ISSN 2355-0082

A Comparison of Traditional Machine Learning Approaches for Supervised Feedback Classification in Bahasa Indonesia

Andre Rusli[1], Alethea Suryadibrata[2], Samiaji Bintang Nusantara[3], Julio Christian Young[4]
[1, 2, 4] Department of Informatics, Universitas Multimedia Nusantara, Tangerang, Indonesia
andre.rusli@umn.ac.id
alethea@umn.ac.id
julio.christian@umn.ac.id
[3] Department of Journalism, Universitas Multimedia Nusantara, Tangerang, Indonesia
samiaji.bintang@umn.ac.id

Accepted on February 24, 2020
Approved on June 15, 2020

***Abstract*—The advancement of machine learning and natural language processing techniques hold essential opportunities to improve the existing software engineering activities, including the requirements engineering activity. Instead of manually reading all submitted user feedback to understand the evolving requirements of their product, developers could use the help of an automatic text classification program to reduce the required effort. Many supervised machine learning approaches have already been used in many fields of text classification and show promising results in terms of performance. This paper aims to implement NLP techniques for the basic text preprocessing, which then are followed by traditional (non-deep learning) machine learning classification algorithms, which are the Logistics Regression, Decision Tree, Multinomial Naïve Bayes, K-Nearest Neighbors, Linear SVC, and Random Forest classifier. Finally, the performance of each algorithm to classify the feedback in our dataset into several categories is evaluated using three F1 Score metrics, the macro-, micro-, and weighted-average F1 Score. Results show that generally, Logistics Regression is the most suitable classifier in most cases, followed by Linear SVC. However, the performance gap is not large, and with different configurations and requirements, other classifiers could perform equally or even better.**

***Index Terms*—Bahasa Indonesia, F1 Score, Feedback Classification, Requirements Engineering, Supervised Machine Learning**

I. INTRODUCTION

Feedback from users is an essential source of information for software engineers, especially requirements engineers. As time goes, software and their requirements also evolve. To get a grip on which direction a software product must be going to, a lot of developers rely on user feedback. There are several reasons which could cause software requirements to change or evolve, including defects to be fixed, project fluctuations in terms of priorities and constraints, better customer understanding of the system's actual features, and so on [1]. Furthermore, merely knowing users' opinions and sentiments regarding a specific feature in a software product could be very useful for developers and business owners. User involvement in software requirements engineering is crucial in delivering the right product, even after the product is released [2]. However, the amount of information the users provide can quickly become too abundant to be analyzed manually [3], causing various scalability problems. Especially for software products with lots of users, it becomes more challenging to put in the effort required to assess all user feedback; thus, the idea of building an automatic tool for processing user review seems promising.

Interests in classifying user feedback into several categories have grown in recent years. Several related works of research [4, 5, 6] use traditional machine learning classification approaches, such as Naïve Bayes classifier and Decision Tree classifier (C4.5 or J48), Support Vector Machines, and Logistics Regression, combined with NLP methods, and the result displays the enormous potential of those methods in classifying feedback written in English. Our research focuses more on supporting requirements engineers in processing Bahasa Indonesia, which has different structures and poses various challenges compared to English. Previous publications [7, 8] have already shown the possibility of doing text classification in Bahasa Indonesia for news articles and also to classify user sentiments in product review written in Bahasa Indonesia. Our previous work [9, 10] had tried to implement Naïve Bayes classifier to process user feedback to classify them into several categories, as well as categorizing them into positive and negative sentiments, showing promising results but could still be improved. Furthermore, the classification results and performance of the previous works could be re-evaluated by performing similar

NLP preprocessing techniques on the same dataset using different classification algorithms.

