Journal of Theoretical and Applied Information Technology
15th April 2017. Vol.95. No 7
© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A HYBRID METHOD OF FEATURE EXTRACTION AND NAÏVE BAYES CLASSIFICATION FOR SPLITTING IDENTIFIERS

[1]NAHLA ALANEE, [2]MASRAH AZRIFAH AZMI MURAD

Faculty of Computer Science and Information Technology Universiti Putra Malaysia, 43400 Serdang, MALAYSIA

E-mail: [1]nahlaalanee@yahoo.com, [2]masrah@upm.edu.my

ABSTRACT

Nowadays, integrating natural language processing techniques on software systems has caught many researchers' attentions. Such integration can be represented by analyzing the morphology of the source code in order to gain meaningful information. Feature location is the process of identifying specific portions of the source code. One of the most important information lies on such source code is the identifiers (e.g. Student). Unlike the traditional text processing, the identifiers in the source code is formed as multi-word such as 'Employee-Name'. Such multi-words are not divided using white space, instead it can be formed using special characters (e.g. Employee_ID), CamelCase (e.g. EmployeeName) or using abbreviations (e.g. EmpNm). This makes the process of extracting such identifiers more challenging. Several approaches have been performed to resolve the problem of splitting multi-word identifiers. However, there is still room for improvement in terms of accuracy. Such improvement can be represented by utilizing more robust features that have the ability to analyses the morphology of identifiers. Therefore, this study aims to propose a hybrid method of feature extraction and Naïve Bayes classifier in order to separate multi-word identifiers within source code. The dataset that has been used in this study is a benchmark-annotated data that contains large number of Java codes. Multiple experiments have been conducted in order to evaluate the proposed features independently and with combinations. Results shown that the combination of all features have obtained the best accuracy by achieving 64.7% of f-measure. Such finding implies the usefulness of the proposed features in terms of discriminating multi-word identifiers.

Keywords: *Feature Location, Split Identifiers, Feature Extraction, Naïve Bayes, Source Code*

1. INTRODUCTION

Software engineering is the process of analyzing software systems in order to improve the efficiency [1]. This process can be explained as supplying recommendation, illustration and providing reports for enhancing the performance of a particular system. To do so, a comprehensive analysis should be concentrated on the significant features shown in the source code of the system [2].

Analyzing these features within the code provides valuable understanding of the intention of the code which facilitate the process of re-use and modification that would be performed on such code. One of the common concepts that are frequently used in any source code is the identifiers (e.g. string Name) [3]. Extracting such identifiers would offer a good opportunity to understand the headlines of the source code where the programmer declares all the objects that will be used in the system (e.g. student, employee, etc.) [4]. In addition, the process of extracting identifiers has a significant impact on improving feature locations. Feature location aims to extract specific portion of the source code that typically correspond to the developer's query [5].

Since the source code is written by the natural language, analyzing the source code can be done by using Natural Language Processing techniques. However, there are multiple differences between the regular text and the source code. In the source code, the multi-word identifiers are written without a space between them, instead several strategies can be used. First, it may be divided using special characters such as 'Employee-Name' or 'Employee_Name' [3]. Second, it may be written using 'CamelCase' approach, this approach aims to capitalize the first letter of the first words and the first letter of the second word without spacing (e.g. EmployeeName) [6]. Apart from the multi-word splitting problem, the identifiers in the source code

© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

may be written using abbreviations such as 'Emp' for 'Employee' [7].

The most complicated splitting mechanism is the multi-word that are separated neither by special characters nor by CamelCase. In such case, both words are in lowercase and attached to each other without a white space such as 'studentid'. Hence, there is no unified or agreement mechanism to write the identifiers in the source code. This can make extracting such identifiers from the source code a challenging task

Several approaches have been proposed to resolve such problem [3, 4, 6, 7]. Yet, there is a need for enhancement in terms of recall and precision. Such requirement of improvement is represented by using more robust features that have the ability to recognize the splitting words.

Therefore, this study aims to identify an extension of features with machine learning technique in order to separate the multi-word identifiers. These features will have the ability to utilize the characteristics of multi-word identifiers in the source code. After that, supervised machine learning technique of Naïve Bayes will be used in order to classify the identifiers based on the required number of splitting.

