Using Augmented Software Requirements for Automatic Classification

DANUT V. COPAE, University of Twente, The Netherlands

The recent growth in the number and complexity of software applications is producing an increasing need for high-quality software requirements. Manual classification of requirements can be a cumbersome and tedious process, which leads to leveraging Machine Learning and Natural Language Processing techniques for automated classification. This paper explores the effects of augmenting a software requirement dataset with synonymous words by using the *word2vec* word embedding technique and diverse feature extraction and classification methods. The experiment results show that the proposed augmentation technique improves the F1-score when using the Multinomial Naive Bayes and Logistic Regression classifiers by 0.57% and 0.88% respectively.

Additional Key Words and Phrases: machine learning, requirements, classification, word2vec, PROMISE_exp

1 INTRODUCTION

Requirements Engineering (RE) serves as one of the most important pillars in the development of software applications. This discipline involves the elucidation, analysis, verification, and documentation of user requirements usually conveyed in natural language [3]. A crucial part of this process is classification, which separates requirements into Functional Requirements (FR), which describe the important features of the project, and non-functional requirements (NFR), which specify quality constraints and attributes. The resulting documentation is called the Software Requirements Specification, which represents an essential aspect in the success of a software project [2].

Numerous studies [4, 22] show that although it's possible to classify requirements manually, it is a time-consuming and error-prone process, for reasons such as depending on the analyst's understanding of the topic or analysts using different terminologies for the same underlying concept. Hence, recent works in the domain of Machine Learning (ML) have analysed algorithms for efficient automatic classification of software requirements.

Text classification algorithms are usually deployed for categorizing data expressed in natural language. The supervised learning algorithms of this type label textual documents with predefined classes. In traditional approaches, statistical methods are used to represent a document using vectors, with the relationships and semantics between the words being ignored [1]. However, these techniques can disregard useful information that can be used to improve classification performance. For example, two words constituted of different letters that have a synonymous meaning are treated as separate entities in the vector space. Since the effectiveness of the classification performance is largely limited by the quality of the features extracted [14], the traditional methods allow further refinements. To mitigate these drawbacks, this paper aims to evaluate the efficiency of text classification with semantic features, by answering the following research questions:

RQ1: What are the most efficient feature extraction methods to classify software requirements in Functional Requirements and subclasses of Non-Functional Requirements?

RQ2: Which classification algorithms perform best to classify software requirements in Functional Requirements and subclasses of Non-Functional Requirements?

RQ3: How do the previous best feature extraction and classification methods change when the dataset is augmented with several similar words?

RQ4: How does the augmentation of a dataset with similar words impact the overall classification performance?

In this work, we augmented the PROMISE_exp requirements dataset with synonymous words to analyse the effects on classification performance and the best algorithms choice. To accomplish this, we implemented an array of feature extraction and classification techniques to find the combination with the highest efficiency. Then, we augmented the dataset by utilizing the similarity scores provided by the *word2vec* word embedding technique. Lastly, we applied the same feature extraction and classification techniques to the augmented dataset to assess how the results have changed.

The rest of the paper is structured as follows. Section 2 presents past works and developments in classifying requirements and the usage of the *word2vec* tool. Section 3 introduces the concepts and implementation techniques that were used to conduct the research process. The findings of the study are presented in Section 4, together with an analysis of the results. Section 5 presents the possible limitations of the study and the threats to validity. Lastly, the results are summarized, and the future work is presented in Section 6.

2 LITERATURE REVIEW

This paper is based on previous work in the field and aims to reproduce the results whenever possible, to build a stable base for answering the research questions. More specifically, we will try to replicate the results from Dias, Cordeiro et al. [6], who evaluated the classification performance of algorithms on the PROMISE_exp dataset. They have extracted features using the Bag of Words (BoW) and Term Frequency—Inverse Term Frequency (TF-IDF) vectorization techniques and used the Chi^2 method to remove insignificant words. For classification, the authors used Support Vector Machine (SVM), Multinomial Naive Bayes (MNB), k-Nearest Neighbours (kNN) and Logistic Regression (LR). Based on the F-measure, the greatest performance was achieved by TF-IDF followed by LR, with a classification performance of 78%.

J. Slankas and L. Williams [24] have extended the regular PROMISE dataset [5] with various documents such as data use agreements, install manuals, regulations, requirements specifications and user manuals, creating an extensive corpus of 11876 requirements. Intending to classify the NFR in the documents into 14 categories (Availability, Legal, Maintenance, etc.), they obtained the highest F1 measure of 62.3% using an SVN classifier. Although the F1 score is

TScIT 37, July 8, 2022, Enschede, The Netherlands

16% lower than the results from Dias, Cordeiro et al. [6], most of this study's requirements were not stated formally but extracted from large blocks of text. Thus, it gives baseline expectations on how well ML algorithms can generalize to requirements outside formal software requirements specification documents.

