Comparative Analysis of Machine Learning Techniques for Splitting Identifiers within Source Code

Abeer Abdulsalam
Faculty of Art, Computing and Creative Industry, Sultan Idris Education University, Malaysia.
E-mail: abeerabdulsalam94@gmail.com

Nazre Abdul Rashid
Senior Lecture, Computing Department, Computing and Creative Industry, Sultan Idris Education University, Malaysia.

Received August 01, 2020; Accepted October 03, 2020
ISSN: 1735-188X
DOI: 10.14704/WEB/V17I2/WEB17066

Abstract

Feature location is the process of extracting identifiers within source code. In software engineering, it is a usual procedure to upgrade software by adding new features. In order to facilitate this process for the developers, feature location has been proposed to extract the significant components within the source code which are the identifiers. One of the challenging issues that faces the feature location task is handling multi-word identifiers where developers may use different type of separations among the words. Different research studies have used various types of techniques. However, recent studies have showed interest in Machine Learning Techniques (MLTs) due to their substantial performance. With the diversity MLTs, there is a vital demand to identify the most accurate one in terms of splitting the identifiers correctly. Therefore, this study aims to provide a comparative analysis of different MLTs including Naïve Bayes, Support Vector Machine and J48. The dataset used in the experiment is a benchmark data that contains vast amount of source codes along with numerous identifiers. Results showed that the best accuracy has been achieved by using the J48 classifier where the f-measure was 66%.

Keywords

Feature Location, Split Identifiers, Source Code, Naïve Bayes, Support Vector Machine, J48.

Introduction

Software engineering is an area of study that concentrates on improving the software systems by proposing approaches that facilitates the use of such software. This means facilitating the process of creating, modifying and altering software codes in order to

improve the efficiency of building the systems [1]. One of the important tasks when dealing software codes is the altering where a company for example needs to update its system. In this vein, developers are asked to add new features to the software of such system. Even if the same developers who built the system are still working for the company, it is very hard for them to remember every single detail about their own codes. What if these developers are leaving the company, and new developers are asked to modify other's code. Apparently, it would be a difficult task for them to add minor features. This is because they have to find the exact location within thousands of code's lines that is related to such feature. As well as, they have to consider the dependencies of such feature where some functions are depending on specific procedure. Therefore, altering such procedure would definitely lead to deactivate or affect the dependent functions [2].

For this purpose, different search tools have been proposed to help developers for locating the exact function, entity or feature. This task of searching is known as Feature Location and it aims to identify the location of an existing feature or entity [3-5]. Suppose there is a feature or entity within the code called Patient. Identify the location of such entity and their dependencies such as 'Patient name', 'Patient ID' or 'Patient age' can be considered as one of the feature location tasks. However, feature location is facing different challenges such as the multi-word identifiers. Multi-word identifiers are the entities that consist of multiple terms such as 'Patient Name'. Since most of the programming languages are prohibiting the process of separating the multi-word entities with blank space such as 'Patient name' therefore, developers tend to use several approaches to declare these entities. Some of them used the camel case which aims to capitalize the first character of each word while leaving the other characters to be lower-case such as 'PatientName'. Other developers are using some special characters such as punctuations to do the separation task (e.g. Patient_Name). Another approach is used by the developer which is the digit where the separation is performed using digits (e.g. Patient2Name). Finally, some developers are using a different approach which aims to abbreviate one or more words within the multi-word identifiers such as 'PName', 'PatientN' or 'PN'.

Obviously, all these approaches would significantly affect the accuracy of feature location task where the developer may search for the entity 'Patient' within a code written by a developer who uses camel case or abbreviated mechanisms. Apparently, the feature location task would not be able to locate the query of 'Patient'. This is because the search approaches used in the feature location are mainly depend on the orthography of the word in which the word 'Patient' is different than the word 'Patients' even though they have the same meaning, but the plural 's' makes the second word different in terms of the

orthography. Therefore, there is an emergent demand to separate the multi-word identifiers in order to improve the feature location task.

