Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 2, April 2023
DOI : https://doi.org/10.33395/sinkron.v8i2.12415
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Class Balancing Methods Comparison for Software Requirements Classification on Support Vector Machines

Fachrul Pralienka Bani Muhamad[1)*], Esti Mulyani [2)], Munengsih Sari Bunga [3)], Achmad Farhan Mushafa [4)]
[1,2,3,4)] Politeknik Negeri Indramayu, Indonesia
[1)]fachrul.pbm@polindra.ac.id, [2)]estimulyani@polindra.ac.id, [3)]nengslim85@gmail.com,
[4)] achmadfarhanmushafa@gmail.com

Submitted : May 10, 2023 | **Accepted** : May 14, 2023 | **Published** : May 15, 2023

Abstract: Cost, time, and development effort can increase due to errors in analyzing functional and non-functional software requirements. To minimize these errors, previous research has tried to classify software requirements, especially non-functional requirements, on the PROMISE dataset using the Bag of Words (BoW) feature extraction and the Support Vector Machine (SVM) classification algorithm. On the other hand, the unbalanced distribution of class labels tends to decrease the evaluation result. Moreover, most software requirements are usually functional requirements. Therefore, there is a tendency for classifier models to classify test data as functional requirements. Previous research has performed class balancing on a dataset to handle unbalanced data. The study can achieve better classification evaluation results. Based on the previous research, this study proposes to combine the class balancing method and the SVM algorithm. K-fold cross-validation is used to optimize the training and test data to be more consistent in developing the SVM model. Tests were carried out on the value of K in k-fold, i.e., 5, 10, and 15. Results are measured by accuracy, f1-score, precision, and recall. The Public Requirements (PURE) dataset has been used in this research. Results show that SVM with class balancing can classify software requirements more accurately than SVM without class balancing. Random Over Sampling is the class balancing method with the highest evaluation score for classifying software requirements on SVM. The results showed an improvement in the average value of accuracy, f1 score, precision, and recall in SVM by 22.07%, 19.67%, 17.73%, and 19.67%, respectively.

Keywords: Class balancing; classification; random over sampling; software requirements; support vector machine

INTRODUCTION

Software development enables a comprehensive development process, beginning with the analysis of software requirements and ending with the selection of technology to use or develop in order to ensure the efficiency of the software produced. According to(Vogelsang & Borg, 2019)(Aminu Umar, 2020)(Yanmin Yang et al., 2020), software requirements analysis is an important concern because software development begins with requirements analysis; if an error occurs in the analysis, it can increase cost, time, and effort, and affect software function and quality. Software requirements consist of two categories, functional and non-functional. There are several categories of non-functional

*Corresponding Author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 2, April 2023
DOI : https://doi.org/10.33395/sinkron.v8i2.12415
e-ISSN : 2541-2019
p-ISSN : 2541-044X

requirements: Security, Usability, Reliability, Portability, Performance, Compatibility, and Maintainability (Mulyawan et al., 2021).

In contrast to the traditional approach, which relies on a detailed process and comprehensive planning, determining software requirements with an Agile approach can still be obtained in stages by a Product Owner (Aminu Umar, 2020). Still, given the relatively large and changeable Product Backlog, as well as reduced levels of concentration and human focus when doing repetitive work, more significant effort is needed to determine the needs of the software. Therefore, we need a model that can provide information about label descriptions of software requirements, both functional and non-functional requirements.

In this study, we investigated how to improve the quality of software requirements classification by analyzing the use of Bag of Words (BoW) vectorization techniques, class balancing techniques (Ao et al., n.d.), and the number of folds in cross-validation used by SVM. The main contributions to this research are a comparison of feature extraction techniques, a comparison of the use of several class balancing techniques, as well as the number of folds in cross-validation as measured by accuracy scores, f1-scores, precision, and recall (Canedo & Mendes, 2020). Based on these problems, we conducted research on the classification of software requirements as a tool for system analysts and developers to analyze software requirements.

LITERATURE REVIEW

Bag of Words (Bow)

One of the vectorization techniques used in this study is Bag of Words (BoW) (Shreda & Hanani, 2021)(Md. Ariful Haque et al., 2019). This vector space model represents unstructured text as a numerical vector, where it determines the presence of feature words from the words of an instance. The software requirements are converted into numerical vectors so that each document is represented by a vector (row).

SMOTE

The process of the SMOTE algorithm is to generate new data in the classes that are under-represented. The SMOTE algorithm generates new data by connecting new data to its nearest neighbors (Gazali Mahmud et al., 2023). However, this algorithm causes the data to overlap. See Figure 1 for an illustration of data distribution using SMOTE.

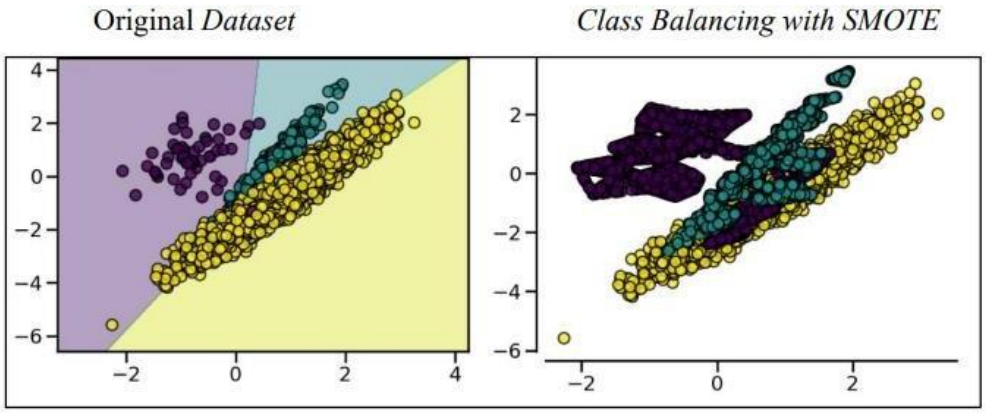

Fig. 1. The illustration of SMOTE class balancing

Borderline SMOTE

The Borderline SMOTE algorithm process generates new data in underrepresented classes. Borderline SMOTE generates new data by connecting new data to nearest neighbors based on line boundaries (Majzoub & Elgedawy, 2020). Line boundaries are defined for each data in order to reduce data overlapping. An illustration of data distribution using Borderline SMOTE is shown in Figure 2.

