אוניברסיטת אריאל

2024

Machine Learning Course Final Project

Software Requirements Classification

<span>Author:</span> Nikita Breslavsky 332363498

Abstract

In this project [1], I address the challenge of classifying system requirements. My approach was to employ and assess a range of machine learning algorithms to pinpoint the most efficacious model for the task. The project spanned initial data exploration and preprocessing, followed by judicious model selection based on performance metrics. This document succinctly outlines the methodology, the model implementation, and provides insights into the experiences gained, paving the way for a detailed analysis of the results and conclusions to follow.

Introduction

Classifying system requirements is essential in software development, ensuring projects meet their goals efficiently. This project focuses on using machine learning to automate the classification of system requirements, employing a dataset of system specifications.

Functional requirements specify what a system should do, detailing its operations and features, like allowing users to log in or process payments. Non-functional requirements define how the system performs these tasks, focusing on quality and constraints, such as speed, security, and usability. Essentially, functional requirements are about the system's actions, while non-functional requirements are about the system's attributes and performance.

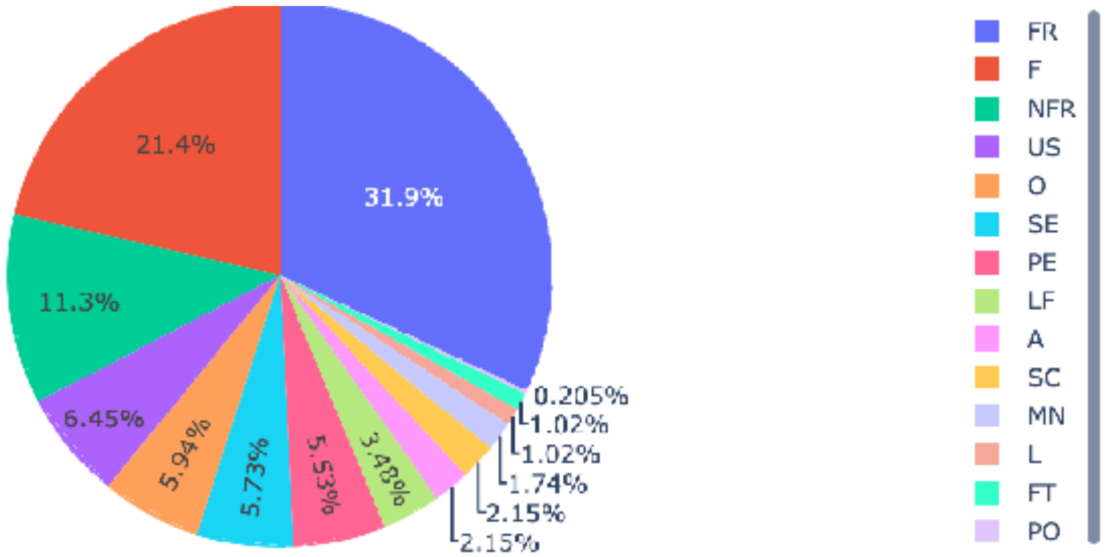
The project underscores the potential of machine learning in enhancing the precision and efficiency of software engineering processes.

Dataset Description

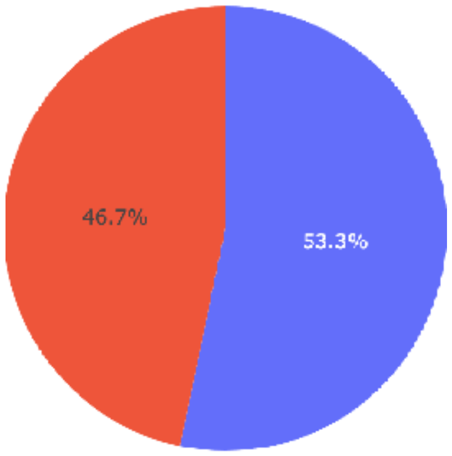
The search for a suitable dataset for classifying system requirements led to the discovery of a unique dataset on Kaggle [2], one of the few available in this specialized field. With only 1,000 entries, it is a relatively small dataset but is classified meticulously into 14 distinct classes. The scarcity of datasets in the field of system requirement classification poses a significant challenge, making this particular dataset invaluable despite its limitations.