In this vein, the objectives of this paper can be represented as developing the extended features and combining it with Naïve Bayes classifier. The classification will be based on the number of separations required to divide the multi-word identifiers extracted from source codes.

The paper is being organized as; Section 1 provides the introduction, Section 2 discusses the related work, Section 3 illustrates the process of carrying out the proposed method. Section 4 depicts the experimental results obtained by the proposed method. Section 5 concludes the research findings and highlights both limitations and future directions.

2. RELATED WORK

Nowadays, researchers pay more attention in terms of applying information retrieval approaches for extracting identifiers, and identifying feature location from the source code. For instance, Marcus & Maletic [8] have proposed a Latent Semantic Indexing (LSI) method for software engineering applications. Such method aims to classify the portions of the source code by identifying the similarity among such portions. One of these portions is the identifiers. The authors have linked the concepts (i.e. identifiers) with each other in a matrix of similarity. In this manner, the lexical similarity among the identifiers will be examined.

In the same manner, Poshyvanyk et al. [9] have proposed a Visual Studio plugin for enhancing the process of search within the source code based on natural language processing techniques. In this vein, the developer will be able to type a query (usually as identifier) in order to get relevant portion from the source code. The proposed tool works by identifying the most similar portion in the source code with the typed query in terms of lexical similarity.

However, one of the challenging task that facing the mapping the between query typed by the developer and the relevant portion within the source code is the multi-word identifiers. Obviously, many identifiers are being declared with multiple words. Since the programming languages hinder the developer to separate the multi-word identifiers by a blank space therefore, developers tend to use multiple approaches for the separation whether using punctuation, digit or using CamelCase. Hence, there is a vital demand to accommodate a separation process in order to divide the multi-words identifiers into their original form.

Binkley & Lawrie [4] have addressed this problem by proposing an approach for handling the process of dividing multi-word identifiers automatically. They have used regular expression approach in order to exploit the CamelCase and special characters such as 'underscore'.

Similarly, Field et al. [7] have proposed a dictionary-based approach where numerous keywords and tokens are located. Then, a process of string-matching has been performed in order to match the words between the source code and the dictionary.

Enslen et al. [3] have proposed a statistical approach for dividing multi-word identifiers based on word frequency. Their hypothesis emphasis that the occurrence of an identifier should be frequent such as 'Employee-Name' and 'Employee-Income'. Therefore, the process of dividing multi-word identifiers lies on analyzing the frequency of such identifiers where the clues (e.g. Employee) could be appeared.

Lawrie & Binkley [6] have proposed a normalization technique for the vocabulary in the source code. Their technique aims to expand the abbreviations before matching any vocabulary dictionary. Their hypothesis lies on the frequent use

© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

of abbreviations in the source code such as 'Emp-Name'.

It is obvious that there is a diversity in terms of the techniques used for dividing the multi-word identifiers. for this purpose, this study aims to examine multiple features such as lexical features (i.e. punctuation, capitalization and digit) and dictionary-based approach (i.e. spell checker) in order to highlight the most appropriate feature set for the separation process. Such examination will significantly contribute toward improving the effectiveness of the classification results.

The key difference of this study lies on the examination of different kind of the features including morphological features (i.e. capitalization, containing digit and containing punctuation) and dictionary-based feature (i.e. spell checker).

3. PROPOSED METHOD

The proposed method consists of four main phases as shown in Fig. 1 including Dataset, Transformation, Feature Extraction and Classification. Dataset phase discusses the data that will be used in the experiment including the source of such data, details and characteristics. Whereas, transformation phase discusses the preparation tasks that have been conducted in order to turn the data into an appropriate form for representation. Feature extraction phase is associated with the contribution of this study in which multiple features are being developed to enhance the process of splitting identifiers. Finally, classification phase is associated with the type of machine learning used to classify the instances based on the developed features.

