# Runtime Prediction Model using Ensemble model

Kuldeep Kumar, Gourav Gujariya
Department of Computer Science and Engineering,
Dr. B R Ambedkar National Institute of Technology Jalandhar, Jalandhar, Punjab, India
gouravg.cs.20@nitj.ac.in, kumark@nitj.ac.in

***Abstract — Time complexity is the measure of how quickly an algorithm completes a task. It is defined as the amount of time that takes to execute an algorithm on input size n with memory requirement M and processing capability P i.e T(n ,M , P) = O(n) In other words, it's equivalent to the notation of It "O (n)algorithms machine learning field in helping to provide an automation tool which with the help of some algorithmic features present in the code could compute the time complexity of a given operation. This calculation is done manually which results in human error and not being able to get an approximate assumption before creating the algorithm about their computational complexity. We will be using the corcod dataset[1]. We are using google colab here as a platform to work with the dataset and model. We will be looking into a dataset and seeing if any feature is not important to the model or imbalanced. We will be using algorithms that are helpful for multiclass classification which are: Random Forest Classifier, Knearest Classifier, and SVM. The following study shows the classifiers are trained and tested for the complexity of a given dataset. The proposed approach achieved an accuracy of 74.28% which is much higher than the existing methods. The supervised mechanism is used to exemplify the use of several classifiers for employment scam detection. Experimental results indicate that the extra tree classifier outperforms its peer classification tool.***

***Keywords—Time complexity, ensemble models, Prediction model.***

## I. Introduction

In the growing world of the internet and software. We are continuously developing new algorithms, models, and solutions for complex solutions. For example, from the analytical engine, the very first algorithm out there for calculation of Bernoulli numbers to creating human action automation algorithms, throughout the journey, we are focused on creating efficient algorithms. There are problem statements which are having their algorithmic solution present, but the work doesn't end there because we also need to make sure that the solution is efficient and more effective in comparison to previous solutions proposed. For doing so the department of design and analysis of algorithms uses the concept of time complexity where scientists calculate the time which will be required to be executed by the algorithms based on the various input given to the algorithm. In doing so, the time complexity of an algorithm is calculated by considering the number of times it will be called for input data and the number of times it will branch.

Since there are many concepts in computer science, this article will discuss only one concept, which is time complexity. many times it is misunderstood with execution time, which is machine-dependent, but here we are calculating based on input size. Time complexity is in a way the measure of how quickly an algorithm completes a task. It is defined as the amount of time that takes to execute an algorithm on input size n with memory requirement M and processing capability P i.e T(n,M,P) = O(n). In other words, it's equivalent to the notation of "O (n)" in algorithms. For solving any problem, we always need to find a good algorithm that would execute and solve it in less time and take into account the required memory and processing capabilities of the problem. The lower the value of time complexity is, the more efficient this algorithm is. But this calculation is done manually which results in human error, time consumption, the effectiveness of the answer, and not being able to get an approximate assumption before creating the algorithm about their computational complexity.

This problem statement motivates the machine learning field in helping to help provide an automation tool that, with the help of some algorithmic features present in the code, could compute the time complexity. Of a given operation. To do so, the code must be available and have a clear way of expressing what it is trying to accomplish. For example, if a function is meant to compute the complexity time of an operation, then some lines in its code will indicate the number of loops, several priority queues present in the algorithm, and more. Which will then be fed to the model and the model will be providing the time complexity, respectively. Doing so will be able to help researchers and scientists in getting an approximate assumption about the time complexity of their algorithm within less time and would help them in creating an efficient algorithm or checking the efficiency of their algorithms.

We will be using [1] dataset for the creation of the machine learning model because-

1. Created by researchers so will not contain any error
2. Created by using the word 2 vec method to work with the code input
3. Here to create code into the dataset researchers did feature engineering in a step-by-step manner where they selected only those algorithms which will be having runtime errors or time limits exceeding tags attached to them.

