Journal of Artificial Intelligence and Capsule Networks (2020)
Vol.02/ No. 03
Pages: 177-184
http://irojournals.com/aicn/
DOI: https://doi.org/10.36548/jaicn.2020.3.005

Plagiarism Detection in Programming Assignments using Machine Learning

[1]Nishesh Awale, [2]Mitesh Pandey,[3]Anish Dulal, [4]Bibek Timsina
Department of Electronics and Computer Engineering, Pulchowk Campus, Lalitpur, Nepal
[1]073bex423.nishesh@pcampus.edu.np,
[2]073bex417.mitesh@pcampus.edu.np,[3]073bex406.anish@pcampus.edu.np, [4]073bex409.bibek@pcampus.edu.np

Abstract: Plagiarism in programming assignments has been increasing these days which affects the evaluation of students. Thispaper proposes a machine learning approach for plagiarism detection of programming assignments. Different features related to source code are computed based on similarity score of n-grams, code style similarity and dead codes. Then, xgboost model is used for training and predicting whether a pair of source code are plagiarised or not. Many plagiarism techniques ignores dead codes such as unused variables and functions in their predictions tasks. But number of unused variables and functions in the source code are considered in this paper. Using our features, the model achieved an accuracy score of 94% and average f1-score of 0.905 on the test set. We also compared the result of xgboost model with support vector machines(SVM) and report that xgboost model performed better on our dataset.

Keywords: Source Code Plagiarism Detection; Xgboost; Programming Assignments Plagiarism; Source Code Features;

1. Introduction

With the increase in digital content because of the Internet, information has been easily available for everyone. The sharing of assignments, whether text based or programming assignments, has been easy for students. Due to this, plagiarism has been increasing in the academic sector. Plagiarism will not result in the fair evaluation of the students as well as it hampers the learning of the student. The manual checking for plagiarism in assignments is tedious and time consuming.

Many tools have been made for checking plagiarism in programming assignments. The plagiarism detection task can be considered as a classification problem. The two assignments can be taken as inputs and the system will determine if the pair of assignments are plagiarised or not. Early methods for plagiarism detection in programming assignments are based on n-gram techniques which create fingerprints of the assignments to measure similarity between them. They do not take code style similarity such as braces, comments and whitespace similarity into consideration. Some machine learning techniques have been pro- posed based on traditional learning algorithms such as k-nearest neighbors, naive Bayes, etc. Also, with the success of deep learning in computer vision and natural language processing, people are trying to use deep learning for source code plagiarism detection as well. One approach uses recurrent neural network to extract the features from hidden layers and train a classifier on those features.

Our approach computes different features from the programming assignments that are helpful for de- tecting plagiarism. We compute different features such as similarity score, number of unused variables and functions, etc from the source code. Then, we train an xgboost learning algorithm on these features and com- pare its result with Support Vector Machine(SVM). We believe that our features will detect plagiarism even if the assignments are highly obfuscated. With our proposed features, our model obtained 94% accuracy on the test set. More discussion on the result is given in section 4.

2. Related Works

Different attempts had been made to detect plagiarism in programming assignments submitted by students. Jplag [1] is one of the earliest and very powerful work we found in the field of testing plagiarism of source code programs. It

ISSN: 2582-2012 (online)
Submitted: 24.05.2020
Accepted: 14.07.2020
Published: 21.07.2020

used a specified language parser or scanner depending upon the programming language in which source code was written. The initial parser pre-processes the source code changing it to appropriate tokens and tokens are chosen in such a manner that characterize the essence of program(which is difficult to change by a plagiarist). For instance, whitespace and comments should never produce a token, since they are the most obvious points of attack. Various run time optimizations are also discussed well in the paper. MOSS(Measure of Software Similarity) [2] is also one of the popular tool to detect plagiarism for digital content. It is a token based method which selects fingerprints from hashes of k-grams. It uses winnowing approach in which a window is defined and from each window, a minimum hash value is selected as a fingerprint. Then, the fingerprints of different documents are compared for plagiarism detection. But it ignores the similarity of braces, comments and number of unused variables and functions.

