Predicting the algorithmic time complexity of single parametric algorithms using multiclass classification with gradient boosted trees

Deepak Kumar Sharma , Sumit Vohra , Tarun Gupta and Vipul Goyal

Division of Information Technology, Netaji Subhas Institute of Technology, University of Delhi, New Delhi, India
Email: dk.sharma1982@yahoo.com, sumitvohra90@gmail.com, tarungupta1729@gmail.com, vipulgoyal794@gmail.com

***Abstract*—The amount of code written has increased significantly in recent years and it has become one of the major tasks to judge the time-complexities of these codes. Multi-Class classification using machine learning enables us to categorize these algorithms into classes with the help of machine learning tools like gradient boosted trees. It also increases the accuracy of predicting the asymptotic-time complexities of the algorithms, thereby considerably reducing the manual effort required to do this task, at the same time increasing the accuracies of prediction. A novel concept of predicting time complexity using gradient boosted trees in a supervised manner is introduced in this paper.**

Keywords—Tensorflow, xgboost, Machine Learning, Pandas, Scikit, Pyplot, Matplotlib, Python

I. INTRODUCTION

The task of judging the time-complexity of algorithms has remained a hectic task for a long period of time. It takes significant amount of effort to judge the complexity of an algorithm. Various manual methods are used till now to get calculate the time complexities such as Master Method [11] , using control flow graph [10] , but all of them remain tedious to work with and owing to their manual nature, have limitations and are prone to error. Most of the algorithms known tend to show significant differences in behaviour in terms of execution time according to varied input sizes by deviating from their expected curves. This happens mostly due to dominance of constants and other factors. Apart from constants, the time - taken by an algorithm also depends on the machine on which it runs, operating system on that machine and the priority of that process. Owing to all these factors normal unitary method which tries to directly map time with respect to the input size(N) by calculating the ratios of time with respect to the input size doesn't work and often gives wrong results. A comparison to further strengthen this proposition is discussed later in the paper. To counter this problem and devise a technique which could effectively predict the time-complexity of an algorithm a multi-class classification using gradient boosted trees has been used in a supervised way to train the proposed model on different categories of algorithms. Since the practical complexities of algorithms are fairly limited therefore 7 categories have been used which are prominent to classify the algorithms namely O(N) , O(NLogN) , O(N^2) , O(LogN) , O(NSqrt(N)), O(N^3), O(Sqrt(N)).

II. RELATED WORK

Predicting the time complexity of any algorithm, in general, using automated methods is a relatively new field of research. Most of the research in this field is concentrated on analysis of sorting and searching algorithms. The work in [13] provides a detailed mathematical study of quicksort algorithm and suggests an improvement over the traditional algorithm. In [14], a comparative study and analysis of some well known traditional and proposed searching techniques. It gives a detailed contrast between merits and demerits of traditional search algorithms such as linear search, binary search and jump search, discusses the methodology and scope of newly developed algorithms such as Bi-linear search, multiple solution vector approach, nearest neighbour search algorithm and tabulates their relative advantages and disadvantages. Despite being informative, it is quite limited in its scope as its main focus is domain of searching and sorting.

A few works have applied significant efforts towards estimating the time complexity of any general algorithm. In paper [9], Tomaz Dobravec has proposed a framework, ALGator, which tries to predict the time complexity of an algorithm by counting its corresponding java bytecode instructions. Moreover, a formula has also been derived which predicts the time complexity of the algorithm with least possible error. Although the paper presents a novel idea, it is severely restricted due to the selected algorithm's nature and its independence from the input data. Also, the formula derivation is time consuming as the constants in the formula tend to differ according to the algorithm chosen.

Another method in [11] has been proposed which use control flow graph to find the time complexity of an algorithm. But the method tedious as the entire structure, design and code of algorithm has to be studied.Also, this method is applicable only to small programs or segments of a larger program.

The rest of the paper is structured as follows : Section III explains the proposed solution which is divided further into 2 parts: the 1st part elaborates the procedure of generating the dataset and the 2nd part deals with training the model using the machine learning classifier. Section IV illustrates the results obtained using the proposed method and section V defines the conclusion and scope for future work.