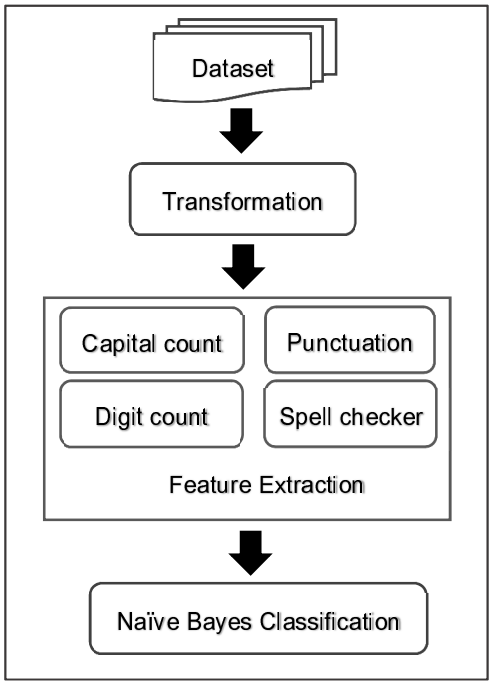

Figure1. Proposed method

3.1. Dataset

In order to apply the proposed method of splitting identifiers, it is necessary to find a benchmark dataset of a source code where experts have splitted the identifiers manually which can facilitate the process of training using machine learning technique. For this purpose, a benchmark of source code has been collected from the study of Enslen et al. [3] that will be used in this study. Such data contains 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. As shown in Figure 2, the data is formed un-structurally in a text file containing multiple information such as the ID number of the identifiers, the original identifiers, programming language used, the name of the program, splitted identifiers by experts, number of splitting and other information.

© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

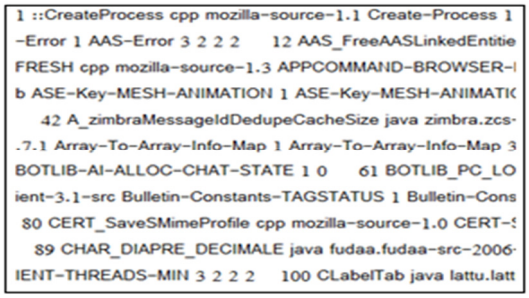

Figure 2. Sample of the dataset

3.2. Transformation

As mentioned earlier, the data should be transformed into an appropriate representation in order to facilitate the feature extraction. This can be represented by dividing each identifier with its information in a record in order to be relational data. Obviously, this requires naming the attributes' labels and accommodate a filtering task to avoid the unwanted attributes. This task has been performed using a tab delimiter separation mechanism. Fig. 3 shows the results of this phase.

ID	Original	Language	Splitted	Class
1	::CreateProcess	cpp	Create-Process	1
2	::DrawThemeTab	cpp	Draw-Theme-Tab	2
3	::GetFrontWind...	cpp	Get-Front-Wind...	4
4	::GetPrivateProf...	cpp	Get-Private-Prof....	3
5	::GetScriptMana...	cpp	Get-Script-Mana....	3
6	::GetTokenAt	cpp	Get-Token-At	2
7	::JS_DefineUCPr...	cpp	JS-Define-UC-Pr....	3
8	::SetMenuItemH...	cpp	Set-Menu-Item-....	4
9	::SetResLoad	cpp	Set-Res-Load	2
10	::fread	cpp	fread	0

Figure 3. Transforming the data

As shown in Fig. 3, the data has been transformed into five columns including (i) ID, (ii) original identifier, (iii) programming language used, (iv) splitted identifiers, and (v) the class label. Note that, the class label is considered to be the number of splitting required for each identifier, and it ranged from 0 to 12, this leads to 13 class labels.

3.3. Feature Extraction

Basically, features can be defined as the characteristics and properties of each instance where specific description of the instance can be depicted [10]. For example, the length of a given word could be a significant feature that may indicate the class label of this word. In this vein, features play an essential role in terms of the classification where the significant feature that has the ability to accurately describe the instance would definitely improve the performance of the classification. Vice versa, the weak feature would indeed affect the performance of the classification negatively. Because of that, this study aims to develop accurate features that have the ability to discriminate the situations of splitting identifiers. For this purpose, four features are being developed including capital count, punctuation count, digit count and spell checker. These four features can be illustrated as:

3.3.1 Capital Count

Most of the developers are using the CamelCase in order to declare an identifier. CamelCase aims to capitalize the first letter in the first word, as well as, capitalize the first letter of the second words, with keeping the remaining letters in lower-case. In this manner, counting the capital letters for each identifier would significantly indicate the required number of splitting. Table 1 depicts a sample of this feature.