The *word2vec* approach has been used by other scientists as well. Lu et al. [17] have tested the efficiency of augmenting user reviews with the *word2vec* tool, with the scope of classifying them in FRs, four types of NFRs (Usability, Reliability, Portability, Performance), and Others. The method proposed by them consists in concatenating the training data set with several words that are similar to the original reviews and applying BoW vectorization. Using three different classifiers (Naive Bayes, J48 and Bagging) they observed improvements in the F1 measure ranging between 1.4% and 2.4%.

Joseph et al. [15] have analysed a newsgroup text dataset of roughly 18000 samples, to categorize it into 20 different topics. The authors found that when they concatenated the base document with the TF-IDF weighted similar words provided by *word2vec* and used a linear SVM for classification, the accuracy was improved by 1.4%. This improvement, however, was not consistent across all the categories, since, among a few of them, the method resulted in a hundredth of a percentage lower performance.

Lastly, we have found two writings that investigate a topic similar to our first and second research questions. The first one is a Master's thesis recently written by N. Thuy [19], which analyses whether ML algorithms intended for functional requirements can be used for non-functional requirements and vice-versa, and if these methods can be extended to classify both types of requirements. Using a total of 20 different ML methods applied to a dataset consisting of 1838 requirements, the author found that 15 out of 20 methods can be successfully applied for requirement types they are not intended for, and 17 out of 20 can be applied for classifying mixed requirements. The second work is written by A. Mitrevski [18] and aims to reproduce the findings from Dias, Cordeiro et al. [6], while adding additional classification methods such as Gradient Boosted Decision Trees (GBDT) and Recurrent Neural Network (RNN).

3 METHODOLOGY

In this section, we will discuss the methods used in the research project and their role in the data workflow. As seen in Figure 1, the first step is normalizing the PROMISE_exp dataset. This step is particularly important since the normalized dataset will act as input for all the remaining steps. We achieve this by converting the words to lower case, removing stop-words and applying lemmatization. Secondly, we will create the augmented dataset using the *word2vec* tool. The next steps onwards are applied for both the original PROMISE_exp dataset and the one that is augmented.

Features will be extracted using the BoW and TF-IDF vectorization techniques and reduced using the DF, *Chi*^{2} and ANOVA statistical significance tests. Then, for each distinct set of word vectors, we will be applying the MNB, LR, SVN and kNN classification algorithms and determine the performance based on the F1-score. Finally, we will compare the performances originated from the initial dataset and the augmented one, and draw conclusions from the results.

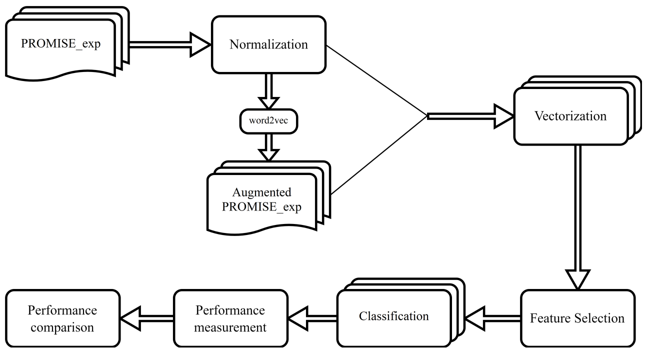

Fig. 1. Data workflow.

Table 1. Requirement labels classification

Requirement	Label	Count
Functional Requirement	F	444
Availability	A	31
Legal	L	15
Look and feel	LF	49
Maintainability	MN	24
Operability	O	77
Performance	PE	67
Scalability	SC	22
Security	SE	125
Usability	US	85
Fault Tolerance	FT	18
Portability	PO	12
Total		969

3.1 Dataset overview

This paper uses the PROMISE_exp, which is based on the tera-PROMISE dataset [5], that Lima et al. [16] extended using several software requirements specification documents. The extended corpus increased the total number of requirements to 969, compared to the original dataset which contains only 625. When looking at the class distribution of this dataset in Table 1, it can be observed that the number of Non-Functional Requirements types is highly unbalanced. The least represented NFR type, Portability, composes 1.2% of the dataset, while Security, the most represented one, constitutes 12.9%.

3.2 Normalization

A. Uysal and S. Gunal showed in their article [26] that when it comes to textual classification, it is always recommended to perform extensive pre-processing methods, the most significant technique being stop-word removal. For this reason, the dataset was normalized by converting the words to lower case, removing "stop words" i.e, *"what", "to", "I", "because"*, numbers and punctuations. To get a consistent form of words, lemmatization was also used to replace words with their dictionary form, using the NLTK [1] library.