III. PROPOSED SOLUTION

To counter the problem of difficulties and high inaccuracies in calculating the time complexities of algorithm using the above

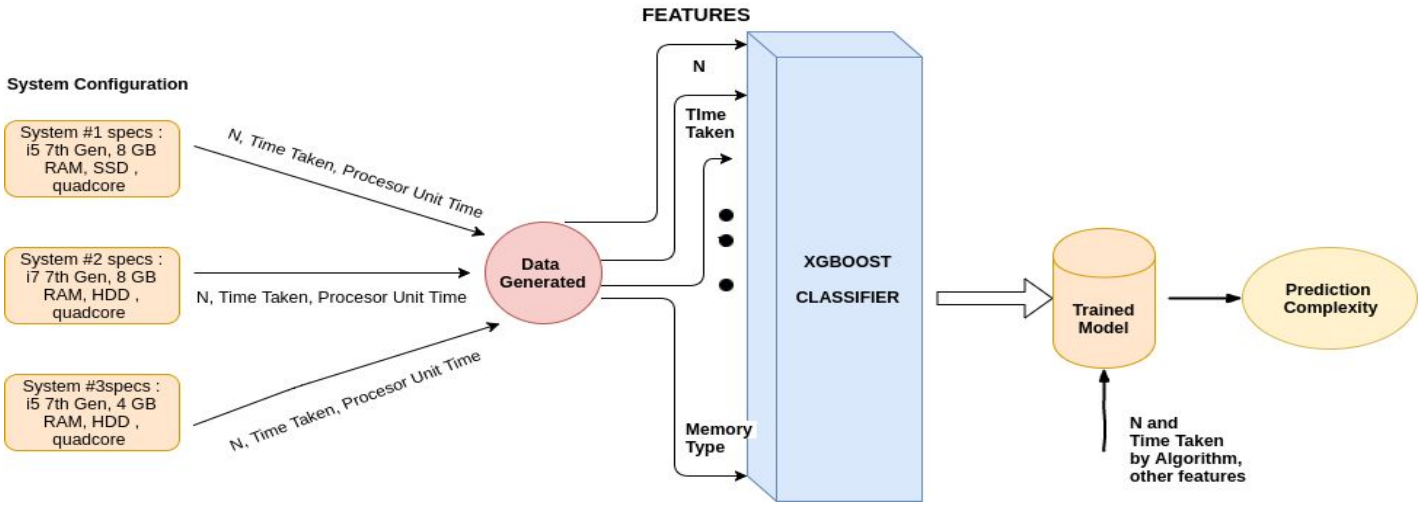


fig1: Block diagram for Proposed solution to predict time complexity of given algorithm. Data collected on three computers with distinct configurations is fed into the xgboost classifier which is used for training the model.

mentioned methods, to reduce difficulty of manually evaluating time complexities and to improve upon previous automated existing work, we propose a novel concept of supervised learning of time-complexities of algorithms using gradient boost trees for most frequently occurring complexities. In this section following understated method was employed to collect the appropriate data and train the model. A block diagram of proposed algorithm is shown in fig1.

A. GENERATING THE DATASET

A dataset was generating for the purpose of training the classifier. Several factors such as hardware configuration of the system, process priority, clock speed, input size were taken into consideration while generating the dataset. It consisted of 3 comma separated values: the input size(N), the execution time of the algorithm on the given input size and the clock speed of the system at the time of execution. A number of algorithms having different time complexities were considered in order to make the dataset large and varied.

A.1 *HOW DATA WAS GENERATED*

- A time complexity was selected and algorithms mentioned in *table 2* pertaining to the targeted complexity were used.
- Algorithms were written in C++ along with additional code in the driver program for noting down the input size, execution time and the clock speed of the machine.
- The algorithm was run for predefined iterations with random input size to collect sufficient data for each complexity. The priority of the process was set as "Very High" to have minimum preemption.
- This process was repeated for various time complexities and for various algorithms underlying the selected complexity.
- Driver code along with the main program can be found over here [17]

A.2. *DATA GENERATED*

Following table shows the amount of data generated for each class of algorithm, it was required as an initial step to feed our xgboost classifier with this data for training.