This paper aims to implement natural language processing techniques for text preprocessing, which then are followed by traditional (non-deep learning) machine learning classification algorithms and finally evaluate the performance of each algorithm in the same dataset that we use. Traditional machine learning approaches are chosen as there are some constraints for deep learning approaches which are, amongst other things [11], data-hungry and thus is not suitable for data with small size and has a high computational cost of learning, both of which is not readily available in every situation, including ours. Based on their performance in previous studies, we experimented with Logistics Regression, Decision Tree, Multinomial Naïve Bayes, K-Nearest Neighbors, Linear SVC, and Random Forest classifiers using the Scikit-Learn library [12] in Python. The performance of the various algorithms to classify the feedback in our dataset into several categories is then evaluated by looking into the confusion matrix results.

II. Resesarch Methods

A. Dataset

Our research uses a dataset of user feedback from a university e-learning web application used by staff, students, and faculty officers of the institution. At the moment of our experiment, the dataset consists of 345 user feedback, labelled manually by the head of the Learning Center Department that is in charge of the system. The texts are classified into several categories, which are Content, Technical, Strategic, and Others. Here are examples for each group in the dataset:

- "*Tolong dibuat fitur notifikasi untuk deadline terdekat*" = Technical (in English: "please build a notification feature for the nearest deadline")
- "*Dosen harus bisa diajak untuk mengupload materi di elearning*" = Strategic (in English: "Lecturer must be encouraged to upload course materials to the e-learning")
- "*Kalau bisa di perbanyak lagi materi - materi atau kisi -kisi*" = Content (in English: "If possible, add more materials and course summaries")
- "*semoga e-learning semakin baik dan menarik :)*" = Other (in English: "I hope for e-learning to be better and more interesting")

Fig. 1 shows the distribution of feedback in the imbalanced dataset that is used in our current experiment. As can be seen, the number of feedbacks in the technical category far outweighs the number of feedbacks in the content category. This kind of dataset possesses various challenges that could be addressed in multiple ways. However, this paper focus on evaluating the classifiers with minimum text preprocessing techniques. The next sub-sections describe the step by step explanation of our current experiments.

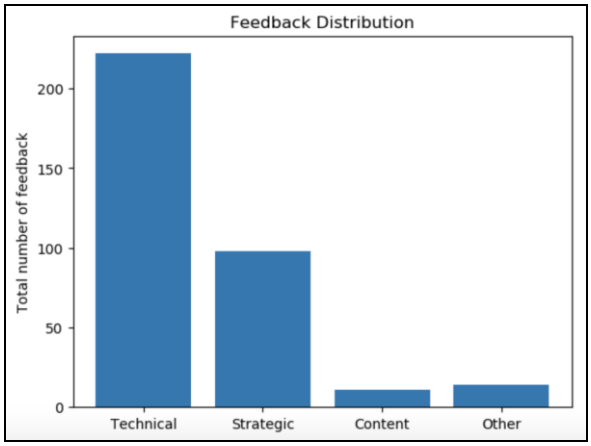

Fig. 1. Feedback Distribution in the Dataset

B. Text Preprocessing

In order to prepare our feedback to be classified using various traditional machine learning techniques, the feedback texts are preprocessed first. In this preprocessing phase, here are the steps that were done in our experiment:

1. Remove any character except alphabets using Regular Expression,
2. Lowercase all characters,
3. Stemming to remove suffixes, and
4. Stop-words removal.

After the text preprocessing phase is done, feedback texts became simpler to be passed on to the next steps. For example, feedback such as "Tolong dibuat fitur notifikasi untuk deadline terdekat" is processed to "fitur notifikasi deadline dekat".

C. Bag of Words Model

After every word in the dataset is preprocessed, we move on to the feature extraction phase. Our research uses the bag of words model using the CountVectorizer library from Scikit-Learn to convert a collection of text documents to a matrix of token counts [12]. In this phase, we experimented with various parameters in CountVectorizer regarding the word and character n-grams as the tokens converted from the text documents of our feedback dataset. Some of the settings that we experimented with are word unigram, word bigram, word trigram, word uni+bi+trigram, character unigram, character bigram, character 5-gram, and character 1- until 9-gram. Not all results will be described in this paper as many results are similar to each other; thus, only some of the findings that are considered to provide enough comparison between the various classification techniques are displayed.