[1]https://www.nltk.org/

3.3 Word2vec

Word2vec is a technique utilized for Natural Language Processing that applies a multi-layer neural-network model to learn associations between words. In this technique, each word is represented by a vector in a space built such that similar words are close to each other. Due to this structure, *word2vec* provides excellent performance in finding the similarity between words and suggesting additional words [11], usually by computing the cosine similarity between the vectors. However, when it comes to text classification, this technique has one disadvantage: it is not aware of the importance of words within the document. To mitigate this, we will test different methods to weight the scores reported by the library.

At the time of writing, the most extensive pre-trained model of *word2vec* is provided by Google [2]. This model has been trained on 100 billion words originating from Google News posts, uses a skip-gram model with a vector size of 300 and has been successfully used in previous studies [21] that focus on deep learning approaches.

3.4 Augmentation method

The augmentation method proposed by Lu et al. [17] consists of appending several similar words to the user requirement. To determine the similarity between a requirement and a potential similar word, the following formula from Li et al. [13] will be used:

$$sim(r_k, t_j) = \sum_{t_i \in r_k}^{n} (w_i * sim(t_i, t_j))$$

Here, r_k represents a requirement *k*, t_j a term *j* chosen from the corpus and $sim(t_i, t_j)$ is the similarity score between the terms *i* and *j*, as computed by *word2vec*. Finally, to get the similarity score between the requirement r_k and the term t_j, we have to compute the weighted sum consisted of the similarity between t_j all the terms t_i in r_k.

For the weighting method, we considered the matrixes resulting from the BoW, TF-IDF and the custom OKAPI Best Matching weight formula, taken from [13]. These weights are referred to in the formula by w_i. In all combinations between vectorization techniques and classification methods, using TF-IDF as the weights for the words consistently facilitated the highest results. These findings correlate with the results found by Joseph et al. [15], where the authors also used the TF-IDF formula for the weights.

Using the formula explained above, for each requirement, we go through all the unique terms in the corpus and calculate the similarity score between the requirement and the unique term. Then, we order the terms descending based on the score, and add only the most important ones to the requirement.

The number of added words is given by Equation 1 [17], where θ refers to a variable increasing from 0 to 2 with an increment of 0.1 and *user_requirement_length* is given by the number of words in the original requirement. All the 20 possible values for θ were tested, and θ between 0.2 and 0.25 gave the most significant results for all the classification algorithms.

$$N = \theta * user\_requirement\_length \tag{1}$$

[2]https://code.google.com/archive/p/word2vec/

This method can be illustrated using the preprocessed user requirement *"system shall refresh display every second"*. By using the TF-IDF weights, the most similar words for this requirement would be: *"first", "another", "next", "entire", etc.* Using $\theta = 0.2$ for the *N* formula, we need to pick $0.2 * 6 = 1.2$ words. Finally, after rounding *N* to the nearest integer, the new requirement becomes *"system shall refresh display every second first"*.

3.5 Feature extraction

3.5.1 BoW. Bag of Words [20] is one of the most common methods for vectorizing text documents, due to its high performance and simplicity. This technique essentially creates a histogram, which is a vector of the number of unique word references, such that each word has a weight equal to the number of occurrences as found through the document. One drawback of this method is that it is only a lexical method and does not consider the number of documents that contain a term, or the relationship between words.

3.5.2 TF-IDF. Term Frequency—Inverse Document Frequency [23] aims to normalize the BoW method, by multiplying the frequency of a term in a requirement and the inverse document frequency for each term. This can be seen in Equation 2, where $f_{w,r}$ is the number of times the word *w* appears in requirement *r*, *D* is the corpus, and *|D|* is the size of the corpus (the total number of requirements).

$$TF\text{-}IDF(term_{w,r}) = f_{w,r} * log(|D| f_{w,D}) \tag{2}$$

3.6 Feature selection

Feature selection has the purpose of retaining as much accuracy as possible, in some cases even improving it, while greatly reducing the time and complexity of ML algorithms. To accomplish this, a dependence relationship is identified between each feature and the target class. Afterwards, features, or in the context of textual classification, words, that are independent of the target are discarded.

The main method for feature selection used in this study is Univariate Feature Selection, which utilizes statistical tests to select the features that convey the most information. In this method, each feature gets a score, and only the ones with the highest mark will be retained in the dataset. This will be implemented using the *SelectKBest* function from the *scikit* [3] library, which, for each term and class, returns the score of the statistical test and a probability value used to test the significance of the relationship.

According to the overview study about feature selection methods in textual classification conducted by Yang et al. [27], Document Frequency (DF), Information Gain (IG) and Chi-square scores have a strong correlation, each of them giving excellent performance. For this reason, we will be using DF and Chi-square. Additionally, we will utilize the F-score ANOVA test. Although it has not been applied in requirements engineering in the past, it has been used successfully to select features for other classification purposes [7] [12].