*Corresponding Author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 2, April 2023
DOI : https://doi.org/10.33395/sinkron.v8i2.12415
e-ISSN : 2541-2019
p-ISSN : 2541-044X

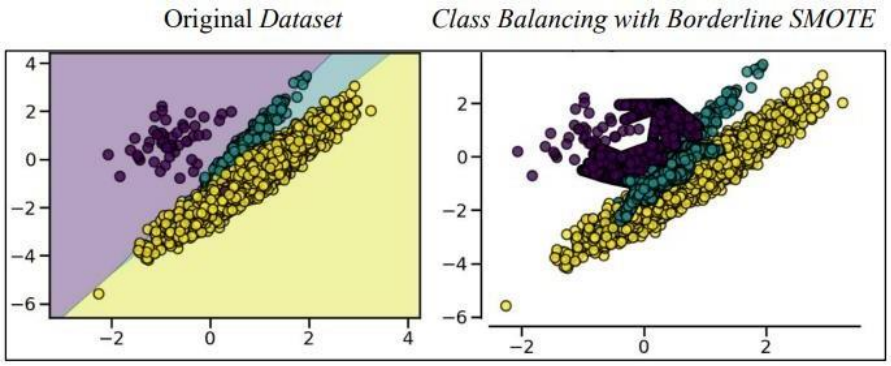

Fig. 2. The illustration of Borderline SMOTE class balancing

Random Over Sampling (ROS)

The process of the ROS algorithm is to generate new data in underrepresented classes. The result is the majority class does not take over the minority class, so all classes are represented during training. An illustration of class balancing using ROS can be seen in Figure 3.

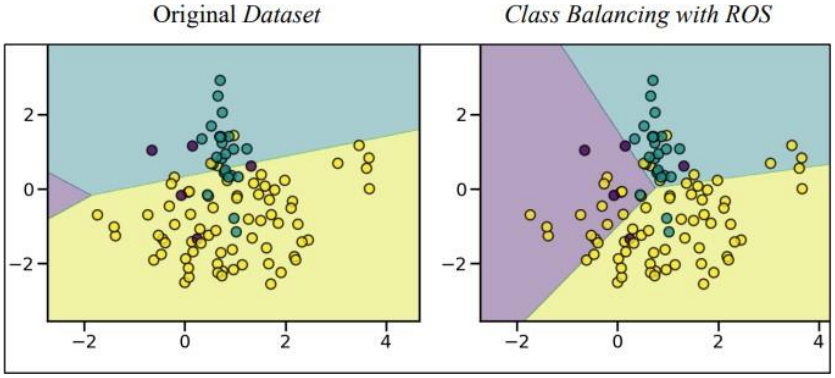

Fig. 3. The illustration of ROS class balancing

Random Under Sampling (RUS)

Unlike ROS, the RUS algorithm reduces the data with the majority class in the data set so that the amount of distribution of the majority data becomes equal to the amount of data with the minority class. Figure 4 shows the class distribution using RUS.

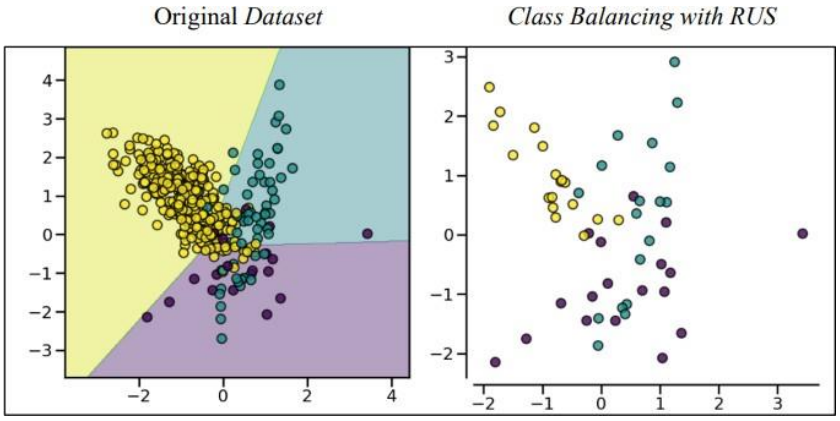

Fig. 4. The illustration of RUS class balancing

* Corresponding Author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 2, April 2023
DOI : https://doi.org/10.33395/sinkron.v8i2.12415
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Support Vector Machine

Support Vector Machines are a particularly powerful and flexible class of supervised algorithms for classifying and regressing (Canedo & Mendes, 2020). SVM can perform linear or nonlinear classification, regression, and even outlier detection, making it a powerful and versatile machine learning model. The algorithm performs the classification by creating a linear hyperplane of maximum margin separating two classes. Due to this margin, there is little chance of misclassifying new instances because there are few possibilities to separate the data from the sample.

K Fold Cross Validation

The actual samples are divided into k subsamples of equal size in the k-fold cross validation method. The process is repeated k times, with each sub-sample used as validation data for testing the classification model. The benefit of this method is that compared to repeated random subsamples, each sample is trained and validated at least once (Pralienka et al., 2017)(Srujan Raju et al., 2018)(Muhammad Asrol et al., 2020). Where k is the user-selected fixed parameter.