D. *Feedback Classification*

Finally, after every feedback is preprocessed and fitted into the bag of words model, the dataset is split into training and testing sets with a 70:30 ratio. Furthermore, we experimented with six classification methods, which are Logistics Regression (LR), Decision Tree (DT), Multinomial Naïve Bayes (MNB), K-Nearest Neighbors (KNN), Linear Support Vector Classification (LSVC), Random Forest (RF). For the K-Nearest Neighbors, we experimented with k={1,3,5,7,9}, however only show the results of n=7 as it is the value with the overall best performance so far in the dataset and for the Linear SVC, we experimented with tolerance for stopping criteria (tol) equals to 1e-5.

III. Results and Discussion

The result of our experiments is generally divided into two groups, the first group uses the word n-gram, and the results are shown in Fig. 2 – Fig. 4, while the second group uses the character n-gram, and the results are shown in Fig. 5 – Fig. 7. Furthermore, the resulting figures depict the bar charts showing the comparison of the six classifiers in predicting the class of feedback from the testing set with different evaluation metrics, which are the macro-F1, micro F-1, and the weighted F-1 average score. We use different f1 score average metrics, using functions from the Scikit-learn library because each is used in different situations.

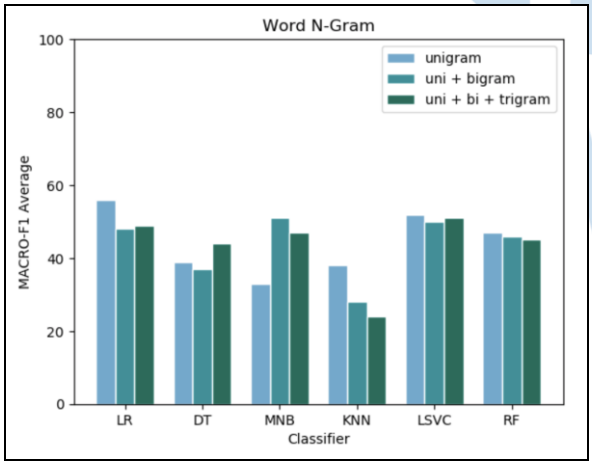

Fig. 2. Macro-average F1 Score with Word N-Gram

Macro-average F1 score calculates metrics for each label, and find their unweighted mean. This does not take label imbalance into account. The second metric, the micro-average F1 score, calculates metrics globally by counting the total true positives, false negatives, and false positives. And finally, weighted-average F1 score calculates metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F-score that is not between precision and recall [12]. However, a higher f1-score does not always mean or translate to a better classifier, it depends on the condition of the dataset and the purpose of the classifier. Three types of f1-score metrics are used in this paper to provide a general comparison between the six classification methods in the dataset.