3.6.1 Document Frequency. Document Frequency [8] is a simple method that measures how many documents a given word appears in. This method is included in the BoW and TF-IDF algorithms

[3]https://scikit-learn.org/stable/

implementations as two different parameters: minimum frequency (min_df) and maximum frequency (max_df) setting. As explained in Yang et al. [27], DF seems to favour common terms over rare terms. Consequently, in this approach, we will only be restricting features solely based on the minimum number of documents that they appear in.

For BoW, the most optimal value for *min_df* is 0.002. This means that if a word appears in less than 0.2% of documents, we will discard it. Considering that the corpus used has 969 requirements, this implies that each word has to be present in at least two requirements. However, for TF-IDF, we found that the most optimal value for *min_df* is 1, meaning that no features should be discarded. This is expected since TF-IDF is already making the necessary weight adjustments based on how many documents a word appears in.

3.6.2 Chi-squared. Chi-squared (χ^2) is a test used to measure the dependence between a term t and a classification class c. According to Yang et al. [27], the formula for this test is the following:

$$\chi^2(t,c) = \frac{N * (A * D - C * B)^2}{(A+C) * (B+D) * (A+B) * (C+D)}$$

where N is the total number of requirements, A is the number of times the term t occurs in class c, B is the number of times t occurs outside c, C is the number of times c occurs without t and D is the number of times neither c nor t occurs.

All the terms with a statistical significance under a value α are picked. For this study, the greatest results were given for $\alpha = 0.085$ for BoW and $\alpha = 0.82$ for TF-IDF.

3.6.3 ANOVA. The analysis of variance (ANOVA) test is used to determine how different factors affect a given dataset, by making comparisons in the means of a categorical variable in three or more independent population samples. The formula for this test is given in Equation 3, where *between-group variance* represents the variation between the means of all the samples, and *within-group variance* is the variation within each sample. The resulting number is formally called the *F-value*, as the equation is using the F-statistic, which is simply the ratio between two variances.

$$\text{ANOVA F-value} = \frac{\text{between-group variance}}{\text{within-group variance}} \tag{3}$$

In the context of textual classification, the F-value ANOVA test is used to determine how discriminative a word is for a given class. Consequently, features that are independent of the target class can be removed.

Just like in the Chi-square approach, the α value has been adjusted such that the highest F1 measure is obtained. The greatest results are achieved by using $\alpha = 0.17$ for BoW and $\alpha = 0.75$ for TF-IDF.

3.7 Classification

Most software requirements specification documents contain mixed FR and NFR requirements. For this reason, we focused on a general classification method, that analyses FR together with subclasses of NFR. Considering the structure of the PROMISE_exp corpus (Table 1), we classified requirements into 12 classes.

Due to the limited number of requirements available in the dataset, cross-validation was used to split the data into training and testing data. Furthermore, we noticed that the dataset is highly imbalanced in the number of NFR classes: there are 125 security requirements, but only 12 about portability. We adjusted for this imbalance by using a Stratified 10-Fold Cross-validation. In stratified sampling, instead of randomly picking requirements, the distribution of the classes from the dataset will be preserved inside the individual folds as well.

According to the literature review on the current state of the practice in *"Machine Learning & Requirements Engineering"* created by Iqbal et al. [10], Logistic Regression (LR), Multinomial Naive Bayes (MNB), Support Vector Machine (SVM) and k-Nearest Neighbours (kNN) are the most widely used techniques for classifying software requirements. For that reason, we adopted the same methods in this research as well.

3.7.1 Logistic Regression. Logistic Regression (LR) uses a regression model to estimate the probability that a data sample belongs to a target class. To achieve this, it is using the sigmoid function to compute a probability between 0 and 1, and a decision boundary, which represents a threshold above which the data sample will be classified in a certain class.

3.7.2 Multinomial Naive Bayes. Multinomial Naive Bayes (MNB) is a specialized version of the Naive Bayes algorithm, used to classify data which cannot be expressed numerically. This method computes the probability of a data sample belonging to a class and then chooses the class with the highest probability.

3.7.3 Support Vector Machine. Support Vector Machine (SVM) is an algorithm capable of solving both linear and non-linear classification problems. To achieve this, it is creating an optimal hyperplane in N-dimensional space (N representing the number of classes) that separates the data into their different categories.

3.7.4 K-Nearest Neighbours. K-Nearest Neighbours (kNN) classifies a data sample by computing the distance of the new data with all the other existing points, and the nearest K points are used to make a prediction, assuming that similar points are close to each other.

3.7.5 Model selection. To make a fair comparison between classification and vectorization methods, everything needs to be operating at the highest standard. Hyperparameter tuning aims to achieve this goal, by selecting the best possible parameters for learning algorithms. Using the *scikit* library, we passed a parameter grid and a classifier to the *GridSearchCV* class. This class tested each combination of parameters from the grid and returned the one with the highest performance for the given classifier. To come up with the grid of parameters, we started with an initial wide range of values and then narrowed the range on further iterations.