Time Complexity	No.Of Algorithms	Iterations Per Algorithm	Number Of Output
O(N)	6	5000	30000*3
O(NLOGN)	6	5000	30000*3
O(N^2)	5	5000	25000*3
O(LOGN)	6	5000	30000*3
O(N*SQRT(N))	3	5000	15000*3
O(N^3)	4	500	2000*3
O(SQRT(N))	3	5000	15000*3
TOTAL DATA ROWS GENERATED			**441000**

Table 1 : Time Complexity vs number of data rows collected for that Time-complexity.

A.3 *AMOUNT OF DATA GENERATED AND ALGORITHMS USED*

- For every complexity, 3-6 algorithms were used and for each algorithm three output files were created by running

them on computers with different configurations as mentioned in *table 1*.

- Each output file contained predefined output rows with randomly selected values of N, corresponding run time and per unit operation time.
- For all algorithms except O(N^3), 15000 test cases were generated.
- Table 2 states the Complexity of the code and corresponding algorithms used for generating dataset used for training and testing purposes.
- The generated output files could be found over here [18]

A.4 *FEATURES GENERATED*

Following features mentioned in table 3 were used for training the proposed model and were extracted from running the code and using the hardware configuration of user's computer such as the type of processor of the machine on which the code was run. [19] was used to generate a file consisting of Comma Separated Values(CSV) of features from the generated dataset.

- **N** - The input size given to the algorithm
- **Time_taken** - For a given N, it is the time taken by an algorithm to complete successfully
- **Per_Unit_Processor_Time** - This is the average time taken by the CPU to process unit instruction in an operating system. It is computed at the start of the program to determine the average speed of the pc at that time.
- **RAM** - RAM of the computer used
- **Processor_Speed** - The clock speed of the processor used
- **Series** - The series of the processor used
- **Cores** - The number of cores in the processor
- **Per_Core_Processor_Speed** - The rated clock speed of each individual core
- **Memory_Type** - Secondary storage type of the machine

CLASS	ALGORITHMS	INPUT SIZE
LOG(N)	Binary Exponentiation, Binary Search, Heapify, Range Minimum Query, Lowest Common Ancestor	UPTO 1e9
N	Knuth Morris Pratt, Linear Search, Manacher, DFS, Diameter of tree, Maximum of minimum	UPTO 1e8
NLOG(N)	Binary Index Tree, Heapsort, Mergesort, Quicksort, Segment Tree, Sparse Table	UPTO 1e6
N*SQRT(N)	MO Algorithm, Prime Factor, Basic	UPTO 1e9
N^2	Bubble Sort, Cycle Sort, Insertion Sort, Selection Sort, Longest Increasing Subsequence	UPTO 1e4
N^3	Basic, Floyd Warshall, Matrix Chain, Matrix Multiply	UPTO 1e2
SQRT(N)	Basic, Prime Factor, Lowest Common Ancestor	UPTO 1e9

Table 2: *Time Complexity of the code and corresponding algorithms used for generating dataset*

System Properties	
Process Priority	Very High
Programming Language	C++
Operating System	Ubuntu
IDE	Codeblocks
RAM	4GB, 8GB
PROCESSOR	i5, i7
CORES	4
MEMORY TYPE	SDD, HDD
SERIES	7200, 7700
N	UPTO 1e6

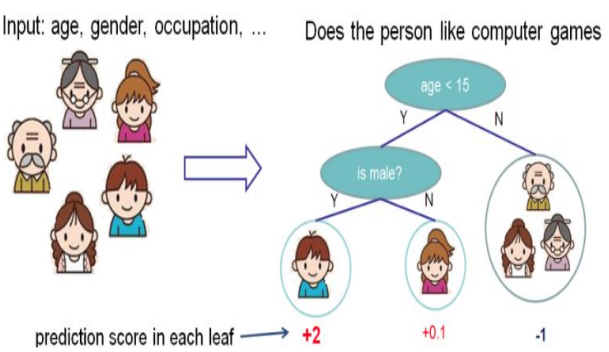
Table 3: System properties which were assumed and maintained to generate the dataset.