Performance Measure

A visualization tool commonly used in supervised learning is the confusion matrix. Each column in the matrix represents the prediction classes, while each row represents the events in the real classes. Actual and predicted information about the classification system is contained in the confusion matrix (Riansyah et al., 2023).

METHOD

This section describes the research methodology used. In general, these studies have been carried out through several major phases, i.e., data collection, data preprocessing, feature extraction, class balancing, and classification through training and test data distribution. An overview of the software requirements classification research steps is shown in Figure 5.

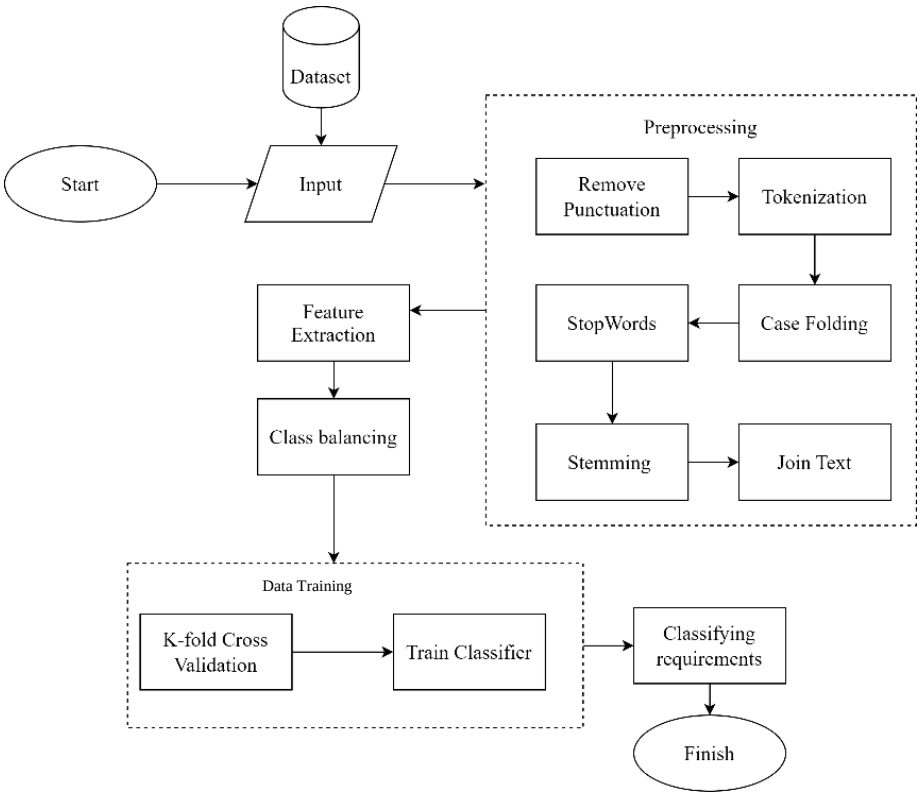

Fig. 5. The research phase of the proposed software requirements classification method

* Corresponding Author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 2, April 2023
DOI : https://doi.org/10.33395/sinkron.v8i2.12415
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Dataset

Data collection in this paper uses public datasets. The most common one is the PURE (Public Requirement) dataset, which has been in use by several previous researchers (Khayashi et al., 2022)(Shreda & Hanani, 2021)(Ferrari et al., 2017). The list of software requirements obtained from the PURE dataset was initially unlabeled, so at this stage experts applied labels to each of these requirement's statements. For example, Table 1 shows the software requirement statements that represent each software requirement category.

Table 1. Software requirements statement and its labeling

Statement	Category
The user interfaces should be designed to make them user intuitive.	Usability
To prevent interruptions in the data transfer, the communication system shall allow for redundant communication channels. Processing of data may be carried out simultaneously on more units. Automatic procedures for detection of communication faults and for managing redundancy of system components shall be established. The physical transport media should possibly be redundant to a certain degree depending on the conditions at the specific plant.	Reliability
The system shall encrypt all customer data in database.	Security
The system shall be able to handle 1000 customers logged in concurrently at the same time.	Performance
Maintainability is a primary goal for this project. For example, using appropriate subscenes in the main Flash game to split up the code will allow for easy alteration at a later date.	Maintainability
Web application should be available to run on browsers like IE, Firefox, Chrome, Opera or Safari.	Portability
Citizens can register their complaints with police and then based on the evidence, facts and following investigation, police shall take the complaint forward. The Registration module acts as an interface between the police and citizens, and it eases the approach, interaction and information exchange between police and complainants.	Functional

This dataset contains seven class labels for software requirements: usability, reliability, security, performance, maintainability, portability, and functional requirements. The record consists of 885 requirement statements, half of which are labeled functional requirements. Referring to previous research, the unbalance distribution of the data can affect the non-optimality of a classifier in providing classification results (Tiun et al., 2020). Table 2 is a comparison of the number and percentage of manually labeled software requirements in the dataset.

Table 2. Software requirements label comparison

Category	Total Statements	Percentage
Usability	102	12%
Reliability	53	6%
Security	106	11%
Performance	109	12%
Maintainability	35	4%
Portability	41	5%
Functional Requirement	439	50%
Total	**885**	**100%**

* Corresponding Author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 2, April 2023
DOI : https://doi.org/10.33395/sinkron.v8i2.12415
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Preprocessing

Once the requirements statement has been labeled by experts, the next step is preprocessing. This process includes punctuation removal, tokenization, case folding, stop words, stemming, and text merging or joining text. Punctuation removal is a process to clean sentences from punctuation marks or perform replacements on the target string based on the specified pattern. The Remove Punctuation process is used to remove punctuation marks and numbers (Hickman et al., 2022).