Due to the scarcity of the available data, the hyperparameter tuning was performed on the whole dataset. This usually produces over-optimistic results, since the hyperparameters will overfit the data. Holding out a separate portion for testing was not an option because a significant portion of the dataset would become unused, which would result in a high variance in the results. Nested cross-validation (Nested-CV) is another possibility to resolve this. In nested cross-validation, each outer loop has another inner loop used solely for finding the hyperparameters, thus reducing the risk of overfitting. However, this method is only used theoretically to indicate generalization performance, as in deployed applications it's not possible to use multiple sets of hyperparameters. Furthermore, the increased training time would prove unsuitable for some classifiers.

For each algorithm, the following hyperparameters were considered:

MNB - *'alpha'* represents the additive smoothing parameter. Little to no smoothing was preferred, with values ranging from 0.001 to 0.1.

LR - *'class_weight'* was set to *balanced*, to automatically adjust the weights based on the class frequencies. Then, *'C'*, the regularization strength was tested for values between 0.1 and 10. Higher values of *'C'* instruct the model to put more weight on the training data, while lower values will put a higher penalty on the parameters. The best value of *'C'* varies across the vectorization methods, resulting in a wide range of values being considered.

SVN – Two hyperparameters were evaluated. The first one is the *'kernel'*, a function that solves non-linear problems, which considers two values: *linear* and *rbf.* The *linear* kernel was preferred in all cases. The second hyperparameter, *'C'*, represents the regularization parameter and has the same function as described above in *LR*. The results show that there is not a single preferred value, and the entire range $[0.1, 10]$ needs to be considered to find the optimal value.

kNN - *'n_neighbours'* represents the number of nearest neighbours that are considered by the algorithm. Surprisingly, the classifier performs best when $n\_neighbours = 1$, which represents the most complex kNN model. Considering the simplicity of the dataset, this likely leads to overfitting and lower performance, as the algorithm tries to generate models that are too complex and do not generalize well to the test data. Additionally, three values for the *'algorithm'* hyperparameter are considered: *ball_tree, kd_tree and brute*, with the most optimal value depending on the vectorization methods.

3.8 Performance measurement

Performance metrics are mathematical formulas used to evaluate the performance of classification algorithms. The *"correct"* metric to use can differ between settings [25]; in some cases, it might be more important to correctly identify positive examples, whereas in others, correctly identifying negative examples could be vital.

In the application of text classification [25], *Precision*, *Recall* and *F1-Score* are three common performance measurements. Precision (Equation 4) represents how many samples classified as X actually belong to class X. On the other hand, recall (Equation 5) corresponds to how many samples belonging to class X were correctly classified as X.

Since precision and recall are not very useful metrics when isolated, the F1-Score (Equation 6) combines these two in a harmonic mean. Unlike the F2-score which weights recall higher than precision, the F1-score gives the same weight to both. Since this metric is often used to compare the performance of two classifiers [9], it is the main performance measurement method in this research as well.

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$\text{F1-score} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{6}$$

4 RESULTS AND DISCUSSION

In this section, we will present the results of the research and their implications for the Requirements Engineering field. The results obtained from the PROMISE_exp dataset are presented in Table 2 and will be referred to as *"Baseline Results"*. When we compare them with the *"Classification with 12 Granularities"* table from Dias, Cordeiro et al. [6], we can see that they are very similar, with only a couple of points difference, which is most likely the effect of different implementations and hyperparameters being used. Since the unmodified and the augmented dataset use the same methodology, the reproducibility with older studies offer a strong base of support for the results to come.

In the baseline results, the absolute highest performance of 76.6% was achieved by combining TF-IDF vectorization, the ANOVA test for feature selection and MNB classification. When it comes to the best feature selection method, TF-IDF coupled with DF gets the highest average score. However, when we exclude the kNN classifier, which consistently performed worse than the other classifiers and got an abnormally high value on TF-IDF and DF, the best feature selection method is in line with the highest value: TF-IDF and ANOVA. Then, looking at the classifiers, LR performs on average the best, with a performance of 75.7%, closely followed by MNB with 75.5%.

The results of the word augmentation can be seen in Table 3. To easily compare the values with the original ones, higher F1 scores from the same row and column are marked with bolded text. Similar to the baseline results, the TF-IDF, ANOVA and MNB combination still managed to achieve the greatest performance, with 77.6%. When we look at the values of the feature selection methods in the augmented dataset, and we do not take kNN into account, TF-IDF with ANOVA remains the best feature selection method. Additionally, LR remains the most performant classifier with 76.4%, closely followed by MNB with 75.9%. It can be observed that the combinations of classifiers and vectorization methods which performed good on the baseline dataset, also performed well on the augmented dataset.