Table 1. Sample of capital count feature

Original Identifiers	Count Capital
CreateProcess	2
DrawThemeTab	3
GetTokenAt	3
GetFront	2

3.3.2 Punctuation Count

Similar to the CamelCase, sometimes developers are using special characters such as '*_@#&' in order to separate the multi-word identifiers instead of capital letters. Therefore, counting these punctuations would indicate the number of splitting required for each identifier. Table 2 depicts a sample representation of this feature.

Table 2. Sample of punctuation count feature

Original Identifiers	Punctuation	Punctuation count
create-process	-	1
draw_theme_tab	_	2
get/token/at	/	2
get|front	|	1

3.3.3 Digit Count

Similar to both CamelCase and punctuation, some developers are using numbers to separate the multi-word identifiers. Apparently, exploiting the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

count of the occurrence for these numeric values would indicate the number of splitting required for each identifier too. Table 3 depicts a sample of digit count feature representation.

Table 3. Sample of digit count feature

Original Identifiers	Digit Capital
Max2stack	1
Value2use	1
Get2class	1
Set3value	1

3.3.4 Spell Checker

This feature aims to utilize the spell checker that is being used by Microsoft Word. As shown in Fig. 4, Microsoft Word utilizes a spell checking for the multi-word such as 'CreateProcess' in order to provide suggestions for correcting the word. To do so, Microsoft uses a dictionary for English words in order to accommodate a matching for possible words from the dictionary.

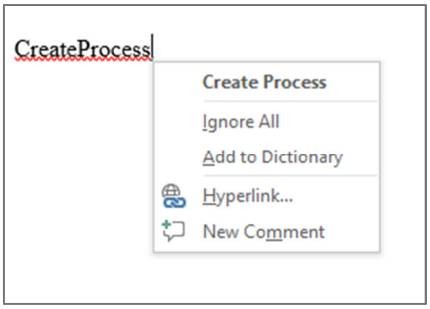

Figure 4. Sample of Microsoft spell checker

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.

3.5. Classification

In this phase, the classification of the identifiers will be conducted in which every identifier will be classified into its actual class label (i.e. number of required splitting). To do so, a Naïve Bayes classifier is being used. The reason behind using such classifier lies on its ability to identify the performance of each feature independently [11]. This can enable us to examine the performance of each feature separately in order to identify the best combination of these features. Naïve Bayes is using a training data shown in Fig. 5 where each identifier is being represented with all the features and the class label. This training data will be used in order to compute the probability of each feature with corresponding class label to classify the testing data that does not contain a class label.

Training

ID	Original	Capital Count	Punctuation Count	Digit Count	Spell Count	Class
1	::CreateProcess	2	0	0	6	1
2	::DrawThemeTab	3	0	0	8	2
3	::GetFrontWind...	5	0	0	16	4
4	::GetPrivateProf...	4	0	0	14	3
5	::GetScriptMana...	4	0	0	14	3
6	::GetTokenAt	3	0	0	8	2
7	::JS_DefineUCPr...	6	1	0	13	3
8	::SetMenuItemH...	6	0	0	17	4

Testing

ID	Original	Capital Count	Punctuation Count	Digit Count	Spell Count	Class
10	::fread	0	0	0	2	
11	AAS_Error	4	1	0	5	

Figure 5. Training and Testing representation used by NB classifier

As shown in Fig. 5, the training set is containing the class labels whereas the testing set does not contain the class labels in which the NB classifier is required to classify the class label.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Naïve Bayes classifier aims to classify the instances based on a probabilistic model in which the maximum probability of certain class label, will be assigned to the testing instance using the following formula:

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

where $P(C_i|d)$ is the probability of a document d given a class C_i, and P(d) is the probability of document d.