After removing the punctuation and numbers, this step changes the font to lowercase or standard letters. This ensures that the sentence or word being edited has the same appearance or uniform character (Rahimi et al., 2020). The next step after case folding is tokenization. At this stage, text from sentences, paragraphs, or documents is broken down into specific smaller pieces. These chunks are known as tokens. The separation of the text is based on the amount of space in the text (Hickman et al., 2022).

The auxiliary verbs, prepositions, pronouns, adverbs, and conjunctions of the following tokenization results are removed at this stage. For example, the, be, to, in, is, and others. This is done so that the data to be processed is only data that has meaning or essential information (Hickman et al., 2022). The output of the stop word results is then mapped and decomposed into its basic word forms or root words or stems (Binkhonain & Zhao, 2019). For example, the words "creative", "creating", "created", and "creates" are mapped to "create. The previously split words into tokens were then combined into a sentence. This is done to group words based on the initial sentence.

Feature Extraction

This step converts the pre-processed requirements document into a format that the machine learning model can understand (Ramos et al., 2018). In this step, the document is represented as a vector, where the value of this word is weighted by different techniques, such as binary methods (Dharma & Saragih, 2022). One of the vectorization techniques used in this study is Bag of Words (BoW) (Shreda & Hanani, 2021)(Md. Ariful Haque et al., 2019). This vector space model represents unstructured text as a numerical vector, where it determines the presence of feature words from the words of an instance. The software requirements are converted into numerical vectors so that each document is represented by a vector (row).

Class Balancing

This stage starts with the input of the data, which is performed by feature extraction. Class balancing is performed to balance the amount of distribution of each class or category so that the classification results can be more optimal. Several techniques for performing class balancing include SMOTE, Borderline SMOTE, Random Over Sampling (ROS), and Random Under Sampling (RUS) (Susan & Kumar, 2021)(Chakraborty et al., 2021). Each method of class balancing has different characteristics. An overview of the data distribution performed by class balancing is shown in Table 3.

Table 3. Data distribution comparison

Category	Raw	SMOTE	Borderline SMOTE	ROS	RUS
Usability	102	439	439	439	35
Reliability	53	439	439	439	35
Security	106	439	439	439	35
Performance	109	439	439	439	35
Maintainability	35	439	439	439	35
Portability	41	439	439	439	35
Functional Requirement	439	439	439	439	35

* Corresponding Author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 2, April 2023
DOI : https://doi.org/10.33395/sinkron.v8i2.12415
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Classification

After class balancing on the dataset, the next step is to divide the data equally into several training and testing sets. The data partitioning is done using the K-fold cross validation technique, so that each data can be used as test data and training data in different folds. The Support Vector Machine (SVM) method is also used to classify software requirements at this stage. The parameters of accuracy, F1-score, precision, and recall from the previous research are used to evaluate the results (Shreda & Hanani, 2021).

RESULT

This section describes the test scenario of this study and the results obtained. The test scenario is performed after the feature extraction stage is completed. It consists of class balancing, cross validation, classification, and evaluation. The illustration of the test scenario is shown in Figure 6.

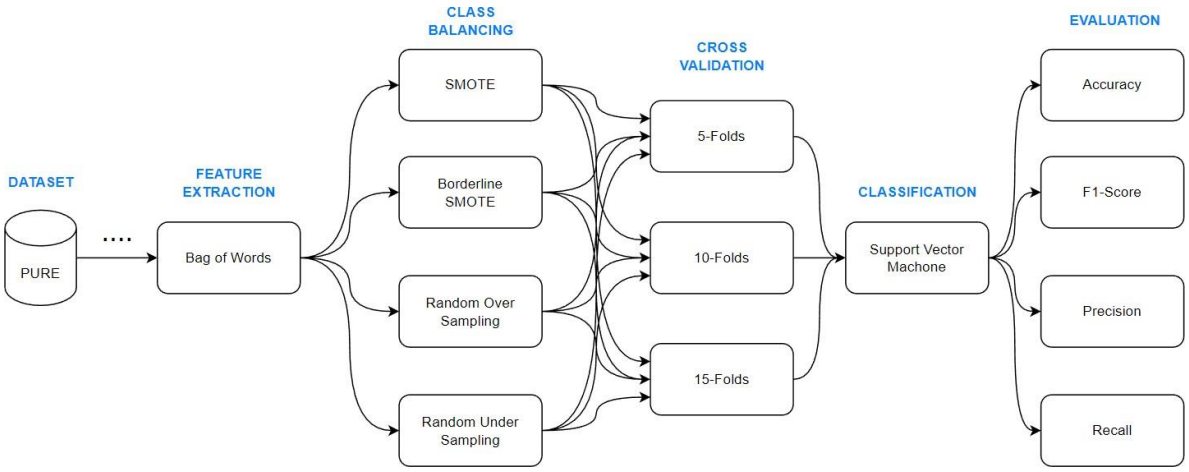

Fig. 6. Test scenario

Preprocessing Implementation

Punctuation removal is the first process performed in the preprocessing phase. Figure 7 shows that all punctuation and digits have been removed. The next step is tokenization and case folding. The process is performed sequentially, which is used as a single step. The results of this process are shown in Figure 8. Meanwhile, Figure 9 shows the results of stopwords, which removes auxiliary verbs, prepositions, pronouns, adverbs, and conjunctions. The last preprocessing step is stemming. At this stage, all words are converted into root words. The results of stemming are shown in Figure 10.

```
# remove punctuation
def remove_punctuation(text):
    return re.sub('[^a-zA-Z]',' ', str(text))
data_statement_puct_remove = data_statement.apply(lambd

print(data_statement_puct_remove)
✓ 0.1s

0      The user interfaces should be designed to make...
1      The user interfaces of the system should compl...
2      ICT accessibility  ISO         shall be the st...
3      Providing text equivalents for non text media ...
4      Making navigation self descriptive  Navigation...
                             ...
880    This is the feature of Libra that decides how ...
881    This feature decides on which node and on whic...
882    This feature takes the job and inserts it into...
883    Every time a job is scheduled for execution on...
884    As long as there are jobs in the queues  this ...
Name: Statement, Length: 885, dtype: object
```