B. TRAINING MODEL WITH XGBOOST

xgboost is abbreviation for "Extreme Gradient Boosting"[15][20]. xgboost is based on model of tree-ensembles. Tree ensemble is a set of Classification and Regression Trees(CART).

B.1 W*ORKING OF XGBOOST*

First an example is illustrated to demonstrate working of xgboost and then a mathematical formulation of the same is described. ·



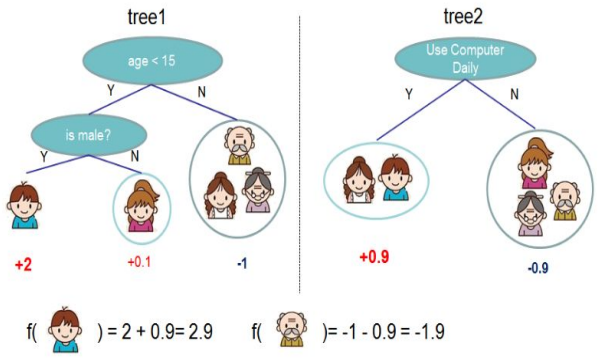


Fig2 : Classification of family members done using CART[15]

Members of the family are classified into different leaves and a score is assigned to each one of them. It is better than using conventional decision trees as they store just the decision values whereas in case of CART like xgboost a better representation in the form of overall score is stored which is used to make unified classification decisions later.

Multiple such trees in the form of tree ensemble are used which aggregates the prediction of multiple trees together. This model can be mathematically represented as follows.[16]

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F \quad \text{.....(1)}$$

where y_i is the prediction score, K is the number of trees, f_i is one of the function representing a particular CART and $\mathcal{F}$ is set of all possible CARTs. The final objective function for the ensemble becomes:

$$obj(\theta) = \sum_{i}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k) \quad \text{..... (2)}$$

Here, the first part of the equation represents the training loss function and second part represents the regularisation term.

To train such tree structure, additive training is used where in a single tree is added one at a time to previous learned structure.

$$\hat{y}_i^{(0)} = 0 \quad \text{.....(3)}$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \quad \text{.....(4)}$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \quad \text{..... (5)}$$

...

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \quad \text{..... (6)}$$

The prediction values at each step of training is as follows:

so the final objective function takes the form :

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \quad \text{.....(7)}$$

$$= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant \quad \text{.....(8)}$$

For training, the following function is broken down using taylor expansion.

$$obj^{(t)} = \sum_{i=1}^{n} [l(y_i, y_i^{(t-1)}) + g_i f_t(x_i) + \tfrac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) + constant$$

..... (9)

where g and h are represented as follows :

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \quad \text{..... (10)}$$

$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) \quad \text{..... (11)}$$

B.2 *CALCULATING THE REGULARISATION CONSTANT*

To limit overfitting, regularisation or restricting the complexity of the model is an essential step. To define complexity of the tree $\Omega(f)$, *f(x)* is represented as:

$$f_t(x) = w_{q(x)}, w \in R^T, q : R^d \rightarrow \{1, 2, \cdots, T\}. \quad \text{..... (12)}$$

where *w* is the vector of the score on leaf, q is a function assigning each data point to the corresponding leaf, and *T is* the number of leaves. In xgboost, regularization is defined as :

$$\Omega(f) = \gamma T + \tfrac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \quad \text{..... (13)}$$

B.3 *LEARNING THE TREE STRUCTURE*

So the complete objective function can be reformed as :

$$obj^{(t)} \approx \sum_{i=1}^{n} [g_i w_{q(x_i)} + \tfrac{1}{2} h_i w^2_{q(x_i)}] + \gamma T + \tfrac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \quad \text{..... (14)}$$

The above described objective function is then learned by calculating the structure score as described in official documentation of xgboost [16] .