These results suggest that the augmentation of similar words can provide benefits in non-complex classification methods, such as MNB and LR. Surprisingly, we were unable to match the 1.4% improvements observed by Joseph et al. [15] with the use of *word2vec* and linear SVM. However, this lack of improvement is likely the cause of a significantly smaller dataset used by us (the authors used roughly 18,000 samples, while we had 969), and not because of the relationship between SVN and *word2vec*. This is further explained below.

The average difference between the augmented and baseline results for each classification algorithm is as follows: MNB: *+0.57%*; LR: *+0.88%*; SVN: *-0.62%*; kNN: *-0.05%*. Overall, the classification was

Table 2. Baseline results

	DF		Chi^2		ANOVA	
	BoW	TF-IDF	BoW	TF-IDF	BoW	TF-IDF
MNB	**0.749**	0.747	0.763	0.740	0.762	0.766
LR	**0.765**	0.758	0.754	0.751	0.751	0.762
SVN	**0.752**	**0.757**	0.742	0.74	**0.742**	0.745
kNN	0.597	**0.691**	**0.591**	**0.583**	0.585	0.517

Table 3. Augmented results

	DF		Chi^2		ANOVA	
	BoW	TF-IDF	BoW	TF-IDF	BoW	TF-IDF
MNB	0.743	**0.748**	**0.770**	**0.742**	**0.774**	**0.776**
LR	0.753	**0.767**	**0.763**	**0.763**	**0.766**	**0.769**
SVN	0.733	0.749	**0.743**	**0.743**	0.734	**0.748**
kNN	**0.609**	0.675	0.581	0.581	**0.591**	**0.525**

improved in the MNB and LR algorithms, stayed almost the same in kNN, and was slightly decreased in SVN. All the differences are under 1%, due to the small number of words that were added. After removing the stop words, the dataset contains 1566 unique words, creating a restricted pool of words available for augmentation. Consequently, picking more than $0.2 * user\_requirement\_length$ words resulted in a significantly lower F1 score. This is because *word2vec* calculates the similarity between words using Cosine similarity, and has no underlying knowledge of the probability of two words belonging to the same class. Using different criteria for adding words and classification is thus likely to create incorrect relationships between features and classes, and to even weaken correct correlations.

This idea is exemplified by analysing the number of features selected by Chi^2. In TF-IDF combined with Chi^2, the number of features was reduced from 505 to 470. In this combination, the most important word was *"second"*, with $\chi^2 = 498.4$. After augmentation, the same word was still the most important one, but with a lower score: $\chi^2 = 465.29$. The reasoning for the lower score is the following: in the baseline document, the word *"second"* is used 48 times, of which 34 times on PE requirements. However, in the augmented document, the same word is used 53 times, but only on 35 PE requirements. Consequently, we can see how the importance of this word is lowered by adding it to the requirements of other types. A similar argument can be made for the other classifier and vectorization combinations, where more features are selected after the augmentation. In those cases, words with previously low importance are added to multiple requirements, due to their high *word2vec* similarity score.

Figure 2 illustrates the confusion matrix of the algorithm with the most significant performance (in this case the augmented dataset with TF-IDF, ANOVA and MNB). Generally, the requirements types that have a limited amount of available samples, portability (PO) and fault tolerance (FT) have a lower F1 score than the others, due to a low recall score. This means that when a requirement is predicted to belong to these classes, the prediction is generally good (high precision), but only a trivial part of the requirements belonging to

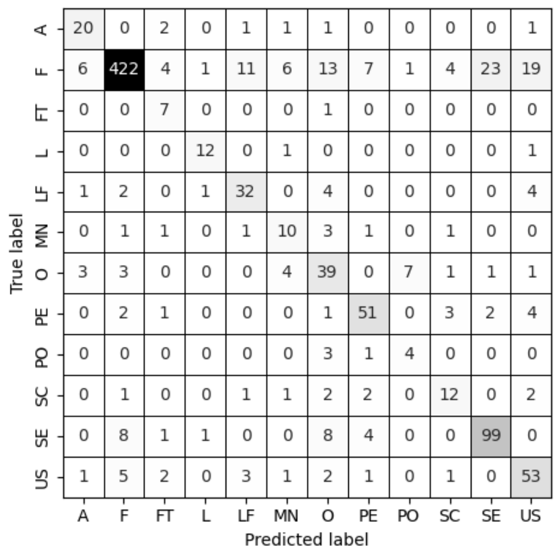

Fig. 2. Confusion Matrix

PO and FT were correctly retrieved (low recall). A possible cause for this is that words such as *"server"*, *"standard"*, and *"operate"* are used in PO and FT requirements, and also used in many other requirements types with many more samples. Consequently, these words are more likely to be associated with other classes.