Data Exploration and Preprocessing

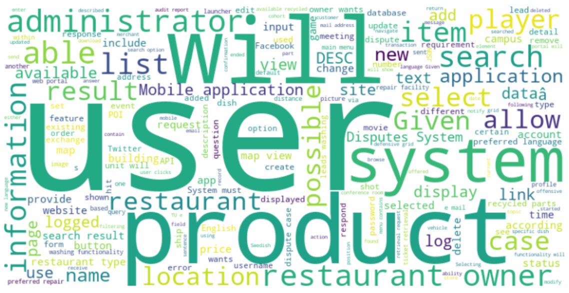
This project's dataset comprises two primary columns: Type and Requirement. Initially containing 14 distinct types, the dataset distinguishes between functional requirements and non-functional. The text data within the Requirement column is clean, focusing on specific system requirements without irrelevant noise such as personal names or external links. It has mix of small and large letters, numerals (e.g., time, percentages, amounts), and technical terms.

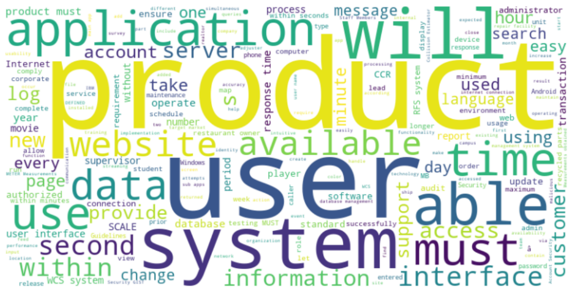


Given the dataset's small size and class imbalance, I condensed the 14 types into two broad categories: functional and non-functional. This reorganization aimed to address the imbalance, simplifying the classification task and focusing the analysis. Such consolidation is expected to improve the accuracy of models in distinguishing between the two types of system requirements.



Word cloud analysis revealed commonality across categories with terms like "user," "product," and "system," indicating their central role in system requirements discussions. However, distinct differences emerged, such as "restaurant" and "search" being prevalent in functional requirements, versus "website" and "application" in non-functional requirements, reflecting their respective focuses.





Efforts to conduct a bigram analysis were challenged by the dataset's limited size and variability, resulting in no significant findings of recurring phrases. This may happen due dataset’s limited size.

The preprocessing phase was critical for preparing the dataset for analysis. A versatile preprocessing function, req_to_words, was developed to clean and process the textual data based on specific criteria: removing stopwords and numbers, lemmatization or stemming of words, and number normalization. This function was designed to be adjustable, enabling the fine-tuning of preprocessing parameters to explore their impact on model accuracy.

Example: <u>The</u> system shall refresh <u>the</u> display every <u>60</u> seconds<u>.</u>

<u>Without stop words</u>: system shall refresh display every 60 seconds

<u>With normalized numbers</u>: the system shall refresh the display every num seconds

<u>Without numbers</u>: the system shall refresh the display every seconds

<u>With lemmatization</u>: the system shall refresh the display every second

<u>With stemming</u>: the system shall refresh the display everi second

<u>Most clean</u>: system shall refresh display everi second

For text vectorization, the TF-IDF (Term Frequency-Inverse Document Frequency) technique was selected for its simplicity and effectiveness in highlighting the importance of words within documents relative to the dataset. Although alternative methods like Count Vectorizer and more complex solutions such as word embeddings were considered, they were ultimately deemed too complex for my needs. These advanced methods require substantial external data and did not demonstrate a significant improvement in results over TF-IDF. Additionally, the use of n-grams within TF-IDF was explored but dismissed due to the previously noted absence of meaningful n-grams in our dataset, a result of its limited size.

The preprocessing steps undertaken were aimed at refining the dataset to a state that would support the most accurate and insightful analysis possible with the chosen machine learning models. By focusing on TF-IDF for vectorization and offering flexibility in text processing through req_to_words, the preprocessing phase laid a solid foundation for the subsequent modeling and analysis stages.

Methodology

The project's analytical approach was guided by the machine learning models covered in the course, supplemented with additional methods to enrich the comparative analysis. The selected models are as follows:

- Logistic Regression (LogR): Chosen for its efficiency in binary classification problems, Logistic Regression served as a baseline for evaluating model performance due to its straightforward interpretation and implementation.
- K-Nearest Neighbors (KNN): This model was selected for its simplicity and effectiveness in classification tasks, leveraging the 'closeness' of data points to predict class membership.
- Decision Trees: Known for their interpretability, Decision Trees were utilized both independently and in conjunction with AdaBoost. The latter, an ensemble method, was employed to enhance the model's accuracy by combining multiple weak learners into a strong learner.
- Support Vector Machine (SVC): SVC was included for its robustness in high-dimensional spaces, making it a suitable choice for text classification tasks. Its capability to model complex relationships between features and classes warranted its use.