B.4 TRAINING THE XGBOOST CLASSIFIER

Features mentioned above were extracted from the data generated using the above mentioned method and then fed into an xgboost classifier (extreme gradient boosted classifier) . The parameters were then tuned with the following configuration: *Learning rate* was set at *0.03, number of estimators* were set to be *200, max depth* (maximum depth of the constructed tree) was kept at *7, min child weight* (the minimum sum of the weights of all observations required in the child node of the tree) of *0.7* was used and *train to validation ratio* was kept at *0.8. Gamma* (A node gets split only when the resulting split results in reduction of loss greater thn gamma) was kept at *0*. The total data generated was randomly split into *0.8* into training and testing data. The model was then trained using xgboost, *importance bar-graph* [fig2] was generated to determine the relative weightage of features used to train the model. [19] was used to train the model.

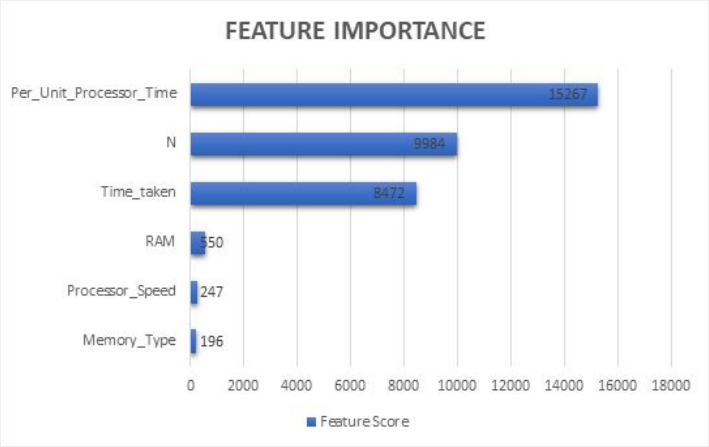


Fig3: *Feature Importance Graph generated using xgboost*

IV. RESULTS

To graphical represent the results of our algorithm, confusion matrices are plotted. A Confusion matrix compares the predicted results/labels with actual results/labels. True label is represented on the y axis and predicted label on x axis with the entries depicting the number of instances of dataset on which the model was evaluated.

Train and test accuracies can be calculated with the help of confusion matrices using the under mentioned formula :

$$Accuracy = \frac{\sum_{i=1}^{d} conf_{ii}}{\sum_{i=1}^{d}\sum_{j=1}^{d} conf_{ij}} \qquad \text{.....(15)}$$

where *conf* represents the confusion matrix and *d* is the dimension of the confusion matrix.

Four confusion matrices were plotted which represent the following cases. [fig3] shows the confusion matrix of training - dataset using the proposed method, [fig4] shows the confusion matrix for test-dataset using the proposed method, [fig5] and [fig6] show confusion matrices for training and test dataset using unitary based method.

A training accuracy of *0.998895848774* and test accuracy of *0.999030289719* were obtained using the train [fig4] and test confusion matrices [fig5].

This was a big improvement to the unitary method which tried to find nearest complexity based on time and N as described here [19] and had accuracy of *0.616300732792* [fig5] on training dataset and *0.617197244615* [fig6] on test dataset.

It also had a major advantages to previously mentioned theoretical approaches [9][11][13] which required a lot of manual labour and couldn't be generalised for most algorithms.