Fig. 7. Punctuation removal result

```
from nltk.tokenize import word_tokenize
# tokenize
def tokenize(text):
    tokens = word_tokenize(text)
    return tokens
data_statement_token = data_statement_puct_remove.apply
print(data_statement_token)
✓ 0.2s

0      [the, user, interfaces, should, be, designed, ...
1      [the, user, interfaces, of, the, system, shoul...
2      [ict, accessibility, iso, shall, be, the, stan...
3      [providing, text, equivalents, for, non, text,...
4      [making, navigation, self, descriptive, naviga...
                             ...
880    [this, is, the, feature, of, libra, that, deci...
881    [this, feature, decides, on, which, node, and,...
882    [this, feature, takes, the, job, and, inserts,...
883    [every, time, a, job, is, scheduled, for, exec...
884    [as, long, as, there, are, jobs, in, the, queu...
Name: Statement, Length: 885, dtype: object
```

Fig. 8. Tokenizing dan case folding results

* Corresponding Author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 2, April 2023
DOI : https://doi.org/10.33395/sinkron.v8i2.12415
e-ISSN : 2541-2019
p-ISSN : 2541-044X

```
# remove stopwords
stopwords = nltk.corpus.stopwords.words('english')
def remove_stopwords(text):
    tokens = [token for token in text if token not in st
    return tokens
data_statement_token_stop = data_statement_token.apply(
print(data_statement_token_stop)
```

✓ 0.1s

```
0      [user, interfaces, designed, make, user, intui...
1      [user, interfaces, system, comply, standard, iso]
2      [ict, accessibility, iso, shall, standard, gui...
3      [providing, text, equivalents, non, text, medi...
4      [making, navigation, self, descriptive, naviga...
                             ...
880    [feature, libra, decides, job, scheduled, base...
881    [feature, decides, node, queue, job, placed, e...
882    [feature, takes, job, inserts, queue, executio...
883    [every, time, job, scheduled, execution, host,...
884    [long, jobs, queues, feature, run, essentially...
Name: Statement, Length: 885, dtype: object
```

Fig. 9. Stopwords result

```
# stemming
stemmer = nltk.stem.PorterStemmer()
def stem_tokens(tokens):
    text = [stemmer.stem (word) for word in tokens]
    return text
data_statement_token_stop_stem = data_statement_token_stop.apply(
data_statement_token_stop_stem
```

✓ 0.4s

```
0          [user, interfac, design, make, user, intuit]
1        [user, interfac, system, compli, standard, iso]
2      [ict, access, iso, shall, standard, guidanc, i...
3      [provid, text, equival, non, text, media, obje...
4      [make, navig, self, descript, navig, design, h...
                             ...
880    [featur, libra, decid, job, schedul, base, bud...
881    [featur, decid, node, queue, job, place, execu...
882    [featur, take, job, insert, queue, execut, hos...
883    [everi, time, job, schedul, execut, host, feat...
884    [long, job, queue, featur, run, essenti, decid...
Name: Statement, Length: 885, dtype: object
```

Fig. 10. Stemming result

Feature Extraction Implementation

The bag of words method is used to extract features. This approach gives word weighting by providing three indicators: the word's description from the sentence's order (0), the word's index (444), and the number of times the word appears in the dataset (1). The results of applying the bag of words method are shown in Figure 11.

```
#Feature Extraction
vectorizer = CountVectorizer().fit(data_clean)
data_vtr = vectorizer.transform(data_clean)
label = categories.values.tolist()
print(data_vtr)
```

```
Output exceeds the size limit. Open the full outpu
  (0, 444)      1
  (0, 859)      1
  (0, 869)      1
  (0, 981)      1
  (0, 1786)     2
  (1, 302)      1
  (1, 859)      1
  (1, 877)      1
  (1, 1584)     1
  (1, 1646)     1
  (1, 1786)     1
  (2, 11)       6
  (2, 97)       2
  (2, 143)      1
  (2, 341)      2
  (2, 736)      3
  (2, 772)      1
  (2, 790)      2
```

Fig. 11. Bag of words (Bow) methods implementation

Class Balancing Implementation

This section describes the results of implementing the class balancing method: SMOTE, borderline SMOTE, ROS, and RUS. Sequentially, the class balancing implementation recapitulation can be seen in Table 3. The class balancing methods SMOTE, Borderline SMOTE, and ROS produced 3073 rows of data. Meanwhile, RUS has only 245 rows of data. Next, the results of each class balancing method are combined with the Support Vector Machine (SVM) classification method.

DISCUSSIONS

Each result of the class balancing method consisting of SMOTE, Borderline SMOTE, ROS and RUS are used as input for classification. Experiments were conducted using the K-Fold cross validation method, 5-fold, 10-fold, and 15-fold, respectively. Each experimental result is evaluated by calculating the average accuracy, F1 score, precision, and recall. After the evaluation, the next step is to perform a classification experiment on the five labeled requirement statements (Table 4). The five requirement statements are made manually and do not refer to the PURE or other data sets.

* Corresponding Author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 2, April 2023
DOI : https://doi.org/10.33395/sinkron.v8i2.12415
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Table 4 Experimental requirements for performing classification

No	Requirements	Category
1	The system should give a notification after finishing the calculation.	Functional
2	The software should prevent another user to access another user data.	Security
3	The system should provide any additional instruction in every step of a transaction.	Usability
4	The system shall be ready for receiving any request from user 24 hours a day.	Reliability
5	The software should have facilities to balance the load, so that it can be available 24 hours a day.	Performance

5-Fold

Based on the 5-fold test results, it was found that the ROS-SVM combination produced the highest average accuracy, F1 score, precision, and recall among the other method combinations (Table 5). The results of the comparison of the scores are presented in a graph, which can be seen in Figure 12.