5 LIMITATIONS AND THREATS TO VALIDITY

One of the limitations of this study is given by the dataset size. Since a larger corpus creates a more significant pool of unique words, the probability of appending the same word to multiple requirements is likely to diminish. This would suggest that the performance of the word augmentation technique is directly proportional to the size of the dataset and the richness of the vocabulary. However, this assumption would need to be further tested.

Another limitation is represented by the uneven distribution of classes in the dataset. Even if various weighting techniques were used to mitigate this issue, generally, the classes that had fewer samples showed lower performance than their counterparts. With a higher number of Portability, Legal, and Fault Tolerance samples, the F1-score for these classes should be improved, as the classification algorithms will be better in distinguishing their unique properties.

Lastly, it may be possible that the observed results are a by-product of a lucky combination of methods and parameters, and that the results cannot be reproduced in other scenarios. To reduce this chance, further studies that analyse the effect of augmenting a software requirement dataset with similar words would need to be conducted in the future.

6 CONCLUSION

In this paper, we used 2 vectorization techniques: BoW and TF-IDF, 3 tests for dimensionality reduction: DF, Chi^2 and ANOVA and 4 classification algorithms: MNB, LR, SVN and kNN, to categorize software requirements in 12 different types: FR and 11 NFR classes.

We found that simple classification algorithms such as LR and MNB perform best, and also benefit the most from the augmentation technique, while more complex classifiers such as SVM or kNN create relationships and formulas that do not generalize well to unseen test data. Additionally, due to the low number of samples in the dataset, the augmentation technique obtained improvements of under 1% in two classification algorithms.

Answering *RQ1*, we found that TF-IDF is the most performant vectorization method, as measured by the F1-score metric. Additionally, it performs best when the features are selected using the ANOVA statistical test. When it comes to the *RQ2*, the best classification method is LR, closely followed by MNB, with only 0.2 points difference in performance.

The next aim was to determine whether appending similar words to the end of user requirements can boost classification performance. We have utilized the *word2vec* library to determine the similarity between words using cosine similarity and weighted the scores by the TF-IDF formula. We found that TF-IDF remains the best vectorization method, while LR, followed by MNB, also resemble classifiers with the highest performance. Answering *RQ3*, the augmentation method maintained the order of efficiency in the algorithms.

The augmentation of similar words using *word2vec* favours simple algorithms such as Multinomial Naive Bayes and Logistic Regression. Answering *RQ4*, we obtained +0.57% and +0.88% improvements in these classification methods, and a performance decrease of -0.62% in SVM and -0.05% in kNN. Considering that older papers found improvements of 1.4% to 2.4% on datasets with larger samples using a similar *word2vec* augmentation technique, we suggested that the classification performance is likely dependent on the size and diversity of words of the underlying dataset.

In the future, we aim to broaden the research in the following ways:

(1) Extend the PROMISE_exp dataset with additional Software Requirements Specification documents, to test whether the augmentation technique described in this paper can produce even higher improvements.
(2) Explore further techniques and methodologies that help unbalanced classes with a low number of samples achieve higher performance.

REFERENCES

[1] Berna Altınel and Murat Can Ganiz. 2018. Semantic text classification: A survey of past and recent advances. *Information Processing & Management* 54, 6 (2018), 1129–1153. https://doi.org/10.1016/j.ipm.2018.08.001

[2] Mina Attarha and Nasser Modiri. 2011. Focusing on the importance and the role of requirement engineering. In *The 4th International Conference on Interaction Sciences*. 181–184.

[3] Jean-Louis Boulanger. 2016. 10 - Requirement Management. In *Certifiable Software Applications 1*, Jean-Louis Boulanger (Ed.). Elsevier, 239–282. https://doi.org/10.1016/B978-1-78548-117-8.50010-6

[4] Kun Chen, Wei Zhang, Haiyan Zhao, and Hong Mei. 2005. An approach to constructing feature models based on requirements clustering. In *13th IEEE International Conference on Requirements Engineering (RE'05)*. 31–40. https://doi.org/10.1109/RE.2005.9

[5] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. 2006. The Detection and Classification of Non-Functional Requirements with Application to Early Aspects. In *14th IEEE International Requirements Engineering Conference (RE'06)*. 39–48. https://doi.org/10.1109/RE.2006.65

[6] Edna Dias Canedo and Bruno Cordeiro Mendes. 2020. Software Requirements Classification Using Machine Learning Algorithms. *Entropy* 22, 9 (2020). https://doi.org/10.3390/e22091057

[7] Soufiane El Mrabti, Mohammed Al Achhab, and Mohamed Lazaar. 2018. Comparison of feature selection methods for sentiment analysis. *Communications in Computer and Information Science* 872 (2018), 261 – 272. https://doi.org/10.1007/978-3-319-96292-4_21

[8] George Forman. 2003. An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *The Journal of Machine Learning Research* 3 (2003), 1289–1305.