In [3], the authors have defined different levels of plagiarism from level 0 to 6 and used machine learning techniques to detect plagiarism. They calculate different source code metrics such as number of words in one source code lines, number of characters in one source code lines, number of underscore characters used in identifiers, etc. They calculate similar 9 features and used naive Bayes classifier, k-nearest neighbor and AdaBoost algorithm for classification. It also ignores the dead codes such as unused variables and functions in the source code. In [4], different high level features such as number of lines, characters, input and outputs of the programs are computed. Then, weighted average of all features are used to compute similarity score. Another approach in [5] detect plagiarism at the function level. After preprocessing, source codes are converted to different functions and similarity score is computed between different pairs of functions.

With the rise in deep learning, people have tried to use deep learning for source code plagiarism detection as well. In [6], char-RNN model was pre-trained on Linux kernel source code and the last layer of the RNN model was used to extract deep features. Then, a classifier was trained on those features to detect plagiarism. According to their result, deep learning approaches to plagiarism detection also looks very promising.

In [7], the authors have used plagiarism detection in the submission record of Online Judge System. They calculate various features such as level of concentration of plagiarism targets, difficulty level of the problem, ranking of the student, etc. and irrelevant features were filtered out using information gain. They also used various performance metrics such as macro F1 score, AUC and ROC to select the learning model.

3. Proposed Work

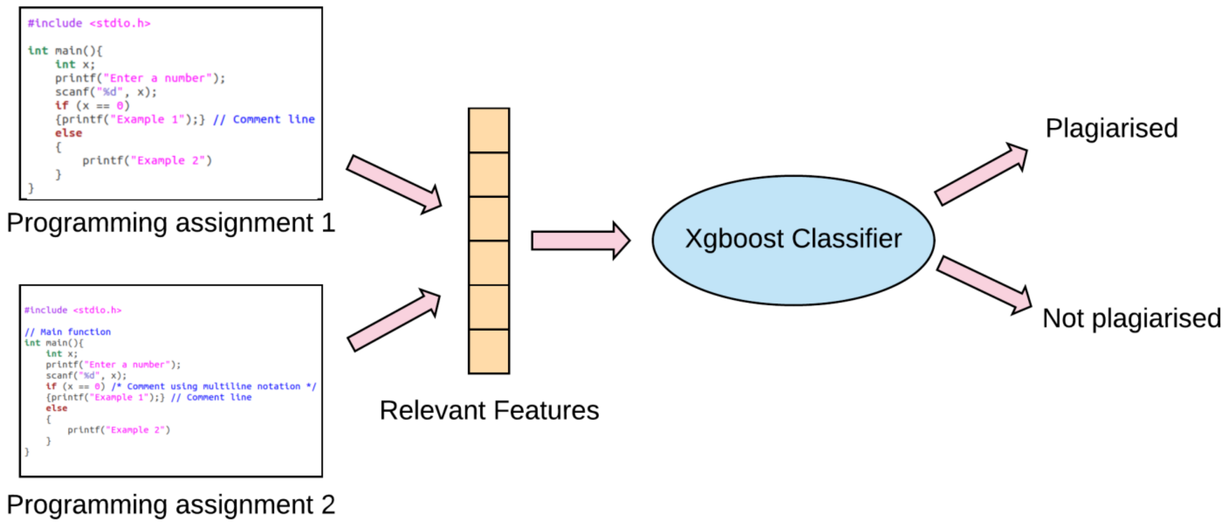
Our approach uses machine learning techniques for plagiarism detection on programming assignments. First, we extract features from the C/C++ source code and then, train an xgboost algorithm on our dataset for plagiarism detection. Our model predicts whether a pair of programming assignments is plagiarised or not. The detail explanation of our approach is shown in figure 1.

3.1 Dataset

We used the programming assignments submitted by students during two introductory programming courses at University of Sarajevo which was found online on IEEEDataPort[8]. From the collection of C/C++ assignment submissions, we formed different combinations of plagiarised and non-plagiarised assignment pairs according to the given ground truth. The plagiarised class was labeled as 1 and non-plagiarised class was labeled as 0. As the ratio of plagiarised and non-plagiarised assignment pairs was very small, so we sample only a portion of non-plagiarised pairs and select all the plagiarised pairs. As a result, we had a total assignment pairs of 5884 out of which 1262 pairs were plagiarised while 4622 were non-plagiarised.