- Naive Bayes: As a probabilistic classifier known for its simplicity and effectiveness in text classification, Naive Bayes was added to the methodology for its suitability in dealing with the high-dimensional data typical of text-based datasets.
- Voting Classifier (Hard and Soft): To leverage the strengths of individual models, both hard and soft voting classifiers were applied. These ensemble techniques combine predictions from the above models to improve overall accuracy. Hard voting makes a prediction based on the majority vote from models, while soft voting considers the probability scores of predictions, potentially offering a more nuanced aggregation of model insights.

This diverse array of models was strategically chosen to explore various aspects of machine learning classification techniques, from simple to complex and individual to ensemble methods. Each model's inclusion aimed to provide a comprehensive understanding of different approaches to the classification task, reflecting the breadth of machine learning methodologies discussed in the course.

Model Implementation

The model implementation phase adopted a uniform approach across various machine learning models to ensure consistency in evaluation and comparability of results. This strategy included steps for hyperparameter optimization, feature importance analysis, result interpretation, and real-world applicability testing.

Hyperparameter Optimization

For each model, hyperparameter tuning was executed using GridSearchCV from sklearn, aiming to optimize model performance for accuracy. The search encompassed a wide range of hyperparameters relevant to each specific model. This step was critical in identifying the optimal model configurations.

Feature Importance Analysis

Upon training, an analysis was conducted for models where feature importance could be directly inferred. This analysis aimed to pinpoint the features (words or phrases from the TF-IDF vectorization) that played pivotal roles in the model's decision-making process. Understanding feature importance helped clarify how different aspects of the requirements data influenced the classification outcomes.

Confusion Matrix Visualization

To assess the models' predictive performance and identify potential areas of confusion between classes, confusion matrices were visualized for each model. This visualization provided a detailed breakdown of true versus predicted classifications, offering insight into each model's precision and its potential misclassification patterns

Experimentation Framework

An encompassing run_experiment function streamlined the process from data preprocessing and model training to evaluation. This modular approach allowed for flexibility in experimenting with different preprocessing parameters, thereby fine-tuning each model's performance based on the dataset's characteristics. The function's design facilitated an iterative optimization process, ensuring the best possible outcomes from each model.

The find_best_cleaning_params function is introduced to identify the optimal text preprocessing parameters. This function tests all combinations of preprocessing options to find the one that yields the best cross-validation score for the models. It streamlines the experimentation process, eliminating the need for manual parameter tuning and ensures that the preprocessing aligns perfectly with the model's learning algorithms for improved accuracy.

Results

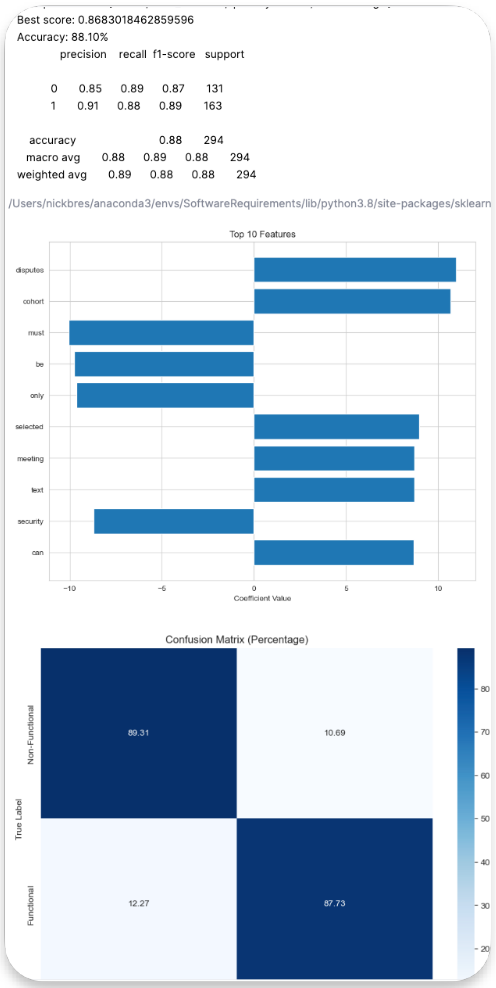
Determining Optimal Text Cleaning Parameters

The preliminary step in my analysis was to determine the most effective text preprocessing parameters to enhance the performance of our models. For this purpose, we designed an experiment to iteratively evaluate the impact of various text cleaning options on model accuracy.

