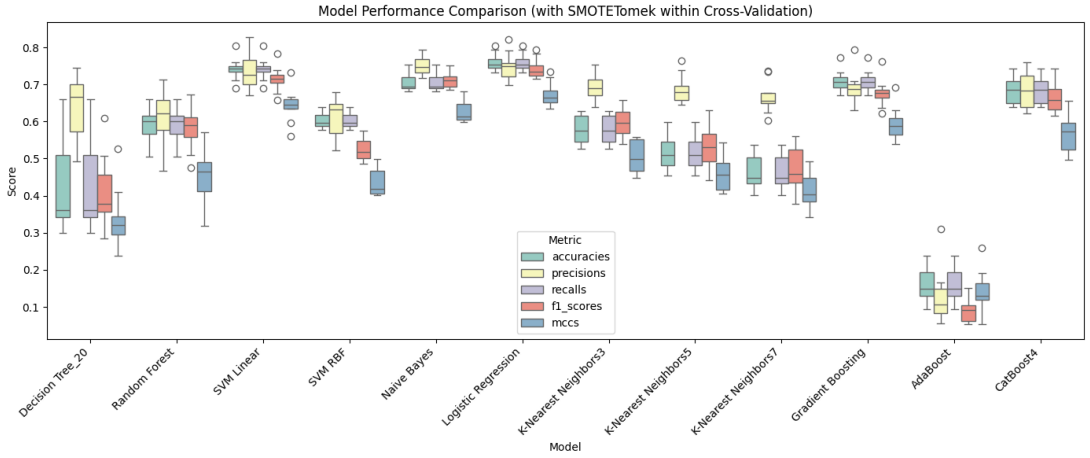

Figure 4. Cross-validation performance of machine learning models with SMOTE-Tomek preprocessing.

Table 5. Performance for each model with SMOTE-Tomek

Model	Mean Accuracy (%)	Mean Precision (%)	Mean Recall (%)	Mean F1-Score (%)	Mean MCC
Decision Tree	41.79 ± 12.44	63.89 ± 7.90	41.79 ± 12.44	40.43 ± 9.53	0.3363 ± 0.0780
Random Forest	59.03 ± 4.50	60.82 ± 6.96	59.03 ± 4.50	58.05 ± 5.39	0.4562 ± 0.0686
SVM (Linear)	74.20 ± 2.83	73.46 ± 4.58	74.20 ± 2.83	71.44 ± 3.26	0.6428 ± 0.0424
SVM (RBF)	60.27 ± 2.12	61.34 ± 5.40	60.27 ± 2.12	52.68 ± 3.09	0.4358 ± 0.0346
Naive Bayes	70.48 ± 2.28	75.04 ± 2.42	70.48 ± 2.28	71.17 ± 2.06	0.6286 ± 0.0280
Logistic Regression	76.16 ± 2.58	75.31 ± 4.30	76.16 ± 2.58	74.34 ± 2.93	0.6736 ± 0.0370
KNN ($k = 3$)	57.79 ± 3.65	69.21 ± 3.19	57.79 ± 3.65	59.59 ± 3.78	0.5063 ± 0.0413
KNN ($k = 5$)	51.60 ± 4.31	68.56 ± 3.76	51.60 ± 4.31	53.00 ± 5.20	0.4596 ± 0.0451
KNN ($k = 7$)	46.34 ± 4.45	66.64 ± 4.05	46.34 ± 4.45	47.11 ± 5.69	0.4148 ± 0.0463
Gradient Boosting	70.90 ± 2.77	69.08 ± 4.15	70.90 ± 2.77	67.76 ± 3.56	0.5938 ± 0.0419
AdaBoost	16.10 ± 4.42	12.67 ± 6.97	16.10 ± 4.42	9.06 ± 3.07	0.1422 ± 0.0526
CatBoost	68.21 ± 3.32	68.34 ± 4.55	68.21 ± 3.32	66.37 ± 3.83	0.5633 ± 0.0492

3.4 Feature Analysis and Model Interpretability

One of the primary advantages of linear models such as Logistic Regression is their interpretability. LR allows direct examination of feature importance through learned coefficients. To investigate the effect of SMOTE-Tomek preprocessing on the model's learned representation, we analyzed the top word coefficients for each requirement type in the multiclass classification setting.

Figure 5 visualizes the absolute values of the top three word coefficients for each requirement category, comparing the LR model trained on the imbalanced baseline dataset with the model trained using SMOTE-Tomek.