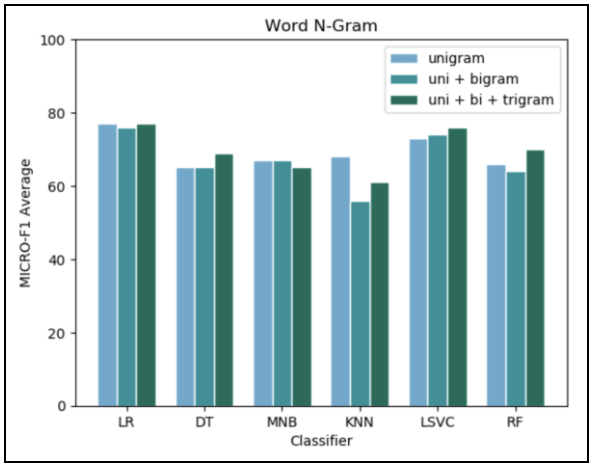

Fig. 3. Micro-average F1 Score with Word N-Gram

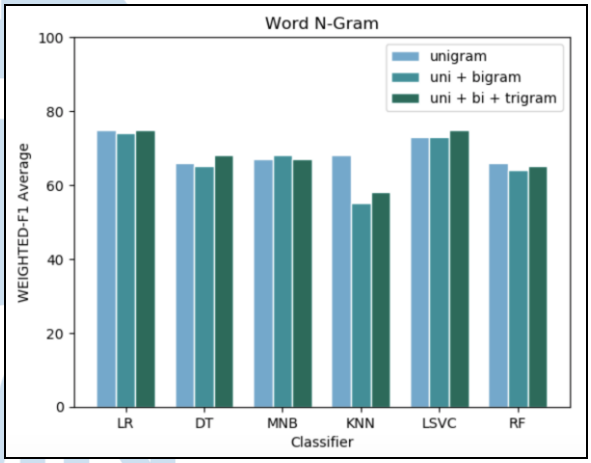

Fig. 4. Weighted-average F1 Score with Word N-Gram

As can be seen in the previous figures from Fig. 2 – Fig. 4, we use word N-Gram to evaluate each classifier using unigram, uni+bigram, and uni+bi+trigram. Other values of N in the N-gram were also assessed but showed similar results. Using the macro-average f1 score, by not taking into consideration the imbalanced dataset and calculating the harmonic mean of precision and recall of each class, Logistics Regression performs the best with word unigram, reaching a score of 0.56. However, it can also be seen that Linear SVC also performs good and showing similar result, whether using the different word n-grams, where Logistics Regression only performs best using the unigram.

Furthermore, by taking the imbalanced condition of the dataset, we calculate the micro- and weighted-average f1 score. By calculating the results globally (micro) and giving weights to each class based on their occurrences, the average f1 scores increase. The

highest score is attained again by the Logistics Regression scoring 0.77, followed tightly by the Linear SVC with 0.76 by combining the word uni+bi+trigram. Furthermore, it can also be seen more clearly that when the micro-average f1 score is the suitable metric, combining uni+bi+trigram could increase the classifiers' performance, except for KNN and Multinomial Naïve Bayes.