Logistic Regression (LogR) Analysis example

For Logistic Regression, I observed that the variation in text preprocessing had a marginal effect on the model's accuracy. This suggests that Logistic Regression is quite robust to the presence of noise in textual data, or that the unique terms within our dataset inherently carry strong signals for classification regardless of the preprocessing steps.

To delve deeper, I examined the features deemed most significant by the model under two scenarios: without any text cleaning and with comprehensive cleaning that included deletion of numbers, stopwords, and lemmatization. This analysis is critical as it provides insights into which terms the model is using to make predictions and whether our text cleaning efforts are helping the model to focus on more relevant features.

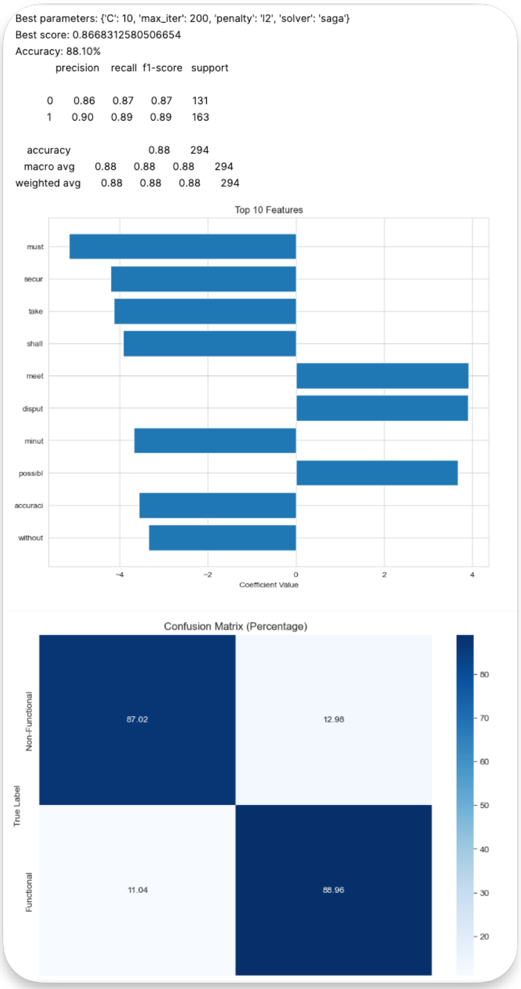


```
Best score: 0.8683018462859596
Accuracy: 88.10%
              precision    recall  f1-score   support

           0       0.85      0.89      0.87       131
           1       0.91      0.88      0.89       163

    accuracy                           0.88       294
   macro avg       0.88      0.89      0.88       294
weighted avg       0.89      0.88      0.88       294

/Users/nickbres/anaconda3/envs/SoftwareRequirements/lib/python3.8/site-packages/sklearn
```



```
Best parameters: {'C': 10, 'max_iter': 200, 'penalty': 'l2', 'solver': 'saga'}
Best score: 0.8668312580506654
Accuracy: 88.10%
              precision    recall  f1-score   support

           0       0.86      0.87      0.87       131
           1       0.90      0.89      0.89       163

    accuracy                           0.88       294
   macro avg       0.88      0.88      0.88       294
weighted avg       0.88      0.88      0.88       294
```

The top features identified in both scenarios can offer an interesting perspective. For instance, if we see a drastic change in the features' nature from uncleaned to fully cleaned data, it might indicate that our cleaning process is either introducing or eliminating noise. However, if the important features remain consistent across both scenarios, it implies that my text preprocessing has successfully stripped away irrelevant data without losing valuable information.

<u>Overall Conclusions on Text Preprocessing Experiment</u>

In assessing the influence of various text cleaning methods on model performance, an initial experiment was conducted without any preprocessing of the dataset. Subsequent iterations applied different preprocessing steps—lemmatization with all cleaned, stemming with all cleaned, number normalization, and stopword removal—individually, to determine their impact on each model's cross-validation score.

Model	**Not Cleaned**	**Lemmatization**	**Stemming**	**No Numbers**	**Numbers Normalized**	**No Stopwords**
LogR	86.5	87.5	86.5	86.6	86.5	86.6
KNN	84.7	82.2	82.7	84.3	85	83.1
DTC	76.7	80	80	76.5	76.2	80
SVC	87.5	88.1	86.9	87.5	87.2	86.8
NaiveBayes	87.5	87	86.2	87.1	87.2	86.5

The initial findings suggested that, while preprocessing steps did influence model performance, the degree of impact varied across models. For instance, Logistic Regression (LogR) demonstrated a modest increase in performance when most preprocessing steps were applied, suggesting that cleaner text can slightly enhance model accuracy. However, the improvement plateaued, indicating that after a certain point, further cleaning did not translate into significant gains.