Table 5 Comparison of 5-fold SVM average scores

Class Balancing Method	SVM Average Score **(5-Fold)**			
	Accuracy	F1-Score	Precision	Recall
SMOTE	0.739	0.745	0.774	0.739
Borderline SMOTE	0.750	0.755	0.778	0.750
ROS	**0.986**	**0.985**	**0.986**	**0.986**
RUS	0.514	0.526	0.607	0.514
None (SVM without class balancing)	0.719	0.790	0.806	0.791

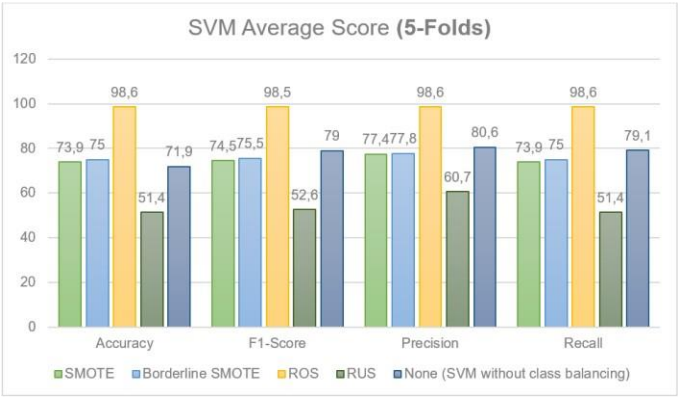

Fig. 12. Graph of the comparison results of the average SVM score (5-Fold)

After the best evaluation value is known, a classification test is performed using the five statements in Table 4. Based on the test results, it is found that the SMOTE-SVM, ROS-SVM, and SVM classification models without class balancing can correctly determine two categories of software requirements according to the classification label (see Table 6). Meanwhile, other classification models can only correctly determine one category at most, i.e., borderline SMOTE-SVM and RUS-SVM.

Table 6. Evaluated classification model experiments (5-fold)

Req No	Requirements Category	SMOTE – SVM	Borderline SMOTE – SVM	ROS – SVM	RUS – SVM	SVM without class balancing
1	**Functional**	Reliability	Reliability	Usability	Portability	Usability
2	**Security**	**Security**	**Security**	**Security**	**Security**	**Security**
3	**Usability**	Reliability	Reliability	**Usability**	Portability	**Usability**
4	**Reliability**	Security	Security	Functional	Functional	Functional
5	**Performance**	**Performance**	Portability	Functional	Portability	Functional
TOTAL		**2**	1	**2**	1	**2**

* Corresponding Author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 2, April 2023
DOI : https://doi.org/10.33395/sinkron.v8i2.12415
e-ISSN : 2541-2019
p-ISSN : 2541-044X

10-Fold

Similar to the 5-fold experiment, in the 10-fold scenario it was concluded that the ROS-SVM combination produced the highest average accuracy, F1 score, precision and recall values (Table 7). However, compared to 5-fold, the combination of ROS-SVM with 10-fold produces a higher evaluation value. The comparison of the results of the 10-fold evaluation, presented as a graph, can be seen in Figure 13.

Table 7. Comparison of 10-fold SVM average scores

Class Balancing Method	SVM Average Score **(10-Fold)**			
	Accuracy	F1-Score	Precision	Recall
SMOTE	0.744	0.749	0.778	0.774
Borderline SMOTE	0.760	0.766	0.791	0.760
ROS	**0.987**	**0.987**	**0.987**	**0.987**
RUS	0.527	0.525	0.635	0.527
None (SVM without class balancing)	0.787	0.787	0.809	0.787

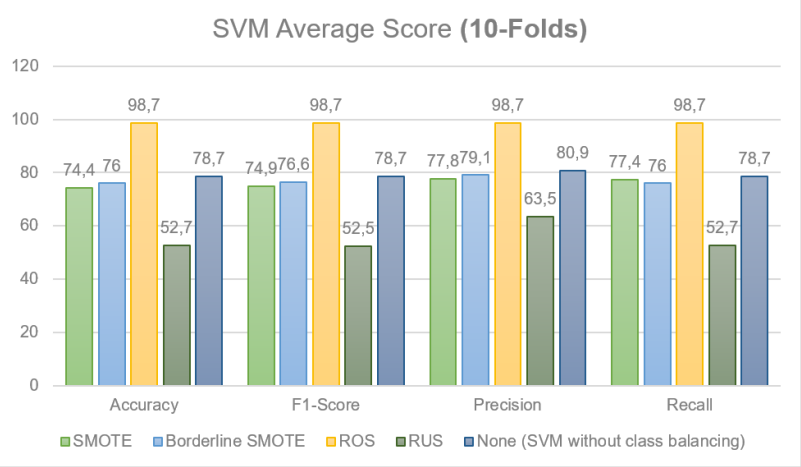

Fig. 13. Graph of the comparison results of the average SVM score (10-Fold)

At this point, a classification test was performed using a model evaluated using the five sentences in Table 4. Based on the test results, only Borderline SMOTE-SVM combination correctly classified one software requirement category. In addition, each combination of methods correctly predicted two categories of software requirements according to their classification labels (Table 8).