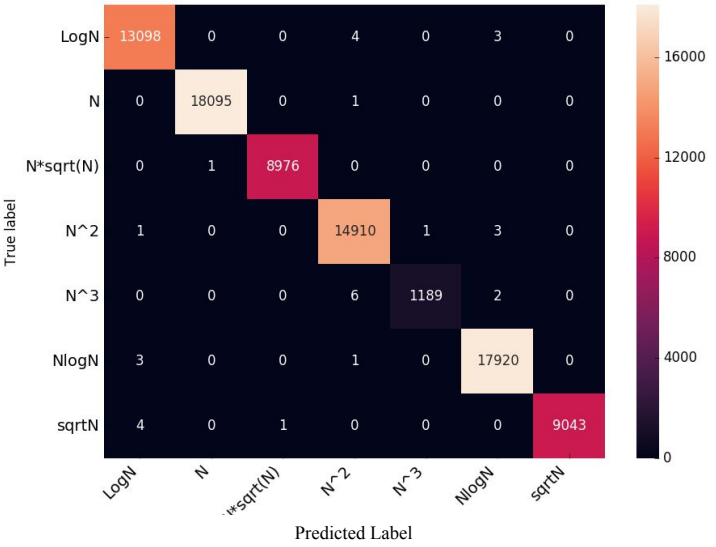


fig4 : confusion matrix for predicted time-complexity classes for train-data

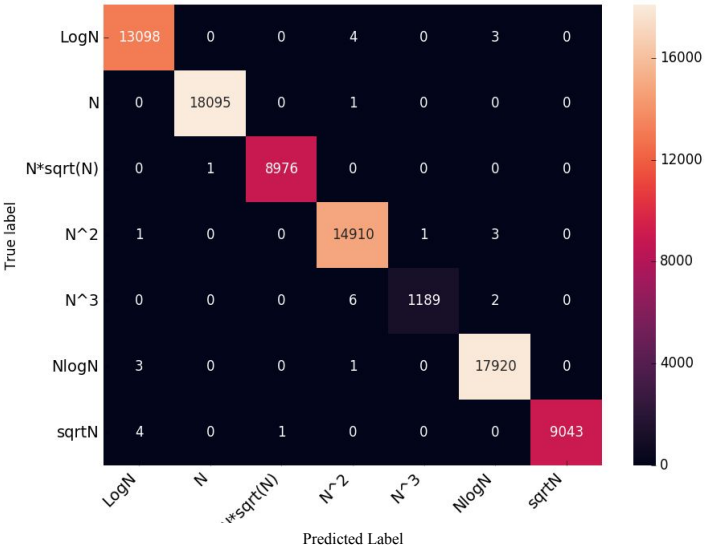


fig5 : confusion matrix for predicted time-complexity classes for test-data

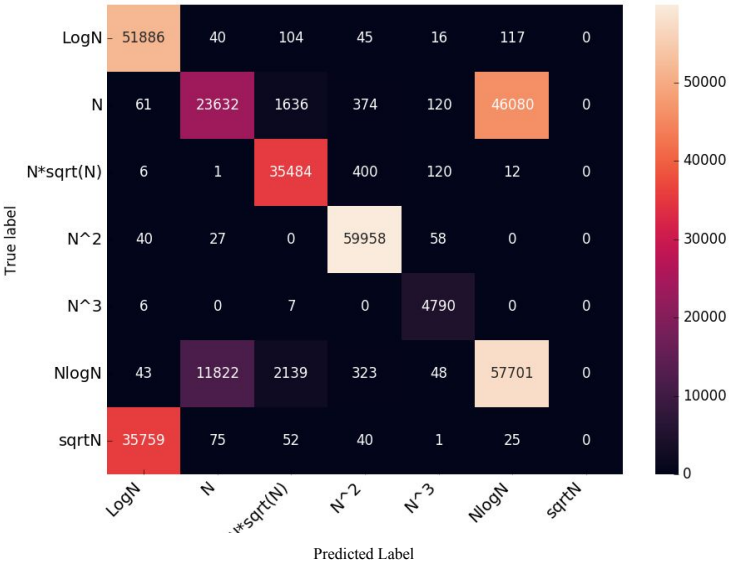


fig6 : confusion matrix for predicted time-complexity classes for brute train-data