An exhaustive search through all possible preprocessing combinations was then performed to determine the optimal method for each model. The results revealed a tailored approach where certain preprocessing steps were more beneficial for some models than others. For instance, Support Vector Machine (SVC) and LogR benefited from lemmatization and the inclusion of numbers, while Decision Tree Classifier (DTC) showed no significant preference towards lemmatization or stemming, but its performance was better without number normalization. NaiveBayes showed its best result without text cleaning.

Model	Lemmatize	Stemming	Delete Stopwords	Numbers	Normalize Numbers	Score
LogR	TRUE	FALSE	TRUE	TRUE	TRUE	87.7
KNN	TRUE	FALSE	FALSE	TRUE	TRUE	85
DTC	FALSE	FALSE	TRUE	TRUE	FALSE	80
SVC	TRUE	FALSE	TRUE	TRUE	FALSE	88.7
NaiveBayes	FALSE	FALSE	FALSE	TRUE	FALSE	87.5

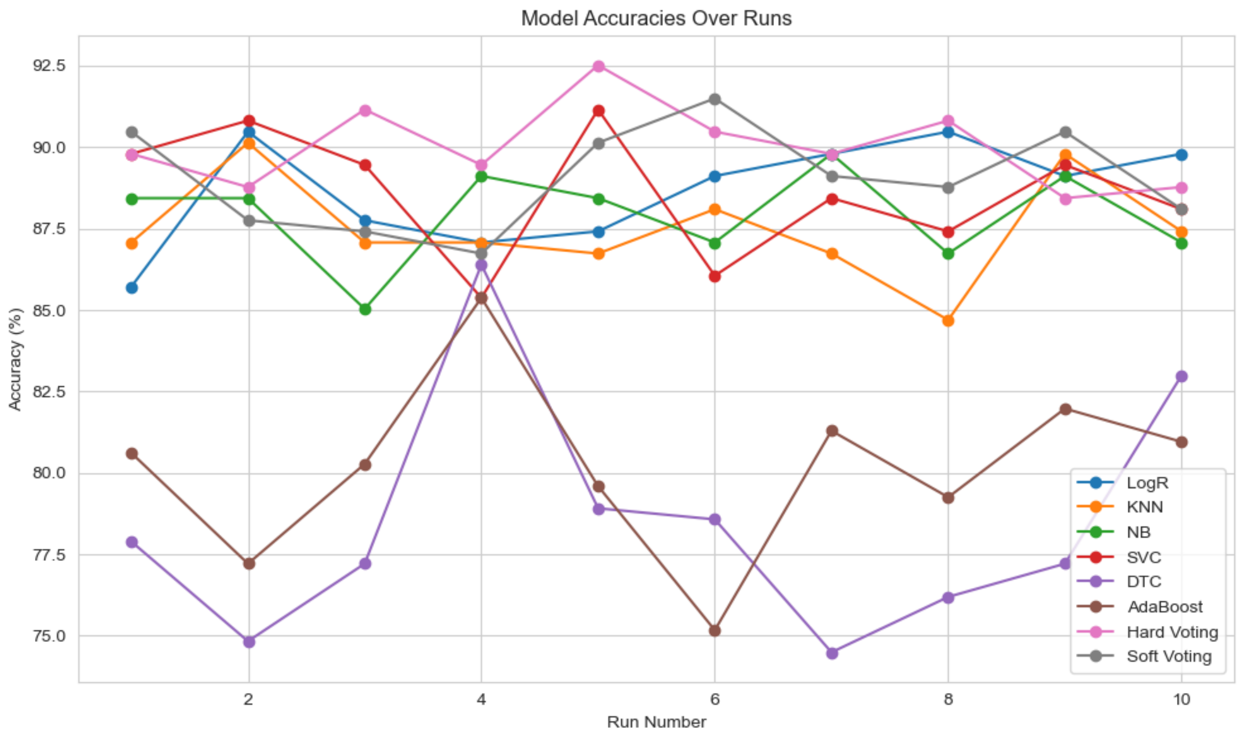
These observations point towards a nuanced understanding of how different models interact with textual data and the necessity of a customized preprocessing pipeline to maximize the predictive performance of each model.