Table 8. Evaluated classification model experiments (10-fold)

Req No	Requirements Category	SMOTE – SVM	Borderline SMOTE – SVM	ROS – SVM	RUS – SVM	SVM without class balancing
1	**Functional**	Reliability	Reliability	Usability	Portability	Usability
2	**Security**	**Security**	**Security**	**Security**	**Security**	**Security**
3	**Usability**	Reliability	Reliability	**Usability**	Portability	**Usability**
4	**Reliability**	Security	Security	Functional	**Security**	Functional
5	**Performance**	**Performance**	Portability	Functional	Portability	Functional
TOTAL		**2**	1	**2**	1	**2**

* Corresponding Author

Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 2, April 2023
DOI : https://doi.org/10.33395/sinkron.v8i2.12415
e-ISSN : 2541-2019
p-ISSN : 2541-044X

15-Fold

Similarly, to the 5-fold and 10-fold experiments, in the 15-fold scenario, it was concluded that the ROS - SVM combination produced the highest average accuracy, F1 score, precision and recall values (Table 9). There is an increase in value of about 0.001 in each evaluation indicator. The comparative graph of the 15-fold evaluation results can be seen in Figure 14.

Table 9. Comparison of 15-fold SVM average scores

Class Balancing Method	SVM Average Score **(15-Fold)**			
	Accuracy	F1-Score	Precision	Recall
SMOTE	0.749	0.755	0.785	0.749
Borderline SMOTE	0.760	0.766	0.792	0.760
ROS	**0.988**	**0.987**	**0.988**	**0.988**
RUS	0.547	0.550	0.660	0.547
None (SVM without class balancing)	0.793	0.792	0.814	0.793

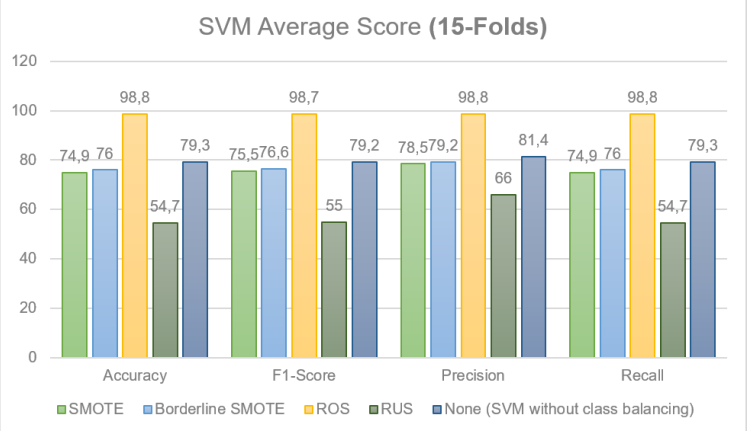

Fig. 14. Graph of the comparison results of the average SVM score (15-Fold)

In the case of 15-fold, the testing of the classification model with the software requirement data in Table 4 gives no different results than in the case of 10-fold. The details of the test results of the 15-fold classification model are shown in Table 10.

Table 10. Evaluated classification model experiments (15-fold)

Req No	Requirements Category	SMOTE – SVM	Borderline SMOTE – SVM	ROS – SVM	RUS – SVM	SVM without class balancing
1	**Functional**	Reliability	Reliability	Usability	Portability	Usability
2	**Security**	**Security**	**Security**	**Security**	**Security**	**Security**
3	**Usability**	Reliability	Reliability	**Usability**	Portability	**Usability**
4	**Reliability**	Security	Security	Functional	**Security**	Functional
5	**Performance**	**Performance**	Portability	Functional	Portability	Functional
TOTAL		**2**	1	**2**	1	**2**

Result Experiment Comparison

This section compares the classification results of combining the best class balancing method (ROS) with SVM and SVM without class balancing. Referring to Figure 16-18, it was found that in the 5-fold experiment, there was an increase in accuracy of 26.7%, f1 score of 19.5%, precision of 17.73% and recall of 19.5%. Meanwhile, in the 10-fold experiment, accuracy, F1 score, precision and recall increased by 20%, 20%, 17.8% and 20%, respectively. For the 15-fold test, the increase was no greater

* Corresponding Author

e-ISSN : 2541-2019
p-ISSN : 2541-044X

than 5-fold or 10-fold, namely 19.5%, 19.5%, 17.4%, and 19.5%, respectively. Table 11 shows the details of the improvement of SVM results with ROS class balancing.

Table 11. Details of SVM score improvement using ROS

K-Fold	Accuracy Improvement	F1 Score Improvement	Precision Improvement	Recall Improvement
5-Fold	26.7%	19.5%	18%	19.5%
10-Fold	20%	20%	17.8%	20%
15-Fold	19.5%	19.5%	17.4%	19.5%
Average	**22.07%**	**19.67%**	**17.73%**	**19.67%**

CONCLUSION

This study aims to analyze the comparative effect of using several combinations of class balancing methods with classifier models, especially SVM, in classifying software requirements. Based on several test results, SVM has generally classified software requirements better than the combination of SMOTE, Borderline SMOTE, and RUS class balancing methods. However, when SVM is combined with ROS, the accuracy, F1 score, precision, and recall values increase significantly, i.e., 22.07%, 19.67%, 17.73%, and 19.67%, respectively. However, based on the results of the classification experiment for the five new requirements, the best classification model (ROS-SVM) still shows no difference with SVM. The following research can be done on the Indonesian language software requirements dataset.

REFERENCES

Aminu Umar, M. (2020). Automated Requirements Engineering Framework for Agile Development. *ICSEA 2020: The Fifteenth International Conference on Software Engineering Advances*, *c*, 147–150.

Ao, S. I., Gelman, Len., Hukins, D. W. L., Hunter, Andrew., Korsunsky, Alexander., & International Association of Engineers. (n.d.). *Balancing Class for Performance of Classification with a Clinical Dataset*. 1538.