4. To ensure the accuracy of solutions, researchers selected only those codes that will pass at least four test input cases
5. The criterion allowed them to include multiple solutions for a single problem with different solutions having different runtime complexities.

Feature engineering of the dataset was done in two manners

1. Manually selecting the features from the algorithms which researchers have done with experts' help.
2. With word embedding and using word 2 vec where they used AST(abstract syntax tree) to get a graph for each different approach of solution and used JDT to prune the tree by removing unnecessary and unused code in the code.Using AST and Code embedding technique they created this dataset which consists of 932 source code inputs,5 complexity classes, and 16 features of algorithms.

## II. Related Work

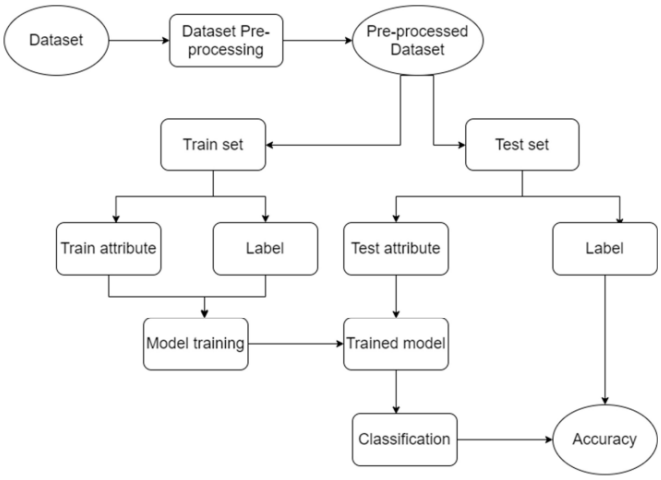
The deep learning community has done a lot of research recently on programming codes. supervised learning techniques for algorithm runtime prediction were put forth by Hutter et al. [9]. As was previously stated, execution time is not a conventional metric to evaluate the effectiveness of algorithms. As a result, we do not take algorithm execution times into account in our work. The majority of deep learning research has been concentrated in one of two areas: either anticipating some program structure or attribute or producing code snippets that are similar in sound.

In order to anticipate method names, Allamanis et al. [3] utilized a convolutional neural network(CNN) with an attention strategy. Alon et al. [4] recommended using AST routes as the background for creating code embeddings and training classifiers alongside. Call graphs were utilized by Yonai et al. [17] to calculate method embeddings and suggest names for existing methods that had functionality resembling the target function. A study on word embedding methods applied to source code was conducted by Chen and Monperrus [8]. The use of code embeddings to anticipate the temporal complexity of programs, however, has not yet been done. Using graph2vec, we established the same as one of our baselines [13].

A study on word embedding methods applied to source code was conducted by Chen and Monperrus [8]. The use of code embeddings to anticipate the temporal complexity of programs, however, has not yet been done. Using graph2vec, we established the same as one of our baselines [13].

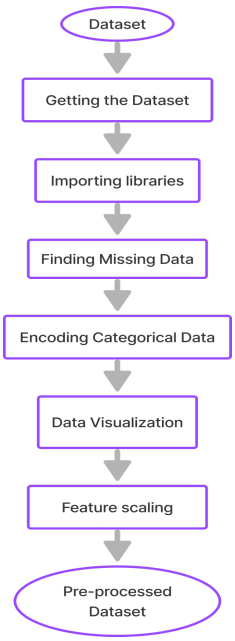
For automatic program grading, Srikant and Aggarwal [14] extract hand-engineered elements from Control Flow and Data Dependency graphs of programs, in such a manner that the number of nested loops, and the number of if statements in a loop, etc. According to grading criteria they utilized. To compute complexity and use them to train the classification models, we employ the same strategy of selecting essential features as the other baseline, which are structures that a human evaluator would consider. However, unlike [14], our characteristics are independent of the problem. Furthermore, the approach in [14] is used commercially, therefore their dataset is not accessible to the general public.

## III. General approach




1. A general approach for creating a machine-learning algorithm to calculate the runtime complexity of algorithms is to first do preprocessing where we will be looking into the dataset and performing necessary changes to the dataset before using it in model training.