ISSN: 2582-2012 (online)
Submitted: 24.05.2020
Accepted: 14.07.2020
Published: 21.07.2020

Journal of Artificial Intelligence and Capsule Networks (2020)
Vol.02/ No. 03
Pages: 177-184
http://irojournals.com/aicn/
DOI: https://doi.org/10.36548/jaicn.2020.3.005



```
#include <stdio.h>

int main(){
    int x;
    printf("Enter a number");
    scanf("%d", x);
    if (x == 0)
    {printf("Example 1");} // Comment line
    else
    {
        printf("Example 2")
    }
}
```

```
#include <stdio.h>

// Main function
int main(){
    int x;
    printf("Enter a number");
    scanf("%d", x);
    if (x == 0) /* Comment using multiline notation */
    {printf("Example 1");} // Comment line
    else
    {
        printf("Example 2")
    }
}
```

Figure 1: Our proposed approach. First, we extract pairwise features from source code. Then, we train an xgboost model for classification

3.2 Computing Relevant Features

After preparing the dataset, we extracted various features from the programming assignment. Various fea- tures which were used for plagiarism detection are described below. Similarity score, code style similarity and categorical value features were computed based on the methodology proposed by Huang et al.[7] with some modifications.

3.2.1 Similarity Score

To get an idea of how similar the given two submissions are, we modify Karp-Rabin string matching algo- rithm to get a similarity score as output. The similarity score was calculated as follows:

1. First, all the variable names, function names and string inside print statements were converted to a fixed character so that even there is change in variable name, function name and the string inside print statements between two assignments, they are treated as plagiarised.

2. The line breaks, tabs and extra spaces were removed as well as the whole program was converted to a single string, S.

3. After converting to a single string S, the k-gram collection from string S was generated. A k-gram consists of the sub-string of length k. For our experiment, we used value of k as 20.

4. For making comparison of k-grams faster, we hash all k-grams to integer value. The hash value of first k-gram($c_1 ... c_k$) was calculated based on the constant b as,

$$H(c_1 ... c_k)=(c1 * b_{k-1} + c2 * b_{k-2} + ... + c_{k-1} * b + c_k) * b$$

To compute the hash values of consecutive k-grams, we used rolling hash function for faster computation instead of equation 1. We computed the consecutive hash values as follows,

$$H(c_2 ... c_{k+1}) = ((H(c_1 ... c_k) - c_1 * b_k) + c_{k+1}) * b$$

We can see that a single string S has n-k+1 hash values.

ISSN: 2582-2012 (online)
Submitted: 24.05.2020
Accepted: 14.07.2020
Published: 21.07.2020

5. After converting an assignment pair to string S_1 and S_2 then, to the collection of hash values H_1 and H_2, we used Jaccard similarity to compute a similarity score as,

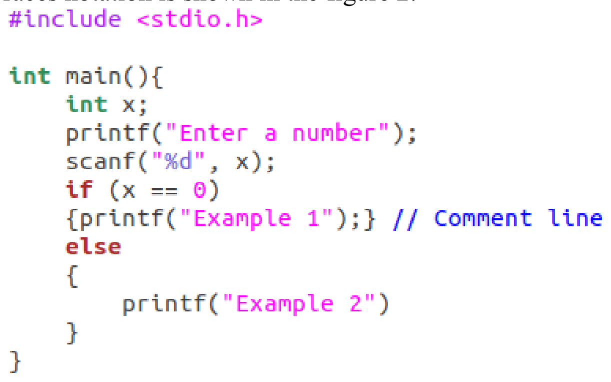
$$\text{Similarity Score} = \frac{(H1 \cap H2)}{(H1 \cup H2)}$$

3.2.2 Code Style Similarity

Different IDEs produce different style of codes i.e. the styling of braces and whitespaces is different among various IDEs. So, plagiarism can also be detected on the basis of style of code written. Braces similarity, comment similarity and whitespace similarity were computed which are described below:

A. Braces Similarity

We classify the braces into four types. The first one is when the brace is at the far left of the line and some code is written after the brace. The second one represents the condition when the brace is at the far right of the line after some code is written on that line. The third one indicates the brace is in the middle of the line. The fourth one indicates the brace is on the separate line and no code is written on that line. These four conditions are represented by 1, 2, 3 and 4 respectively. An example of braces notation is shown in the figure 2.



```
#include <stdio.h>

int main(){
    int x;
    printf("Enter a number");
    scanf("%d", x);
    if (x == 0)
    {printf("Example 1");} // Comment line
    else
    {
        printf("Example 2")
    }
}
```

Figure 2: An example of braces notation. Braces notation of above code is {2{1}3{4}4}4.