Binkhonain, M., & Zhao, L. (2019). A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert Systems with Applications: X*, *1*. https://doi.org/10.1016/j.eswax.2019.100001

Canedo, E. D., & Mendes, B. C. (2020). Software Requirements Classification Using Machine Learning Algorithms. *Entropy*, *22*(9), 1–20. https://doi.org/10.3390/E22091057

Chakraborty, J., Majumder, S., & Menzies, T. (2021). Bias in machine learning software: Why? how? what to do? *ESEC/FSE 2021 - Proceedings of the 29th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 429–440. https://doi.org/10.1145/3468264.3468537

Dharma, A. S., & Saragih, Y. G. R. (2022). Comparison of Feature Extraction Methods on Sentiment Analysis in Hotel Reviews. *Sinkron: Jurnal Dan Penelitian Teknik Informatika*, *7*(4), 2349–2354. https://doi.org/10.33395/sinkron.v7i4.11706

Ferrari, A., Spagnolo, G. O., & Gnesi, S. (2017). PURE: A Dataset of Public Requirements Documents. *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference, RE 2017*, 502–505. https://doi.org/10.1109/RE.2017.29

Gazali Mahmud, F., Iman Hermanto, T., Maruf Nugroho, I., & Tinggi Teknologi Wastukancana, S. (2023). IMPLEMENTATION OF K-NEAREST NEIGHBOR ALGORITHM WITH SMOTE FOR HOTEL REVIEWS SENTIMENT ANALYSIS. *Sinkron: Jurnal Dan Penelitian Teknik Informatika*, *8*(2). https://doi.org/10.33395/10.33395/sinkron.v8i2.12214

Hickman, L., Thapa, S., Tay, L., Cao, M., & Srinivasan, P. (2022). Text Preprocessing for Text Mining in Organizational Research: Review and Recommendations. *Organizational Research Methods*, *25*(1), 114–146. https://doi.org/10.1177/1094428120971683

* Corresponding Author

Khayashi, F., Jamasb, B., Akbari, R., & Shamsinejadbabaki, P. (2022). Deep Learning Methods for Software Requirement Classification: A Performance Study on the PURE dataset. *ArXiv Preprint ArXiv:2211.05286*.

Majzoub, H. Al, & Elgedawy, I. (2020). AB-SMOTE: An Affinitive Borderline SMOTE Approach for Imbalanced Data Binary Classification. *International Journal of Machine Learning and Computing*, *10*(1), 31–37. https://doi.org/10.18178/ijmlc.2020.10.1.894

Md. Ariful Haque, Md. Abdur Rahman, & Md Saeed Siddik. (2019). Non-Functional Requirements Classification withFeature Extraction and Machine Learning: AnEmpirical Study. *1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*.

Muhammad Asrol, Petir Papilo, & Fergyanto E Gunawan. (2020). Support Vector Machine with K-fold Validation to Improve the Industry's Sustainability Performance Classification. *5th International Conference on Computer Science and Computational Intelligence*, 854–861.

Mulyawan, M. D., Kumara, I. N. S., Bagus, I., Swamardika, A., & Saputra, K. O. (2021). *Kualitas Sistem Informasi Berdasarkan ISO / IEC 25010 : 20*(1).

Pralienka, F., Muhamad, B., Perbaikan, :, Kesalahan, P., Siahaan, D. O., & Fatichah, C. (2017). Perbaikan Prediksi Kesalahan Perangkat Lunak Menggunakan Seleksi Fitur dan Cluster-Based Classification. In *JNTETI* (Vol. 6, Issue 3).

Rahimi, N., Eassa, F., & Elrefaei, L. (2020). *SS symmetry An Ensemble Machine Learning Technique for*. *Ml*, 1–25.

Ramos, F., Costa, A., Perkusich, M., Almeida, H., & Perkusich, A. (2018). A non-functional requirements recommendation system for scrum-based projects. *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, *2018-July*(July), 149–154. https://doi.org/10.18293/SEKE2018-107

Riansyah, M., Suwilo, S., & Zarlis, M. (2023). Improved Accuracy In Data Mining Decision Tree Classification Using Adaptive Boosting (Adaboost). *SinkrOn*, *8*(2), 617–622. https://doi.org/10.33395/sinkron.v8i2.12055

Shreda, Q. A., & Hanani, A. A. (2021). Identifying Non-functional Requirements from Unconstrained Documents using Natural Language Processing and Machine Learning Approaches. *IEEE Access*, *4*, 1–22. https://doi.org/10.1109/ACCESS.2021.3052921

Srujan Raju, K., Murty, M. R., Varaprasad Rao, M., & Satapathy, S. C. (2018). Support Vector Machine with K-fold Cross Validation Model for Software Fault Prediction. *International Journal of Pure and Applied Mathematics*, *118*(20), 321–334. http://www.ijpam.eu

Susan, S., & Kumar, A. (2021). The balancing trick: Optimized sampling of imbalanced datasets—A brief survey of the recent State of the Art. *Engineering Reports*, *3*(4). https://doi.org/10.1002/eng2.12298

Tiun, S., Mokhtar, U. A., Bakar, S. H., & Saad, S. (2020). Classification of functional and non-functional requirement in software requirement using Word2vec and fast Text. *Journal of Physics: Conference Series*, *1529*(4). https://doi.org/10.1088/1742-6596/1529/4/042077

Vogelsang, A., & Borg, M. (2019). Requirements Engineering for Machine Learning: Perspectives from Data Scientists. *IEEE 27th International Requirements Engineering Conference Workshops (REW)*, 245–251. http://arxiv.org/abs/1908.04674

Yanmin Yang, Xin Xia, David Lo, & John Grundy. (2020). A Survey on Deep Learning for Software Engineering. *ACM Computing Survey*, *1*(1), 1–35. https://doi.org/10.1145/1122445.1122456

* Corresponding Author