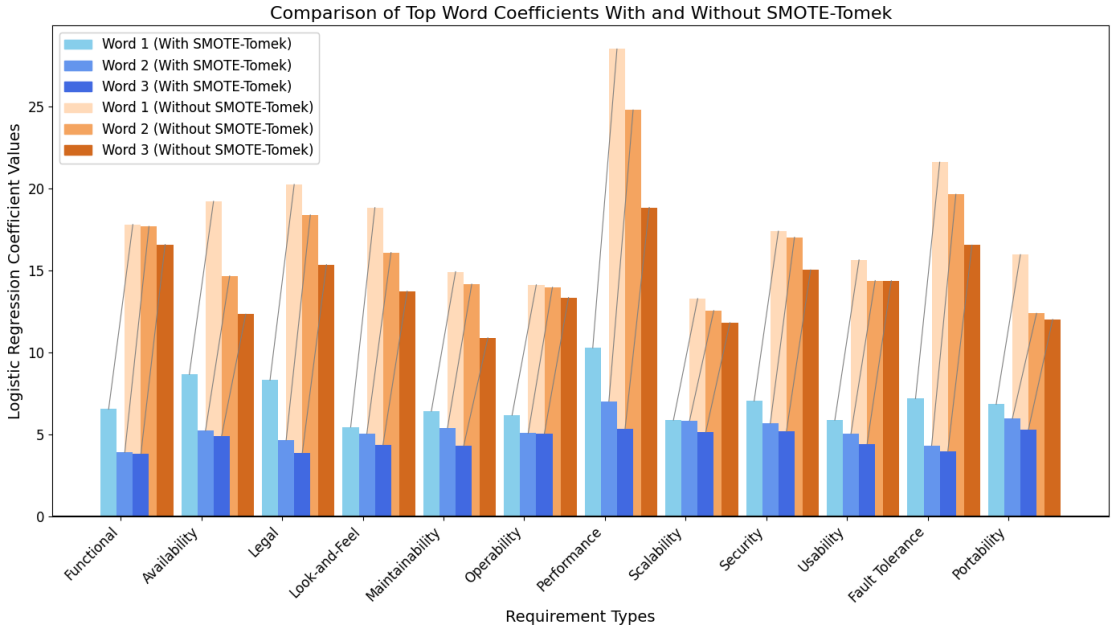

Figure 5. Comparison of top word coefficients with and without SMOTE-Tomek preprocessing across requirement types.

The analysis reveals a clear difference in coefficient magnitudes. Without SMOTE-Tomek, represented by the orange shades, the model exhibits large and variable coefficient values, particularly for minority classes such as Performance, Legal, and Fault Tolerance. For example, the top word for the Performance class reaches a coefficient value near 30. This pattern suggests that the baseline model may rely heavily on a small number of terms in classes with limited training examples.

With SMOTE-Tomek, represented by the blue shades, the top coefficient magnitudes are lower, mostly between 4 and 10, and are more similar across the three displayed features within each class. The result is consistent with reduced reliance on isolated terms and a more distributed lexical representation. Because the figure compares coefficient magnitudes rather than formal stability estimates across repeated runs, these observations should be interpreted descriptively.

3.5 Discussion

The highest observed accuracy was 76.16% for Logistic Regression after applying SMOTE-Tomek. This result highlights both the potential and the limitations of classical ML models for requirements classification. Although performance improved relative to the Logistic Regression baseline, data scarcity and class imbalance remain important constraints in the PROMISE dataset. These challenges motivate the use of larger and more representative datasets to capture the diversity and nuances of requirement types.

Stratified K-fold cross-validation supported an evaluation framework that preserved the original class distribution in every validation fold. Crucially, the validation folds remained untouched by resampling. This protocol reduces the risk of data leakage from synthetic samples and highlights the importance of rigorous evaluation when testing class-imbalance mitigation strategies.

SMOTE-Tomek was associated with a substantial improvement in Logistic Regression metrics. Accuracy increased from 59.85% ± 2.52% to 76.16% ± 2.58%. MCC increased from 0.4181 ± 0.0470 to 0.6736 ± 0.0370, an absolute gain of 0.2555. LR benefited from the changed training distribution and remained interpretable and scalable for the classification task.

Despite the overall improvement, several challenges emerged:

- Ensemble methods such as AdaBoost and Random Forest underperformed even with SMOTE-Tomek preprocessing. AdaBoost achieved a mean accuracy of 16.10% and an MCC of 0.1422.
- Gradient Boosting and Linear SVM demonstrated competitive results, although their performance gains were more modest than that of LR.
- The PROMISE dataset remains small and constrained in its coverage of requirement types. The restricted variety of classes and reliance on textual features alone limit generalization.

4 Conclusions

We demonstrated the potential of integrating SMOTE-Tomek preprocessing with classical ML models to address class imbalance in requirements classification. Logistic Regression exhibited the most substantial improvement, achieving 76.16% ± 2.58% accuracy compared with its baseline of 59.85% ± 2.52%. Its MCC increased from 0.4181 ± 0.0470 to 0.6736 ± 0.0370, an absolute gain of 0.2555. The interpretability of LR enabled an analysis of feature importance, while the coefficient comparison indicated lower and more evenly distributed top coefficient magnitudes after SMOTE-Tomek preprocessing.