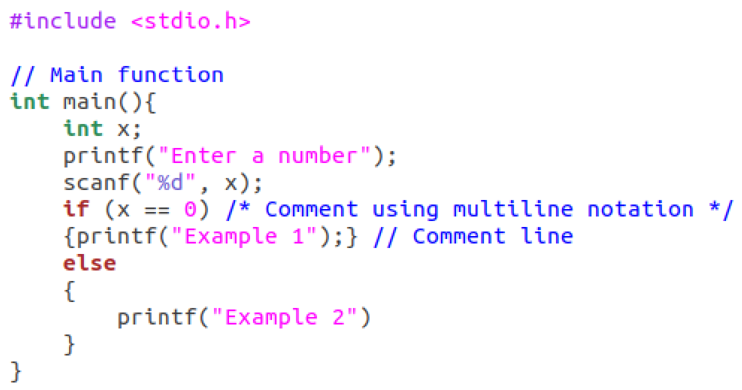
We compute the braces notation of both programming assignments in the pair. Then, the longest common subsequence in the braces notations is computed. If LCS be the length of the longest common subsequence and L_1, L_2 be the length of the braces notation of two assignmentsrespectively then, the braces similarity is calculated as,

$$\text{Braces Similarity (BS)} = \frac{2*LCS}{L1*L2}$$

B. Comment Similarity

Different people have different style of writing comment. Some write the comment in the same line after the code while some people uses new line to write comments. Some people uses multi-line comment notation to write comment while some people uses single-line comment notation. Comments are classified into 3 types. The first one represents comment on a new line. The second one is when comments are written in the same line after writing the code. The thirdone represents the multi-line comment notation. They are represented by S_1,S_2 and M_3 respectively. An example of comment notation is shown in figure 3. For the comment similarity also, we calculate the longest common subsequence between the pair of source codes and use equation 4 to calculate comment similarity.

ISSN: 2582-2012 (online)
Submitted: 24.05.2020
Accepted: 14.07.2020
Published: 21.07.2020



```
#include <stdio.h>

// Main function
int main(){
    int x;
    printf("Enter a number");
    scanf("%d", x);
    if (x == 0) /* Comment using multiline notation */
    {printf("Example 1");} // Comment line
    else
    {
        printf("Example 2")
    }
}
```

Figure 3: An example of comment notation. Comment notation of above code is S1M3S2.

C. Spaces and Newline Similarity

In source code practices, spaces can be in two forms: tabs and whitespaces. Some prefer whitespaces and some prefer tabs. So this practice differs from person to person and can be used as an important feature to check similarity between two source codes. In addition, newlines are also arbitrarily used by coders as per their desire. Some prefer wide clear gaps while some try to make the lines as much less as possible. So,number of newlines implementation could also be an important feature to check the similarity between two source codes. Edit distance(ED) is calculated as the end parameter which would summarize the relative position of tabs,spaces and newlines in two source code program being compared.

$$\text{ED} = \text{tabsDistance} + \text{spaceDistance} + \text{newlineDistance}$$
$$\text{total} = \text{totalTabs} + \text{totalSpaces} + \text{totalNewlines}$$
$$\text{SNS} = 1 - \frac{\text{ED}}{\text{total}}$$

After computing the braces similarity(BS), comment similarity(CS) as well as Spaces and Newline Similarity(SNS), we computed the code style similarity as follows:

$$\text{Code Style Similarity} =) \frac{(\text{BS} + \text{CS} + \text{SNS}}{3}$$

3.2.3 Categorical Value According to Similarity Score

A categorical value is also considered as a feature in which six different classes are defined according to the difference between the similarity score(SS) and the similarity threshold value(ST). Similarity threshold value can be changed according to the nature of the problem. We used the threshold value of 0.4. Distance between similarity score and similarity threshold value is calculated and normalized as,

$$\text{Distance} = \frac{|\text{SS} - \text{ST}|}{\text{ST}}$$

Then, we determine the category using the following algorithm.

```
If (SS < ST), then:
If 0.5 < Distance ≤ 1, then category = 1;
If 0.2 < Distance ≤ 0.5, then category = 2;
If Distance ≤ 0.2, then category = 3;
Else:
If Distance ≤ 0.2, then category = 4;
If 0.2 < Distance ≤ 0.5, then category = 5;
If 0.5 < Distance ≤ 1, then category = 6.
```

3.2.4 Number of Common Lines

The number of common lines in an assignment pair is also a measure of the plagiarism detection between them. Each line in a file are stored in a set so that duplicate lines in the same file are removed. Then, the lines which are common in both files are counted. The blank lines in both files are ignored while calculating this feature.