3.6. Evaluation

In order to evaluate the classification process, the common information retrieval metrics precision, recall and f-measure have been used to for this purpose. Precision can be computed as:

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

Precision is defined as the number of correct identified matches compared to the total number of correct matches and false matches identified by the system. On the other hand, recall can be computed as:

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

Recall is defined as a number of correct identified matches compared to the total of number of correct matches and the needed matches but not identified by the system. Finally, f-measure can be computed as:

$$F - measure = \frac{2Pr \times Re}{Pr + Re} \quad (3)$$

4. RESULTS

In order to assess the performance of the proposed features, three experiments have been conducted. First experiment will be concentrated on the independent performance for each feature (shown in Fig. 6). Second experiment will be concentrated on the pair combination between the features (shown in Fig. 7). Third experiment will be focused on the full combinations among the features (shown in Fig. 8).

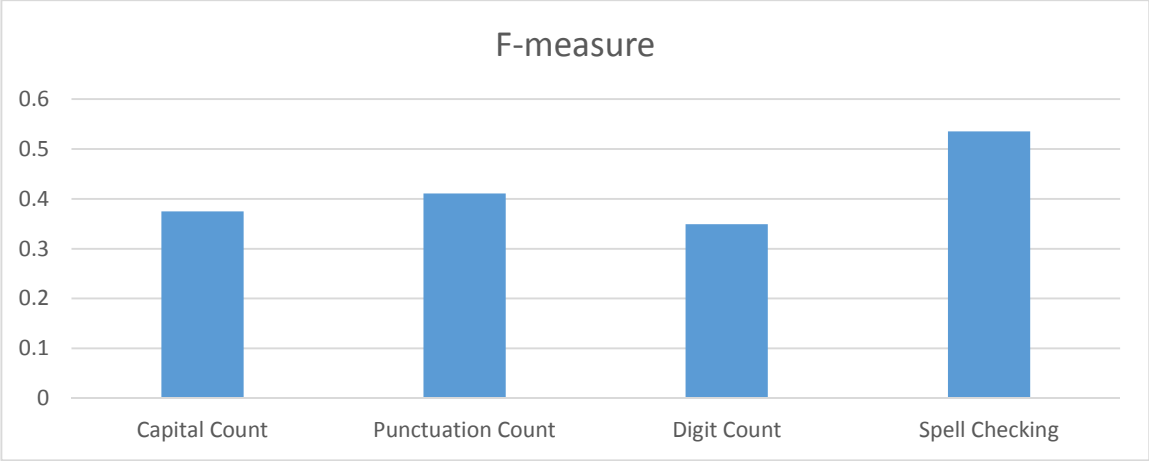

Figure 6. Performances of independent features

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

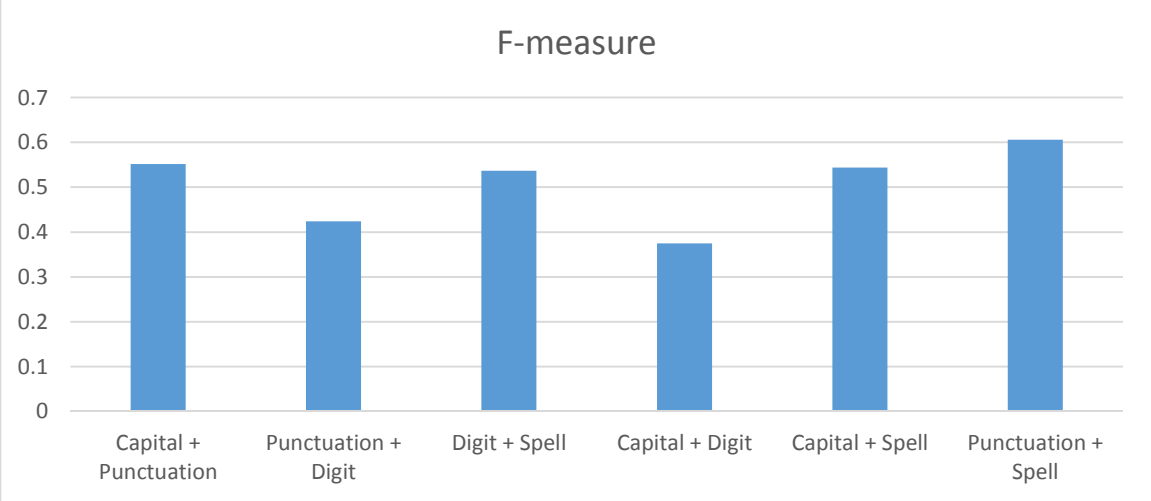

Figure 7. Performances of pair combinations among the features

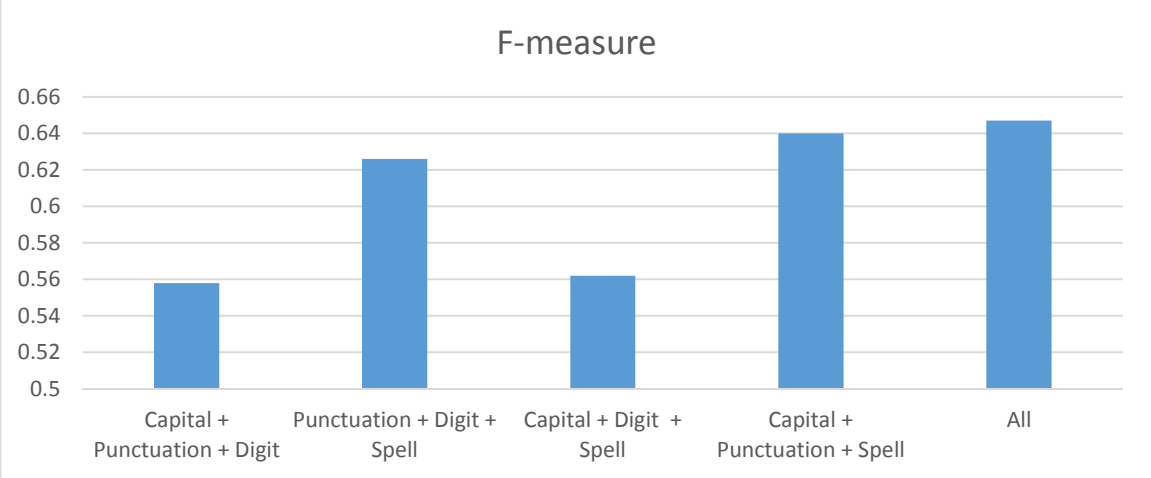

Figure 8. Performances of full combinations among the features

As shown in Fig. 6, spell checking feature has outperformed the other features by achieving an f-measure of 53.9%. This is due to the tremendous capability of using dictionary that contains large amount of English words which the spell checker utilized to retrieve the correct number of