These findings emphasize the practicality and scalability of classical ML models for imbalanced text datasets, particularly in resource-constrained environments. The methodology may also be applicable to domains such as legal or healthcare document analysis, where similar imbalance challenges exist. Future work will explore larger datasets, advanced feature representations, and hybrid approaches combining lightweight models with deep learning techniques to improve performance and generalizability.

5 Conflict of Interest Statement

The author declares no conflict of interest related to this work.

References

[1] Gerald Kotonya and Ian Sommerville. *Requirements Engineering: Processes and Techniques*. Wiley Publishing, 1998.

[2] Daniel Lanfear, Mina Maleki, and Shadi Banitaan. Enhancing software requirements classification with machine learning and feature selection techniques. In *International Conference on Software Engineering and Data Engineering*, pages 14–30. Springer, 2024.

[3] Li Cai, Hongpeng Yin, and Jingdong Lin. A unified framework with incremental learning capacity for industrial fault detection and classification. *IEEE Transactions on Automation Science and Engineering*, 2024.

[4] Edna Dias Canedo and Bruno Cordeiro Mendes. Software requirements classification using machine learning algorithms. *Entropy*, 22(9):1057, 2020.

[5] Gaith Y. Quba, Hadeel Al Qaisi, Ahmad Althunibat, and Shadi AlZu'bi. Software requirements classification using machine learning algorithms. In *2021 International Conference on Information Technology (ICIT)*, pages 685–690. IEEE, 2021.

[6] Jonas Winkler and Andreas Vogelsang. Automatic classification of requirements based on convolutional neural networks. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 39–45. IEEE, 2016.

[7] Manal Binkhonain and Liping Zhao. A machine learning approach for hierarchical classification of software requirements. *Machine Learning with Applications*, 12:100457, 2023.

[8] Wei Wei, Chuan Jiang, and Yuzhe Huang. A data-driven human-machine collaborative product design system toward intelligent manufacturing. *IEEE Transactions on Automation Science and Engineering*, 2023.

[9] Vikas Bajpai and Ravi Prakash Gorthi. On non-functional requirements: A survey. In *2012 IEEE Students' Conference on Electrical, Electronics and Computer Science*, pages 1–4. IEEE, 2012.

[10] Axel van Lamsweerde and Emmanuel Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, 2000.

[11] George Spanoudakis, Andrea Zisman, Elena Pérez-Mi nana, and Paul Krause. Rule-based generation of requirements traceability relations. *Journal of Systems and Software*, 72(2):105–127, 2004.

[12] Michael I. Jordan and Tom M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[13] Márcia Lima, Victor Valle, Estev ao Costa, Fylype Lira, and Bruno Gadelha. Software engineering repositories: Expanding the promise database. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, pages 427–436, 2019.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[15] Maxim Freydin and Barak Or. Learning car speed using inertial sensors for dead reckoning navigation. *IEEE Sensors Letters*, 6(9):1–4, 2022.

[16] Maxim Freydin, Nimrod Segol, Niv Sfaradi, Areej Eweida, and Barak Or. Deep learning for inertial sensor alignment. *IEEE Sensors Journal*, 2024.

[17] Barak Or. Carspeednet: A deep neural network-based car speed estimation from smartphone accelerometer. arXiv preprint arXiv:2401.07468, 2024.

[18] Barak Or. Transformer-based dog behavior classification with motion sensors. *IEEE Sensors Journal*, 2024.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[20] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929, 2020.

[21] Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[22] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[24] Xianchang Luo, Yinxing Xue, Zhenchang Xing, and Jiamou Sun. Prcbert: Prompt learning for requirement classification using bert-based pretrained language models. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–13, 2022.

[25] Ahmad F. Subahi. Bert-based approach for greening software requirements engineering through non-functional requirements. *IEEE Access*, 2023.

[26] Hrvoje Belani, Marin Vukovic, and Željka Car. Requirements engineering challenges in building ai-based complex systems. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pages 252–255. IEEE, 2019.

[27] Raul Navarro-Almanza, Reyes Juarez-Ramirez, and Guillermo Licea. Towards supporting software engineering using deep learning: A case of software requirements classification. In *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pages 116–120. IEEE, 2017.

[28] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[29] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, 6(1):20–29, 2004.

[30] Jalil Abbas, Cheng Zhang, and Bin Luo. Enscl-catboost: A strategic framework for software requirements classification. *IEEE Access*, 2024.

[31] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the First Instructional Conference on Machine Learning*, volume 242, pages 29–48. Citeseer, 2003.

[32] Stephen Robertson. Understanding inverse document frequency: On theoretical arguments for idf. *Journal of Documentation*, 60(5):503–520, 2004.

[33] Justin M. Johnson and Taghi M. Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):1–54, 2019.

[34] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, volume 27, 2014.