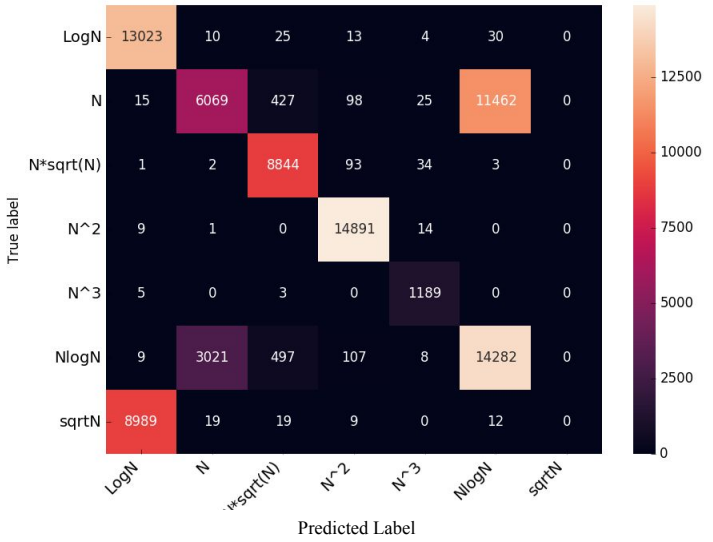


fig7 : confusion matrix for predicted time-complexity classes for brute test-data

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed a solution using multiclass classification to address the problem of predicting the time complexity of algorithms. We first illustrated the working of the whole solution which was further explained in 2 parts. In the 1st part, we explained the process of generating the dataset required for training using the xgboost classifier and in the 2nd part, we explained the working of the xgboost classifier.

Further, we highlighted the results obtained by using the proposed method. We showed that using this method, a training and testing accuracy of about 99% was obtained which was much better than the traditional unitary method. This difference in accuracy was further highlighted using confusion matrices of the training and testing phases of the unitary and the proposed methods.

The proposed approach achieved a very promising accuracy and could be extended to other time complexities as well, it reduced almost all the manual work and improved upon existing automated works which involved complex calculations.

To extend this concept, more categories are being added for which additional data is being generated and collected. Also, it is intended to incorporate adjustment factors which can take care of different processor speeds. Subsequently, accurate predictions can be made for any processor type and computer.

VII. REFERENCES

[1] https://en.wikipedia.org/wiki/Pandas_(software)

[2] https://www.datacamp.com/community/tutorials/tensorflow-tutorial

[3] https://www.tensorflow.org/

[4] https://pandas.pydata.org/

[5] http://xgboost.readthedocs.io/en/latest/model.html

[6] https://en.wikipedia.org/wiki/Random_forest

[7] https://github.com/saisumit/Complexity_Data

[8] https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

[9] Tomaz Dobravec, "Estimating the time complexity of algorithms by counting the Java bytecode instructions", Informatics, IEEE 14th International Scientific Conference, 14-16 Nov. 2017,University of Ljubljana, Vecna pot 113, 1000 Ljubljana, Slovenija, pp. 74 - 79

[10] https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest simplified/

[11] Senthil Kumar and D. Malathi, "A Novel Method to Find Time Complexity of an Algorithm by Using Control Flow Graph", Technical Advancements in Computers and Communications (ICTACC), 10-11 April 2017, ,SRM University, Kattankulathur, pp. 66 - 68

[12] http://www.cse.unt.edu/~tarau/teaching/cf1/Master%20theorem.pdf

[13] Wang Xiang, "Analysis of the Time Complexity of Quick Sort Algorithm", Information Management, Innovation Management and Industrial Engineering (ICIII), 26-27 Nov. 2011 International Conference, School of Information and Electronic Engineering Tianjin Vocational Institute Tianjin, China , pp. 408 - 410

[14] Najma Sultana, Smita Paira, Saurabh Chandra and Sk Safikul Alam, "A brief study and analysis of different searching algorithms", Electrical, Computer and Communication Technologies (ICECCT), Second International Conference, 22-24 Feb. 2017, Calcutta Institute of Technology, Uluberia Howrah, India , pp. 1-4

[15] Chen, T., and C. Guestrin (2016), XGBoost: A Scalable Tree Boosting System, in KDD '16 Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794, Assoc. for Comput. Mach., New York, doi:10.1145/2939672.2939785.

[16] http://xgboost.readthedocs.io/en/latest/model.html

[17] https://github.com/saisumit/Complexity_Data/tree/master/Code_Data

[18] https://github.com/saisumit/Complexity_Data/tree/master/Output_Data

[19] https://github.com/saisumit/Complexity_Data/tree/master/Training%20and %20Generation%20

[20] Friedman, J. H.. 2001. Greedy function approximation: a gradient boosting machine. Ann. Stat. 29: 1189–1232.