[9] Aurelien Geron. 2019. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (2nd ed.). O'Reilly Media, Inc.

[10] Tahira Iqbal, Parisa Elahidoost, and Levi Lúcio. 2018. A Bird's Eye View on Requirements Engineering and Machine Learning. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. 11–20. https://doi.org/10.1109/APSEC.2018.00015

[11] Derry Jatnika, Moch Arif Bijaksana, and Arie Ardiyanti Suryani. 2019. Word2vec model analysis for semantic similarities in English words. *Procedia Computer Science* 157 (2019), 160 – 167. https://doi.org/10.1016/j.procs.2019.08.153

[12] Mukesh Kumar, Nitish Kumar Rath, Amitav Swain, and Santanu Kumar Rath. 2015. Feature Selection and Classification of Microarray Data using MapReduce based ANOVA and K-Nearest Neighbor. *Procedia Computer Science* 54 (2015), 301 – 310. https://doi.org/10.1016/j.procs.2015.06.035

[13] Cheng Hua Li, Ju Cheng Yang, and Soon Cheol Park. 2012. Text categorization algorithms using semantic approaches, corpus-based thesaurus and WordNet. *Expert Systems with Applications* 39, 1 (2012), 765–772. https://doi.org/10.1016/j.eswa.2011.07.070

[14] Qian Li, Hao Peng, Jianxin Li, Congying Xia, Renyu Yang, Lichao Sun, Philip S. Yu, and Lifang He. 2022. A Survey on Text Classification: From Traditional to Deep Learning. 13, 2, Article 31 (apr 2022), 41 pages. https://doi.org/10.1145/3495162

[15] Joseph Lilleberg, Yun Zhu, and Yanqing Zhang. 2015. Support vector machines and Word2vec for text classification with semantic features. In *2015 IEEE 14th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC)*. 136–140. https://doi.org/10.1109/ICCI-CC.2015.7259377

[16] Márcia Lima, Victor Valle, Estevão Costa, Fylype Lira, and Bruno Gadelha. 2019. Software Engineering Repositories: Expanding the PROMISE Database. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering* (Salvador, Brazil) *(SBES 2019)*. Association for Computing Machinery, New York, NY, USA, 427–436. https://doi.org/10.1145/3350768.3350776

[17] Mengmeng Lu and Peng Liang. 2017. Automatic Classification of Non-Functional Requirements from Augmented App User Reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering* (Karlskrona, Sweden) *(EASE'17)*. Association for Computing Machinery, New York, NY, USA, 344–353. https://doi.org/10.1145/3084226.3084241

[18] Aleksandar Mitrevski. 2021. Software Requirements classification using Machine Learning algorithms on the PROMISE_exp data set by Lima et al. (2019). https://github.com/AleksandarMitrevski/se-requirements-classification

[19] Thuy Nguyen. 2021. Cross-applicability of ML classification methods intended for (non-)functional requirements. http://essay.utwente.nl/88236/

[20] Wisam A. Qader, Musa M. Ameen, and Bilal I. Ahmed. 2019. An Overview of Bag of Words;Importance, Implementation, Applications, and Challenges. In *2019 International Engineering Conference (IEC)*. 200–204. https://doi.org/10.1109/IEC47844.2019.8950616

[21] Md. Abdur Rahman, Md. Ariful Haque, Md. Nurul Ahad Tawhid, and Md. Saeed Siddik. 2019. Classifying Non-Functional Requirements Using RNN Variants for Quality Software Development *(MaLTeSQuE 2019)*. Association for Computing Machinery, New York, NY, USA, 25–30. https://doi.org/10.1145/3340482.3342745

[22] Zahra Shakeri, Oliver Karras, Parisa Ghazi, Martin Glinz, Guenther Ruhe, and Kurt Schneider. 2017. What Works Better? A Study of Classifying Requirements. https://doi.org/10.1109/RE.2017.36

[23] Grigori Sidorov. 2019. Vector space model for texts and the tf-idf measure. *SpringerBriefs in Computer Science* (2019), 11 – 15. https://doi.org/10.1007/978-3-030-14771-6_3

[24] John Slankas and Laurie Williams. 2013. Automated extraction of non-functional requirements in available documentation. In *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*. 9–16. https://doi.org/10.1109/NAturaLiSE.2013.6611715

[25] Marina Sokolova and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. *Information Processing & Management* 45 (07 2009), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002

[26] Alper Kursat Uysal and Serkan Gunal. 2014. The impact of preprocessing on text classification. *Information Processing & Management* 50, 1 (2014), 104–112. https://doi.org/10.1016/j.ipm.2013.08.006

[27] Yiming Yang and Jan O. Pedersen. 1997. A Comparative Study on Feature Selection in Text Categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML '97)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 412–420.