Recently, Natural Language Processing (NLP) has been utilized in order to separate the multi-word. NLP aims to use different tasks including lexical, semantic and syntactic [6]. Lexical task aims to use the orthography of the word where the organization of the characters will be considered. Semantic task aims to use the meaning of words such as 'salary' and 'income' in order to separate the multi-words. Usually, semantic task is mainly depending on external knowledge source such as dictionary in order to identify the synonyms or expand the abbreviations. Finally, the syntactic task aims to use the grammatical aspect of the words such as noun or verb in order to understand the meaning of words [7].

Another domain of study is being involved in the task of separating the multi word identifiers which is the Machine Learning Techniques (MLT) [8]. Machine learning aims to use previous or historical data in order to build a model that is able to handle new cases [9]. In this manner, a historical data of source codes will be used for the process of training the machine learning. Within the training, the machine learning model will review plenty of cases for the multi-word identifiers. In each case, the NLP task can be used to indicate the use of separation for each instance. Then, the machine learning model will address the occurrence of special cases such as the occurrence of digits, special characters or the camel case. Hence, it is possible to predict the new instances based on such occurrences.

In fact, plenty of researchers have showed more interesting in using the machine learning in the last decade for different domain of studies. It has proven a fair performance in classification, prediction and extraction tasks. However, with the diversity of MLTs, different methods can be used in which each method has its own strengths and weaknesses. Therefore, it is necessary to identify the most accurate one in terms of splitting the identifiers. This study aims to address this challenge by accommodating a comparative analysis of different MLTs for the splitting identifiers task.

Related Work

Recent studies in the literature has showed more interest in using IR techniques in order to identify feature location. For example, Marcus & Maletic [10] introduced a method for feature location using LSI. The proposed method has been intended to determine the correspondences within the source code in order to classify its components. One of the

significant components that the authors have focused on is the identifiers. LSI has been applied on the source code in order to identify the similar identifiers. In addition, Poshyvanyk et al. [11] introduced a search tool for the.Net framework programming environment. Such tool has been inspired by the NLP tasks for the purpose of finding the similar parts of the code in accordance with the query posted by the developer.

With the difficulties that faced the task of feature location such as splitting the identifiers recently, researchers have focused on such obstacles. Lawrie et al. [12] presented a method for splitting identifiers automatically. The proposed method has utilized the regular expression in order to address the punctuations, numbers and capitalized letters within the multi-word identifiers.

On the other hand, Field et al. [13] has presented a method for splitting identifiers based on dictionary. In fact, the authors have used a dictionary that contains the splitted identifiers. Hence, a lexical similarity has been performed between the dictionary contents and the source code in order to extract the similar identifiers.

Enslen et al. [14] presented a statistical method for splitting identifiers using term frequency. The idea behind such method lies on the possibility of frequent occurrence of a particular identifier. For example, the identifier 'Patient' would be frequently occurred with other words such as 'PatientID', 'PateintName' and 'PatientAge'. In this manner, addressing the frequent occurrence of such identifier may help the process of splitting its instances.

Lawrie & Binkley [5] presented a normalization method for the vocabulary within the source code. The proposed method was intended to extend the abbreviations with their corresponding words. Consequentially, a matching process has been conducted between the source codes with a dictionary in order to perform the splitting.

Recently, more interests have been shown toward utilizing the machine learning techniques in the process of splitting the identifiers. Basically, machine learning techniques have showed superior performance regarding to its powerful mechanism of learning from examples. An example data of the multi-word identifiers along with their splitting form can be used by the machine learning to build a model that has the ability to separate new data.

Based on such concept, [15] has proposed a machine learning method based on Naïve Bayes. The authors have utilized different textual features along with the NB in order to

split identifiers. These features consist of camel case, numbering, punctuations, and spell checker.

MTLS Comparison

In this section the proposed comparison will be carried out. In order to enable such comparison, several phases should be accomplished. In this regard, the workflow of the comparison will be consisted of acquiring a benchmark dataset, extracting features and applying the MLTs. These phases are being depicted in Fig. 1.