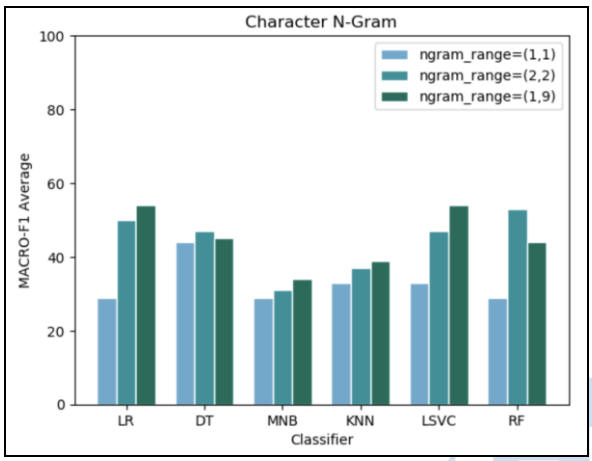

Fig. 5. F1 Score with Character N-Gram

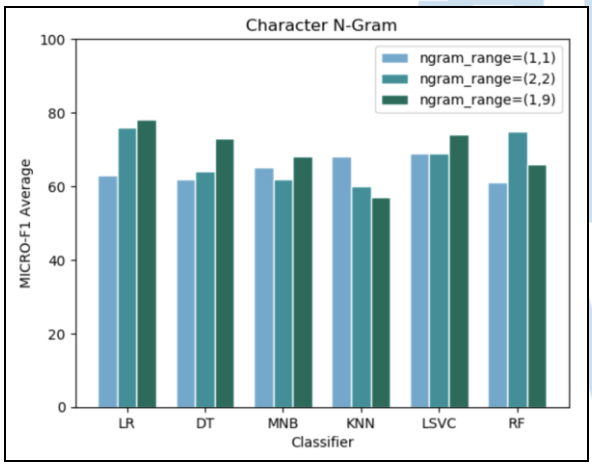

Fig. 6. Micro-average F1 Score with Character N-Gram

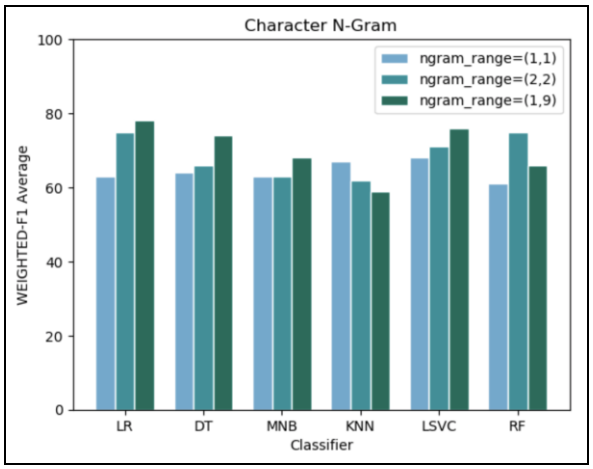

Fig. 7. Weighted-average F1 Score with Character N-Gram

Fig. 5 – Fig. 7 shows the result of using Character N-Gram as the feature extraction technique for the six classifiers. Generally, the combination of 1- until 9-gram shows better results for almost all classifiers, calculated using the macro-, micro-, and weighted average f1 score. The best score is achieved by Logistics Regression with 0.78 with character 1- until 9-gram calculated both by the micro- and weighted-average f1 score and for calculation using macro-average f1-score, Logistics Regression and Linear SVC both perform equally by achieving 0.54 score.

This research conducted experiments with many other n-gram combinations, which are not shown in the previous figures but show exciting results nonetheless. For example, when the weighted-average f1 score is essential, the following five classifiers (Logistics Regression, Decision Tree, Multinomial Naïve Bayes, Linear SVC, and Random Forest) perform almost equally using character 5-gram, as can be seen in Fig. 8. Decision Tree even outperforms all the other classifiers when the micro-average F1 score is essential, using the combination of word uni+bigram, as shown in Fig. 9.

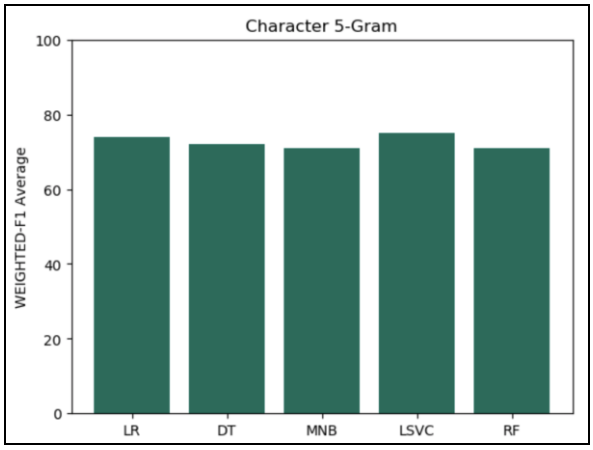

Fig. 8. Weighted-average F1 Score with Character 5-Gram

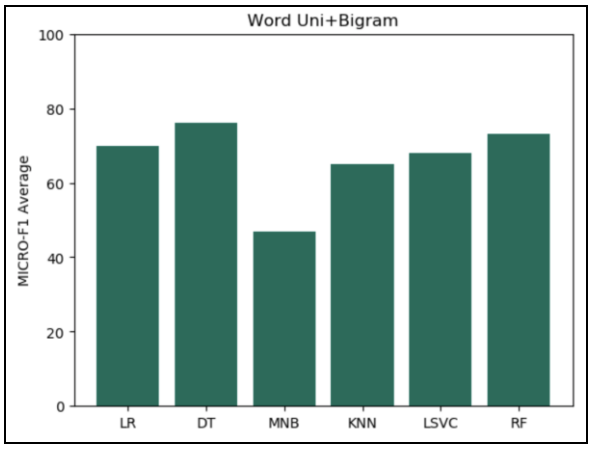

Fig. 9. Micro-average F1 Score with Word Uni+BiGram

These comparison results between each classifier using various n-gram configurations for our dataset shows that Logistics Regression is the most suitable classifier in most cases, followed by Linear SVC. However, many other tools and techniques could be integrated into the system to enhance performances such as feature extraction using word embedding, using various up-sampling and down-sampling techniques to deal with the imbalanced dataset, and many more. By further implementing different NLP techniques, the performance of the classifiers evaluated in this paper could vary greatly. In other cases, some classifiers that do not perform as well in our experiments could beat the others, and vice versa.