Models Comparison

In the exploration of model efficacy, a comprehensive examination was conducted on eight distinct machine learning models. The ensemble included familiar classifiers such as Logistic Regression and K-Nearest Neighbors (KNN), along with more complex models like Support Vector Machines (SVC) and Naive Bayes. The Decision Tree model was evaluated both independently and in conjunction with AdaBoost to enhance its predictive power. Additionally, the collective capabilities of the models were harnessed through hard and soft voting classifiers, which aggregate the predictions from all models.

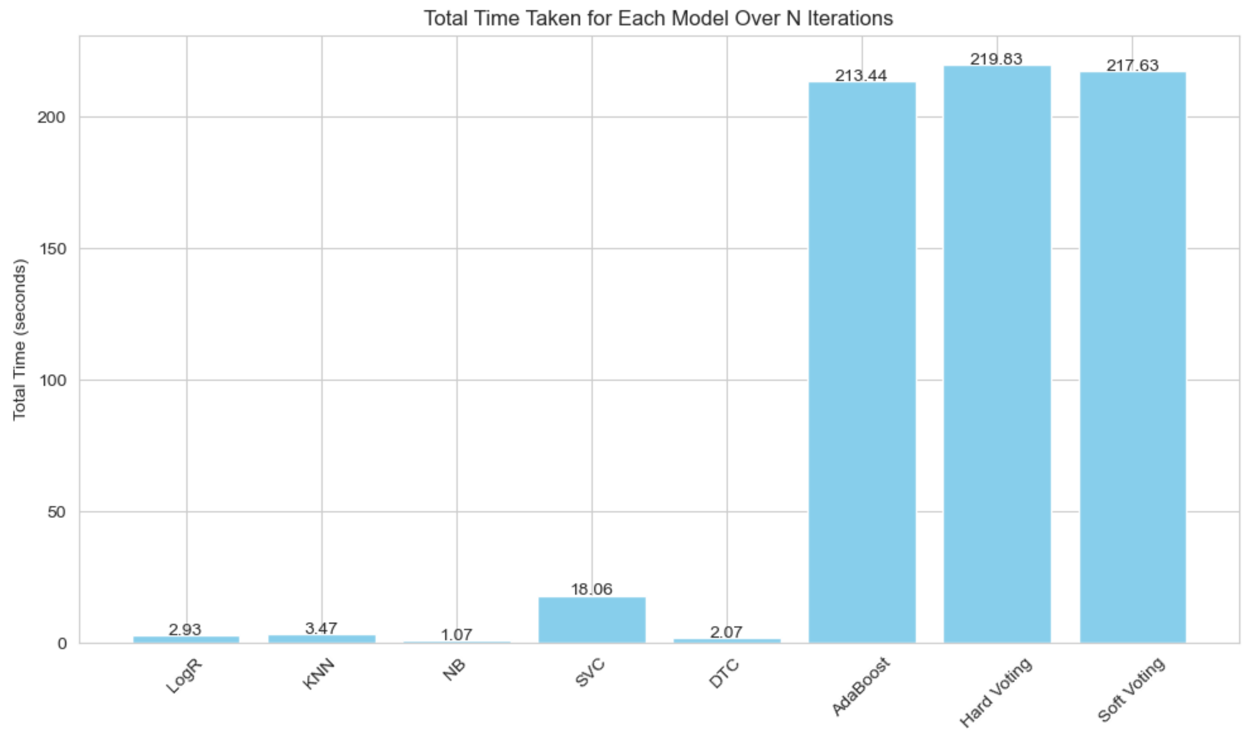
The models underwent multiple evaluations, with the dataset being split into a training set comprising 70% of the data and a test set making up the remaining 30%. This process was repeated n times to assess the stability and reliability of each model across different data splits.

An initial graphical analysis of model performance over 10 iterations provided a snapshot of stability, demonstrating that, generally, machine learning models produced consistent results on our dataset. The SVC emerged as the most stable, exhibiting minimal sensitivity to variations in the training data. In contrast, the Decision Tree model displayed a higher degree of variability, indicating a greater sensitivity to the data it was trained on.