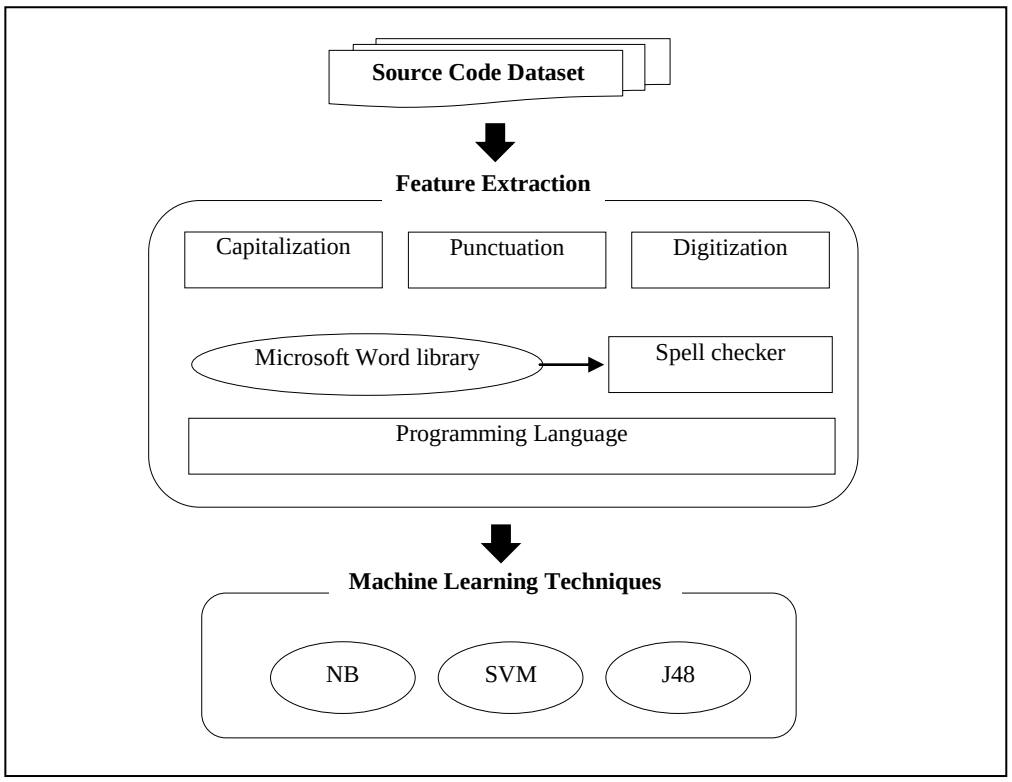

Figure 1 Workflow of the comparison

1) Dataset

In fact, one of the important task to perform the identifiers splitting is to determine a dataset that contains source codes in which numerous identifiers can be used. Therefore, this study will use a dataset that has been introduced by Enslen et al. [14]. This dataset consists of 9000 open source programs using multiple programming languages such as Java, C and C++ from SourceForge which is a website consists of numerous programming projects with its source code.

In order to describe the procedures conducted to obtain the data, it is necessary to mention firstly that the data was gathered from this website (http://www.cs.loyola.edu/~binkley/ludiso). Once downloading the data, it would be obvious that it is arranged in a text file un-structurally. Therefore, a transformation task will be conducted in order to make the data suitable for processing through MLTs. Table 1 shows the output of such task where the significant columns are being obtained. The first columns contains the multi-word identifier that needs to be splitted, second column shows the correct way to split the identifier, and finally, the third column shows the class label which considered to be the number of required operation to split the identifiers. Based on the third column, the classification will be conducted where the classifier will be trained to predict this class.

2) Feature Extraction

This phase aims to extract the features of each multi-word identifier. Such features are the characteristics that are related to the morphology of the word where the number of capital letters, punctuation, digit and the spell are being considered.

Such characteristics contribute toward facilitating the classification to correctly split the multi-word identifiers. The features that will be used in this study have been brought from the study of Alanee & Murad [15]. The following sub-sections will discuss these features.