IV. Conclusion

This paper presented the result of our study, which provides a general comparison using three different metrics on six classifiers for classifying user feedback according to their categories. In our feedback dataset, each feedback is classified into one category from the existing four categories, which are Technical, Strategic, Content, and Other. We use three metrics for calculating the harmonic mean of precision and recall of the resulting confusion matrix in each scenario, the macro-, micro-, and weighted-average f1 score, as each function differently and might be useful in different requirements.

Generally, Logistics Regression is the most suitable classifier in most cases, followed by Linear SVC. Other classifiers also perform quite similarly based on the resulting confusion matrix, though not as good. For example, Random Forest with character bigram performs excellent compared to the others when the micro-average f1 score metric is essential, as it scores 0.75, only 0.01 less than the Logistics Regression. Another case shows that when the weighted-average f1 score matters most, Logistics Regression, Decision Tree, Multinomial Naïve Bayes, Linear SVC, and Random Forest perform almost equally.

References

[1] A. v. Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, Chichester: John Wiley and Sons, 2009.

[2] D. Pagano and B. Bruegge, "User Involvement in Software Evolution Practice: A Case Study," in *35th International Conference on Software Engineering*, San Francisco, 2013.

[3] E. C. Groen, J. Schowalter, S. Kopczyńska, S. Polst and S. Alvani, "Is there Really a Need for Using NLP to Elicit Requirements? A Benchmarking Study to Assess Scalability of Manual Analysis," in *REFSQ 2018*, Utrecht, 2018.

[4] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," Ottawa, 2015.

[5] S. Panichella, A. Di Sorbo, E. Guzman Ortega, C. A. Visaggio, G. Canfora and H. Gall, "ARdoc: App Reviews Development Oriented Classifier," in *24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, Seattle, 2016.

[6] T. Pranckevicius and V. Marcinkevicius, "Comparison of Naïve Bayes, Random Forest, Decision Tree, Support Vector Machines, and Logistic Regression Classifiers for Text Reviews Classification," *Baltic J. Modern Computing*, vol. 5, no. 2, pp. 221-232, 2017.

[7] D. Ariadi and K. Fithriasari, "Klasifikasi Berita Indonesia Menggunakan Metode Naive Bayesian Classification dan Support Vector Machine dengan Confix Stripping Stemmer," *JURNAL SAINS DAN SENI ITS*, vol. 4, no. 2, 2015.

[8] M. A. Fauzi, R. F. N. Firmansyah and T. Afirianto, "Improving Sentiment Analysis of Short Informal Indonesian Product Reviews using Synonym Based Feature Expansion," *TELKOMNIKA*, vol. 16, no. 3, pp. 1345-1350, 2018.

[9] I. Ferdino and A. Rusli, "Using Naïve Bayes Classifier for Application Feedback Classification and Management in Bahasa Indonesia," in *CONMEDIA 2019*, Bali, 2019.

[10] G. P. Wiratama and A. Rusli, "Sentiment Analysis of Application User Feedback in Bahasa Indonesia Using Multinomial Naïve Bayes," in *CONMEDIA 2019*, Bali, 2019.

[11] H. Li, "Deep learning for natural language processing: advantages and challenges," *National Science Review*, vol. 5, no. 1, pp. 24-26, 2018.

[12] Scikit-Learn, "Scikit-learn.org," Scikit-learn.org, [Online]. Available: https://scikit-learn.org/stable/. [Accessed 19 January 2020].