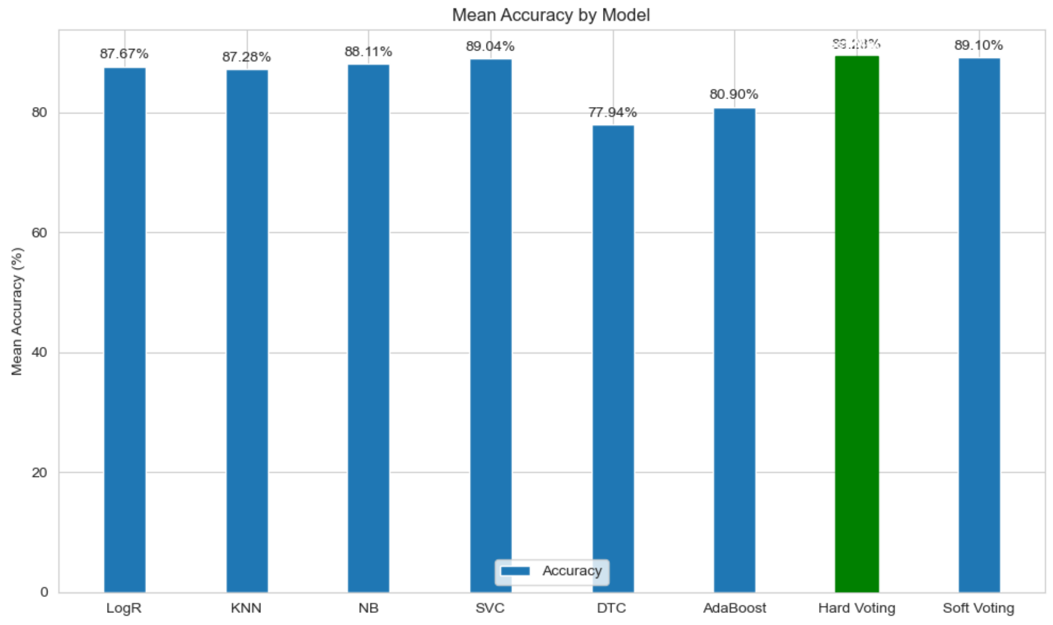


A more extensive test involving 100 iterations was implemented to determine the optimal model in terms of accuracy and computational efficiency. It was observed that the ensemble methods, such as AdaBoost and both hard and soft voting classifiers, consumed more time for training and evaluation. This increased duration can be attributed to their iterative nature and the complexity of combining predictions from multiple models. The SVC, while robust, also required a relatively longer time to train due to its computation-intensive nature, especially when working with higher-dimensional data.

In stark contrast, Naive Bayes stood out as the swiftest model. Its speed is largely due to its assumption of feature independence, simplifying calculations and thereby expediting the training and prediction processes.



The pursuit of the highest accuracy brought several models to the forefront, with the voting classifiers, SVC, and Logistic Regression exhibiting the strongest performance. KNN and Naive Bayes followed closely, delivering slightly lower but comparable accuracy. Despite improvements from AdaBoost, the Decision Tree model lagged behind, underscoring the limitations of its structure and the inherent challenges of preventing overfitting.



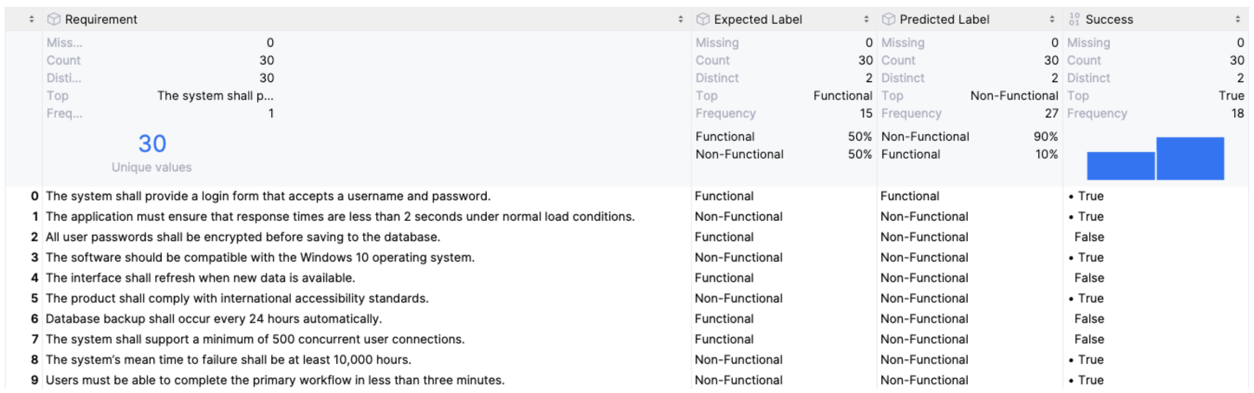
In summary, the choice of the best model depends on a trade-off between accuracy and computational time. If the highest accuracy is paramount, ensemble methods and SVC are superior, though they come with increased computational costs. For faster predictions, Naive Bayes offers a competitive alternative without significantly compromising on performance.