Table 1 Sample of the transformed dataset

Multi-word identifier	Splitted Identifier	No. of required splits (Class)
BaseExc	Base-Exc	1
BlastBin2Csv	Blast-Bin-2-Csv	3
BlockRowPixels	Block-Row-Pixels	2
BooleanECD	Boolean-ECD	1
BugBaseQuery.QUERY_UPDATE_BUGREPORTS	Bug-Base-Query-QUERY-UPDATE-BUGREPORTS	6
BulletinConstants.TAGSTATUS	Bulletin-Constants-TAGSTATUS	3
CAInfo.SELFSIGNED	CAInfo-SELFSIGNED	3

- **Capitalization**

Majority of programmers are utilizing the Camel Case mechanism to separate the multi-word identifiers within the process of declaration. Camel Case is intended to make the

first character located in the first word as capital while lowercasing the rest characters. Similarly, it will make the first character in the second word as capital while lowercasing the rest characters. Hence, the capitalization feature is intended to calculate the number of capital characters which may facilitate the process of indicating the multi-word identifiers. Table 2 shows an example of counting the capital characters.

Table 2 Sample of capitalization feature

Identifiers	Number of capital characters
Client	1
ClassType	2
SearchFileServer	3
CameraPhoneOntologyApp	4

- **Punctuation**

As mentioned earlier, there is no a standard way to separate the multi-word identifier. Therefore, programmers may use punctuation such as dash and underscore to separate the identifiers instead of camel case. Hence, this feature will attempt to calculate the number of these punctuation in order to indicate the number of separation required. Table 3 shows an example to do such computation.

Table 3 Sample of punctuation feature

Identifiers	Punctuation	Number of punctuations
CONV_OLD_CONVERT	_	2
Constants.ATTR	.	1
wu-ftpd	-	1

- **Digit Count**

Another way that programmers may express their own way of separation is by using numbers or digits to divide the multi-word identifiers. In this manner, calculating the number of digits within the identifiers would be a good indicator for the separation process. Table 4 shows an example of such feature.

Table 4 Sample of digitization feature

Identifiers	Number of digits
Is7bitNon	1
Java2Object	1
PK11Object	2
Row2Ptr	1

- **Spell Checker**

This feature is intended to use the spell checker that is being used by Microsoft Word. As shown in Fig. 2, Microsoft Word uses a spell checking for the multi-word such as 'CreateProcess' in order to give recommendation for correcting the word. For this purpose, Microsoft utilizes a dictionary for English words to perform a mapping for potential words from the dictionary.

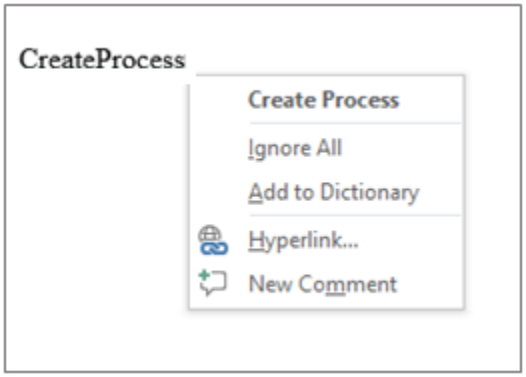

Figure 2 Sample of Microsoft spell checker [15]

As shown in Fig. 2, as a recommendation MS word will attempt to correct the word by giving potential separation. In this manner, this feature will be utilized in this research by counting the number of words depicted in the recommendation. In the case of Fig. 2, there are two words thus, this feature will be denoted by 2.

In order to describe the mechanism of applying this feature, a spell checker called Hunspell has been used that can be accessed through this website (http://www.crawler-lib.net/nhunspell).

In the suggestions box, the correct word can be shown as 'Create Process' therefore, this study aims to use the number of words shown in the suggestion box (i.e. which is 2) in order to indicate the required number of splitting.

- **Programming Language**

This feature aims to address the type of programming language used to declare the identifiers. In fact, there are numerous kinds of programming languages such as C, C++, C#, Java, Python, PHP, Matlab, R programming and others. Each programming language has its own syntax that is being used to declare the identifiers. For example, in C, C++, C# and Java declaring a variable should specify its type as:

```
int Employee-Salary;
Employee-Salary = 2000;
```

While in other programming language such as PHP or Python, there is no need to specify the type as:

```
#Employee-Salary = 2000;
```

Therefore, there is a vital demand to consider the type of programming language used in the feature location analysis in order to understand the differences among these languages. Unlike the previous features, this feature does not need to be developed since it is located in the original dataset.