ISSN: 2582-2012 (online)
Submitted: 24.05.2020
Accepted: 14.07.2020
Published: 21.07.2020

3.2.5 Number of Unused Variables

While plagiarising the source codes, students may add dead codes such as variables name which are unused. So, we used cppcheck which is a tool for static C/C++ code analysis. Using this tool, we count the number of unused variables in both files. The total number of unused variables in both files are added to give a single value which is used as a feature for the assignment pair.

3.2.6 Number of Unused Functions

Similar to unused variables, students may add functions which are never called in the main program. To detect this type of plagiarism, we used cppcheck to count the number of unused functions. Similar to unused variables, the total number of unused functions in both files are added to give a single value which is used as a feature for the assignment pair.

3.3 Training a Classifier

After extracting the above six features from the source code pairs, supervised learning approach was used to detect plagiarism. We trained an xgboost algorithm on these features and the labels. Xgboost is a decision- tree based ensemble machine learning algorithm. It uses a gradient boosting framework. The whole dataset was divided into 70% training set and 30% test set. Then, the model was trained and different metrics such as accuracy and f1-score was used to evaluate the model.

4. Results and Discussion

We used Google Colaboratory for training our learning model. After training the xgboost model, we got an accuracy score of 94% on the test set. As our data was imbalanced in the ratio nearly equal to 1:4, the model which predicted all assignment pairs as non-plagiarised would have got an accuracy of 75%. But we can see that using our source code metrics, the accuracy of the model has greatly increased.

We also used per label f1-score as the metric to evaluate the model which summarized the precision and recall of our model. The per label precision, recall and f1-score of xgboost model has been shown in the table 1.

Class Label	Precision	Recall	F1-score	Accuracy
Not plagiarised	0.95	0.97	0.96	0.94
Plagiarised	0.89	0.82	0.85	

Table 1: Result of Xgboost Model

For the comparison of xgboost model with other classifier, we also trained support vector machine (SVM) under similar conditions. From the tables 1 and 2, we can see that xgboost model performs well and has higher accuracy and f1-score than SVM.

Class Label	Precision	Recall	F1-score	Accuracy
Not plagiarised	0.91	0.97	0.94	0.90
Plagiarised	0.87	0.66	0.75	

Table 2: Result of SVM Model

ISSN: 2582-2012 (online)
Submitted: 24.05.2020
Accepted: 14.07.2020
Published: 21.07.2020

Journal of Artificial Intelligence and Capsule Networks (2020)
Vol.02/ No. 03
Pages: 177-184
http://irojournals.com/aicn/
DOI: https://doi.org/10.36548/jaicn.2020.3.005

The result of true positive, true negative, false positive and false negative of xgboost model has also been summarized by the confusion matrix shown in figure 4. The result in the confusion matrix has been normalized by the total number of true labels of each class. We can see that 97% of the truly non-plagiarised assignment pairs were classified correctly by the model. Similarly, 81% of the truly plagiarised assignment pairs were classified correctly. 18% of truly plagiarised assignments were predicted as non-plagiarised by our model. It might have occurred because we may have missed some features of the assignment pairs to take into consideration.