Manual Model Testing

The robustness and practicality of any predictive model are most convincingly demonstrated through its performance on novel, real-world data. To this end, manual testing with data not seen by the model during training is crucial. This approach ensures that the model's predictions are not merely a result of overfitting to the training data but are indicative of its ability to generalize to new, unencountered examples.

In the manual test conducted, a set of 30 newly crafted software requirements was used, which were equally split between functional and non-functional requirements. These requirements were meticulously composed to reflect realistic scenarios and challenges that a machine learning model might face in a production environment.

Upon testing these requirements against the best model, the results were promising yet provided insights into potential biases within the model. A success rate of 60% was achieved, indicating a decent level of accuracy. However, a pattern emerged where the model exhibited a proclivity for categorizing requirements as non-functional: 90% of the predicted categories fell into the non-functional class, with all errors occurring in the misclassification of functional requirements.



	Requirement	Expected Label	Predicted Label	Success
	Miss... 0	Missing 0	Missing 0	Missing 0
	Count 30	Count 30	Count 30	Count 30
	Disti... 30	Distinct 2	Distinct 2	Distinct 2
	Top The system shall p...	Top Functional	Top Non-Functional	Top True
	Freq... 1	Frequency 15	Frequency 27	Frequency 18
	30 Unique values	Functional 50% Non-Functional 50%	Non-Functional 90% Functional 10%	
0	The system shall provide a login form that accepts a username and password.	Functional	Functional	• True
1	The application must ensure that response times are less than 2 seconds under normal load conditions.	Non-Functional	Non-Functional	• True
2	All user passwords shall be encrypted before saving to the database.	Functional	Non-Functional	False
3	The software should be compatible with the Windows 10 operating system.	Non-Functional	Non-Functional	• True
4	The interface shall refresh when new data is available.	Functional	Non-Functional	False
5	The product shall comply with international accessibility standards.	Non-Functional	Non-Functional	• True
6	Database backup shall occur every 24 hours automatically.	Functional	Non-Functional	False
7	The system shall support a minimum of 500 concurrent user connections.	Functional	Non-Functional	False
8	The system's mean time to failure shall be at least 10,000 hours.	Non-Functional	Non-Functional	• True
9	Users must be able to complete the primary workflow in less than three minutes.	Non-Functional	Non-Functional	• True

Conclusion

This project aimed to shed light on the intricate process of classifying system requirements through the lens of machine learning. A multitude of models were meticulously evaluated, revealing the delicate balance between preprocessing intricacies and model performance. The study was anchored by a dataset that, while valuable, bore inherent limitations that impacted the outcome of manual testing and model robustness.

Dataset Limitations and Manual Testing Insights: The limited size and homogeneity of the dataset, likely sourced from a singular company or project and potentially authored by an individual, presented challenges. Manual testing with diverse, self-composed requirements yielded a modest 60% success rate, underscoring the model's difficulty in generalizing beyond the dataset's constraints. Future enhancements necessitate a more expansive and varied dataset to foster a model capable of capturing the multifaceted nature of system requirements across different domains.

Logistic Regression and SVC Success Factors: Logistic Regression's efficacy can be ascribed to its simplicity and the linear separability of the data. SVC's success is likely due to its ability to define a high-dimensional hyperplane that adeptly segregates classes. Both models' resilience to overfitting and their deftness in managing noise within the dataset emerged as crucial to their leading performance.

Challenges with Decision Trees: The relatively poorer performance of Decision Trees, even with the aid of AdaBoost, directs attention to the model's sensitivity to specific data distributions and a propensity to overfit. While AdaBoost aimed to rectify this by combining weak learners, the lack of data diversity likely hindered the realization of its full potential.

Envisioning Future Improvements: A primary avenue for future work involves amassing a larger, more diverse dataset that encapsulates a broad spectrum of requirements. This expansion would not only enhance the model's accuracy and generalizability but could also support the exploration of a finer-grained classification scheme with more nuanced categories. Advanced tuning and exploration of other classification algorithms may also contribute to a marked elevation in performance.

In sum, this project has been a testament to the capabilities of machine learning in the realm of software requirement classification, while also charting a course for future exploration. The key takeaway is the pivotal role of a rich and diverse dataset in developing models that are not only accurate but also robust and generalizable across various contexts.

Bibliography

[1] N. Breslavsky, "Project's GitHub page," [Online]. Available: https://github.com/NickBres/SoftwareRequirementsClassification.

[2] "Dataset," [Online]. Available: https://www.kaggle.com/datasets/iamvaibhav100/software-requirements-dataset?rvi=1.