3) Classification

This section aims to discuss the classification method that have been carried out where the comparison will take a place. For this purpose, three popular classifiers are being used including Naïve Bayes, Support Vector Machine and J48. The following sub-sections will discuss these classifiers in further detail.

- **Naïve Bayes**

This classifier uses the probabilities of each feature associated with every data instance in order to predict the class label [16]. This can be conducted by analyzing the training data and compute probabilities for each feature in accordance to every class label. In other words, the number of times feature f has led to a class label C. Such probability can be calculated based on the following equation:

$$C(d) = \underset{i=}{\operatorname{argmax}} |C| \ (P(C_i|d)) \quad (1)$$

- **Support Vector Machine**

This algorithm has been applied widely in different domain of interests [17]. SVM aims to classify the multi-word identifiers based on a linear mechanism where the data is being divided using a hyperplane into two parts; identifiers that belongs to class *i*, and identifiers that do not belong to class *i*. This process is being repeated for every class label until all the data is classified. The mechanism of computing the hyperplane can be depicted in the following equation:

$$f(\vec{x}) = sgn\left((\vec{x} \times \vec{w}) + b\right) = \begin{cases} +1: (\vec{x} \times \vec{w}) + b > 0 \\ -1: Otherwise \end{cases} \quad (2)$$

- **J48**

This classifier is also known as Decision Tree which aims to classify the instances based on a tree that simulates the features that contribute toward classifying the instance into a class label [18]. This can be performed by depicting the features on the branches of the tree which lead to a class label that represented in the leaves.

Results

To evaluate the classification method, the traditional machine learning evaluation methods will be used which consists of precision, recall and f-measure. First, precision can be computed based on the following formula:

$$Precision = \frac{|TP|}{|TP| + |FP|} \quad (3)$$

Precision refers to the number of correct extracted correspondences (TP) in accordance to the total number of correct correspondences and false correspondences identified by the system (TP and FP). Second, recall can be calculated as:

$$Recall = \frac{|TP|}{|TP| + |FN|} \quad (4)$$

Recall refers to the number of correct extracted correspondences (TP) in accordance to the total of number of extracted correspondences and the needed correspondences but not identified by the system (TP and FN). Because precision or recall alone cannot accurately evaluate the match quality, so it is necessary to consider a trade-off between them by using the combining measure F-measure which formulated as:

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (5)$$

Based on the above mentioned evaluation metrics, the three classifiers have been trained on 80% of the data and tested on 20% of the data. Table shows the results of the three classification methods.

Table 5 Results of the three classifiers

Classifier	Precision	Recall	F-measure
NB	0.602	0.617	0.601
SVM	0.628	0.645	0.628
J48	0.659	0.668	0.660

As shown in Table 5, NB classifier has obtained a 0.602 of precision, 0.617 of recall which leads to 0.601 of f-measure. While SVM classifier has acquired a 0.628 of precision, 0.645 of recall which leads to 0.628 of f-measure. Finally, J48 gained a 0.659 of precision, 0.668 of recall and 0.660.

It is obvious that SVM has outperformed the NB based on precision, recall and f-measure. The reason behind such superiority lies on the ability of SVM to deal with large number of class labels. However, the J48 classifier has outperformed the other two classifiers; SVM and NB. The reason behind the outperformance of J48 lies on the capability of examining too many cases where all the features are being addressed separately and with combinations. Figure 3 depicts the superiority of J48 over the other classifiers.