words. In addition, punctuation feature has outperformed the remaining features by obtaining an f-measure of 41.1%. This reveals that the separation of multi-word identifiers in the dataset was mostly performed using one of the special characters. Furthermore, capital count feature has followed by the punctuation by achieving an f-measure of 37.5% and outperforming the digit count feature which achieved an f-measure of 34.9%. This was expected because developers would separate the multi-word identifiers using the CamelCase more than the use of digit.

On the other hand, as shown in Fig. 7, the highest performance for the pair combination was obtained by the combination of punctuation and spell by achieving an f-measure of 60.6%. This was expected because both features have shown superior results independently. Surprisingly, the combination of capital and punctuation shown the second highest value of f-measure by achieving 55.2%. On the other hand, the two combinations that contains spell feature shown good results including capital with spell and digit with spell by achieving 54.8% and 54.1% respectively. Finally, the combination of capital and digit shown the lowest performance by achieving 37.5% of f-measure. This was expected

© 2005 – ongoing JATIT & LLS
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

because both features got the lowest performances independently.

Finally, as shown in Fig. 8, the highest performance was achieved when all the features have been combined in which the f-measure was 64.7%. This was followed by the combination of capital, punctuation and spell by achieving 64% of f-measure. Furthermore, the combination of punctuation, digit and spell achieved 63.1% of f-measure. Finally, the lowest performance was when capital, punctuation and digit have been combined by achieving 55.8% of f-measure.