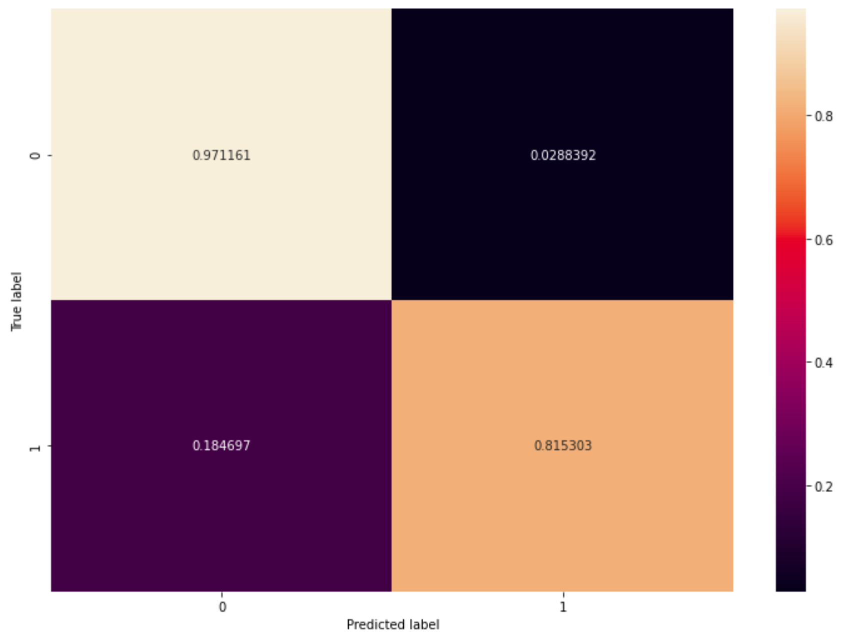


Figure 4: Confusion matrix using Xgboost model

5. Conclusion

In this paper, we have discussed a machine learning approach for plagiarism detection in programming assignments. Different features related to programming assignment pair were computed and xgboost model was used for classification. Based on the result of our model on test set, machine learning approach for plagiarism detection looks very promising. Although there are various deep learning methods for text based classification, but we found they are not particularly suitable for testing of source code plagiarism. We are still thriving to improve our algorithm by incorporating compiler based features(specific to programming languages) in our classification model.

Acknowledgement

We are very grateful to Prof. Dr. Shashidhar Ram Joshi for supervising our project and helping us in every possible manner to achieve the targeted goals. We also like to convey our deepest gratitude to Prof. Dr Subarna Shakya for guiding us in publishing this paper. He helped us draft and critically revise the paper in keeping with important intellectual content and mitigating our errors.

ISSN: 2582-2012 (online)
Submitted: 24.05.2020
Accepted: 14.07.2020
Published: 21.07.2020

Journal of Artificial Intelligence and Capsule Networks (2020)
Vol.02/ No. 03
Pages: 177-184
http://irojournals.com/aicn/
DOI: https://doi.org/10.36548/jaicn.2020.3.005

References

[1] Prechelt,L.,Malpohl,G., &Phillipsen,M. (2000). Jplag:Finding plagiarisms among a set of programs, *Technical Report* 2000-1,FakultätfürInformatik , Universität Karlsruhe.

[2] Schleimer, S., & Wilkerson, D. S., Aiken, A. (2003). Winnowing: local algorithms for document fingerprinting, *Acmsigmod*, 76-85.

[3] Bandara, U., &Wijayarathna, G. (2011). A Machine Learning Based Tool for Source Code Plagiarism Detection, *International Journal of Machine Learning and Computing*, 1(04), 337-343.

[4] Narayanan, S., & Simi, S. (2012). Source code plagiarism detection and performance analysis using fingerprint based distance measure method, *7th International Conference on Computer Science & Education*, 1065-1068.

[5] Agrawal, M., & Sharma, D. K. (2016). A novel method to find out the similarity between source codes, *IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engi- neering*, 339-343.
[6] Yasaswi, J., Purini, S., &Jawahar, C. V. (2017). Plagiarism Detection in Programming Assignments Using Deep Features, *4th IAPR Asian Conference on Pattern Recognition*, 652-657.

[7] Huang, Q., Fang, G., & Jiang, K. (2019). An Approach of Suspected Code Plagiarism Detection Based on XGBoost Incremental Learning, *International Conference on Computer, Network, Communication and Information Systems*, 88, 269-276.

[8] Ljubovic, V. (2020). Programming Homework Dataset for Plagiarism Detection.*IEEE Dataport*. http://dx.doi.org/10.21227/71fw-ss32

Authors Biography

The paper presented above is a part of major projectdone by the 4th year students of Department of Electronics and Computer Engineering at Institute of Engineering, Pulchowk Campus. The authors have participated in many project competitions as a team and their areas of interests covers Artificial Intelligence, Machine Learning, Block chain and IOT technologies. They were the winner of 2019 Rising Student ICT Award for project named "Raktadaan". Their projects mainly focus on solving the current social problems using feasible technical solutions.

Nishesh Awale

Mitesh Pandey

Anish Dulal

Bibek Timsina

ISSN: 2582-2012 (online)
Submitted: 24.05.2020
Accepted: 14.07.2020
Published: 21.07.2020