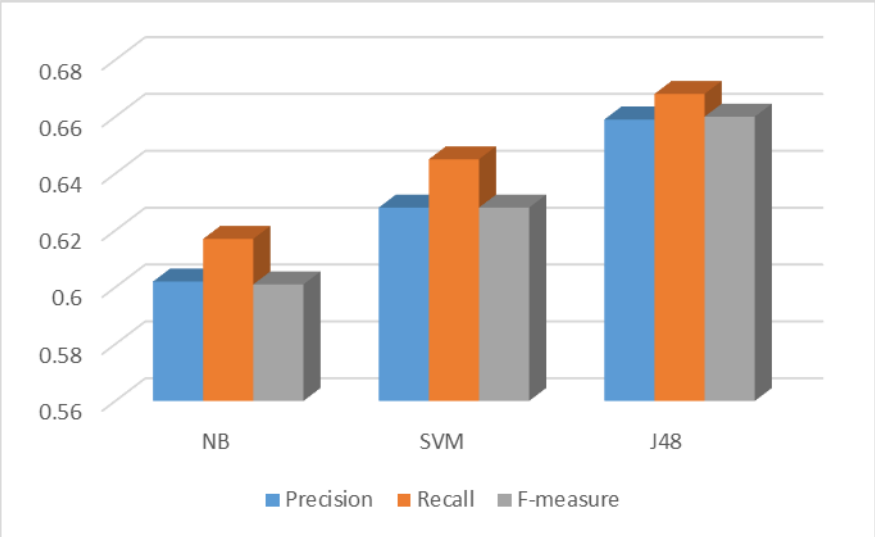

Figure 3 Comparison among NB, SVM and J48

Conclusion

This paper has provided a comparative analysis among different machine learning techniques including NB, SVM and J48 for the process of splitting identifiers within the source code. A benchmark dataset of source codes have been used in the experiments. Results showed that J48 has outperformed the other classifiers in terms of the accuracy of splitting identifiers. For future studies examining more sophisticated data representations such as the character embedding would yield promising results.

References

Carlo, G., Mehdi, J., & Dino, M. (2002). *Fundamentals of software engineering*. Prentice Hall PTR.

Roger, S.P. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

Nouh, A., Natalia, D., Michael, L.C., & Jonathan, M. (2013). Improving feature location by enhancing source code with stereotypes. *In 29th IEEE International Conference on Software Maintenance (ICSM)*, 300-309.

Bogdan, D., Meghan, R., Malcom, G., & Denys, P. (2013). Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process, 25*(1), 53-95.

Lawrie, D., & Binkley, D. (2011). Expanding identifiers to normalize source code vocabulary. *In 27th IEEE International Conference on Software Maintenance (ICSM)*, 113-122.

Daniel, J., James, & H.M. (2000). *An introduction to natural language processing, computational linguistics, and speech recognition*. Ed: Prentice Hall Englewood Cliffs.

Robert, D., Hermann, M., & Harold, S. (2000). *Handbook of natural language processing*. CRC Press.

Ian, H.W., & Eibe F. (2005). *Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann*.

Kotsiantis, S.B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, *160*(1), 3-24.

Jonathan, I.M., & Andrian, M. (2001). Supporting program comprehension using semantic and structural information. *In Proceedings of the 23rd International Conference on Software Engineering*, 103-112.

Denys, P., Andrian, M., Yubo, D., & Andrey, S. (2005). IRiSS-A Source Code Exploration Tool. *In ICSM (Industrial and Tool Volume)*, 69-72.

Dawn, L., Henry, F., & David, B. (2007). Quantifying identifier quality: an analysis of trends. *Empirical Software Engineering*, *12*(4), 359-388.

Henry, F., David, B., & Dawn, L. (2006). An empirical comparison of techniques for extracting concept abbreviations from identifiers. *In Proceedings of IASTED International Conference on Software Engineering and Applications (SEA'06)*, 01-06.

Eric, E., Emily, H., Lori, P., & Vijay-Shanker, K. (2009). Mining source code to automatically split identifiers for software analysis. *In 6th IEEE International Working Conference on Mining Software Repositories, MSR'09*, 71-80.

Nahla, A., & Masrah, A.A.M. (2017). A hybrid method of feature extraction and Naive Bayes classification for splitting identifiers. *Journal of Theoretical and Applied Information Technology*, *95*(7), 1549-1557.

Alshaikhdeeb, B., & Ahmad, K. (2017). Feature selection for chemical compound extraction using wrapper approach with Naive Bayes classifier. *In 6th International Conference on Electrical Engineering and Informatics (ICEEI)*, 1-6.

Wen, Z., Taketoshi, Y., & Xijin, T. (2008). Text classification based on multi-word with support vector machine. *Knowledge-Based Systems*, *21*(8), 879-886.

Lior, R. (2007). *Data mining with decision trees: theory and applications*. World scientific.