5. CONCLUSION

This study aimed to develop a set of features that have the ability to discriminate, distinguish and describe the characteristics of splitting identifiers. For this manner, a benchmark dataset has been used in this study which contains numerous multi-word identifiers that have been extracted and splitted from various source codes. Consequentially, four features are being extracted including capital count, punctuation count, digit count and spell checker. After the features are being extracted, the data will be used to train a Naïve Bayes classifier which will accommodate a classification process to identify the required number of splitting for each identifier. To evaluate the proposed method, the features have been examined in terms of effectiveness independently and with combinations.

Experimental results shown that the best f-measure was achieved by the combination of all the features by obtaining 64.7%. This result implies the usefulness of such feature in terms of splitting identifiers.

The main limitation behind this study lies on the dataset where it tends to be relatively small data. As a future research, examining a large-scale data that may contain numerous forms of multi-word identifiers would significantly contribute toward improving the classification accuracy.

6. DISCUSSION

To provide a proper clarification of the accomplishment of this study, it is necessary to accommodate a comparison with the state of the art. Lawrie et al. [4] who have proposed a morphological approach for dividing the multi-word identifiers, achieved 62% of f-measure. In addition, Enslen et al. [3] have proposed a statistical approach for dividing the identifiers, they achieved an f-measure of 60%. Furthermore, Lawrie & Binkley [12] have proposed a dictionary-based approach and attained an f-measure of 61%. Comparing these results with the proposed method's results (i.e. 64.7% of f-measure), it is obvious that the proposed method has outperformed the state of the art. This could be due to the diversity of features used in this study, unlike the related word who concentrated on a specific type of features.

REFERENCES

[1] Dag IK Sjøberg, Jo E Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, Nils-Kristian Liborg, and Anette C Rekdal, "A survey of controlled experiments in software engineering," *Software Engineering, IEEE Transactions on,* vol. 31, pp. 733-753, 2005.

[2] Stephen W Thomas, "Mining software repositories using topic models," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 1138-1139.doi.

[3] Eric Enslen, Emily Hill, Lori Pollock, and K Vijay-Shanker, "Mining source code to automatically split identifiers for software analysis," in *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*, 2009, pp. 71-80.doi.

[4] Dawn Lawrie, Henry Feild, and David Binkley, "Quantifying identifier quality: an analysis of trends," *Empirical Software Engineering,* vol. 12, pp. 359-388, 2007.doi:10.1007/s10664-006-9032-2 http://dx.doi.org/10.1007/s10664-006-9032-2.

[5] Kunrong Chen and Václav Rajlich, "Case Study of Feature Location Using Dependence Graph," in *IWPC*, 2000, pp. 241-247.doi.

[6] Dawn Lawrie and Dave Binkley, "Expanding identifiers to normalize source code vocabulary," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, 2011, pp. 113-122.doi.

[7] Henry Feild, David Binkley, and Dawn Lawrie, "An empirical comparison of techniques for extracting concept abbreviations from identifiers," in *Proceedings of IASTED International Conference on Software Engineering and Applications (SEA'06)*, 2006.

[8] Jonathan I Maletic and Andrian Marcus, "Supporting program comprehension using semantic and structural information," in *Proceedings of the 23rd International Conference on Software Engineering*, 2001, pp. 103-112.doi.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[9] Denys Poshyvanyk, Andrian Marcus, Yubo Dong, and Andrey Sergeyev, "IRiSS-A Source Code Exploration Tool," in *ICSM (Industrial and Tool Volume)*, 2005, pp. 69-72.doi.

[10] Dayne Freitag, "Machine learning for information extraction in informal domains," *Machine learning,* vol. 39, pp. 169-202, 2000.

[11] Jin Huang, Jingjing Lu, and Charles X Ling, "Comparing naive Bayes, decision trees, and SVM with AUC and accuracy," in *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, 2003, pp. 553-556.doi.

[12] D. Lawrie and D. Binkley, "Expanding identifiers to normalize source code vocabulary," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 113-122.doi:10.1109/ICSM.2011.6080778.