2. Preprocessing include process such as :




   a. Getting the dataset: The dataset is collected directly from the [1] GitHub repository. We are using google colab here as a platform to work with the dataset and model because google colab

provide free usage of GPU, CPU, RAM, and numerous libraries to work with without having to install them into our local machine, and AI assistance to increase speed in implementation. Whereas using the Jupiter notebook would require power consumption and installing all the libraries manually.

b. Importing libraries: In the model creation, we will be importing libraries that are basic to use and are necessary for every model creation and data analysis.

c. Importing datasets: For importing the dataset we are using pandas.

After importing the dataset as data we looked into the dataset's first 3 rows

3. Finding Missing Data: In the following dataset there are no null values present and the dataset is pre-cleaned.
4. Encoding Categorical Data: From using

   we came to know that there are 2 variables that are categorical and needed to be changed into numerical so we could use them in model creation but one feature named file.java is the name of the algorithm to which the features belong which will not help the model in learning anything important so we removed this feature and are left with only complexity as a categorical variable. To deal with this we used label encoders from sklearns preprocessing and converted them into integers. But there is one drawback to label encoder that it provides unique numbers to variables what if the model learns them as a priority in our case model is having complexity prediction and this limitation is not applied in this case.

   Before going for splitting the dataset we will be looking into the dataset and see if there is any feature that is not important to the model or is imbalanced.In the following diagram[2] we saw how features are related to each other and none of them is not related to complexity meaning correlation=0 From the graph[3], we can see that the data is having imbalance at some features but it is because the features are having the following nature, for example, priority queue feature we can see that it is having only logn or 0 nothing else is because when priority queue is present it will take only logn time.So we cannot work about the imbalanced features as they are fact-based and cannot be altered.

5. Feature scaling: As we have seen the data distribution and have converted categorical variables into numerical now we should scale them without the target variable. Many a time machine learning algorithms work more effectively with scalled data rather than non-scaled data that's why we are going to use a standard scaler in order to scale the dataset.

First, we divided them into features and target variables by dropping the complexity feature's column.

Now as the X is separated we will scale it.

Splitting dataset into training and test set: As the dataset is now ready to be fed into a model for training we will be needing to split the dataset into parts where for the first part that is training dataset will be 70% of the dataset on which training on the model will be done and on the rest of 30% will be used for testing the model to see how good or bad training have the model done. It is the main point to note that splitting of the data for training is important as if such actions are not done then the model will perform very badly when new data will be fed to it which will result in its breakdown.So for doing so we are using train_test_split from sklearns model selection.

Now the data is ready to be fed to algorithms to learn.

Model selection: As this problem statement is especially for multiclass classification of algorithms, we will be using algorithms that are helpful for the multiclass classification which is:

1. Random Forest classifier
2. K-nearest Classifier
3. SVM

1. *Random Forest classifier:* In ML, Random Forest can be applied to Classification and Regression issues. It is built on the idea of ensemble learning, which is a method of integrating various classifiers to address difficult issues and enhance model performance.A Random Forest, as the name implies, is a classifier that uses a number of decision trees on different subsets of the provided dataset and averages them to increase the dataset's predictive accuracy. Instead than depending on a single decision tree, the random forest uses forecasts from each tree and predicts the result based on the votes of the majority of predictions.Higher accuracy and overfitting are prevented by the larger number of trees in the forest.The data set used for the creation of a random forest should be as large as possible, both in the number of cases and attributes *Before training a random forest algorithm, certain hyperparameters must be set. These include node size, tree count, and sampled feature count, among others. The random forest classifier can then be used to solve classification or regression issues. A Random Forest method is a collection of several decision trees. The data sample used to build these trees is taken from a training set through the bootstrap sample technique with replacement. One-third of the training data is kept aside and referred to as the out-of-bag (OOB) sample, which can be used to test later. After the dataset has been inputted, feature bagging adds a further randomness which enhances its diversity & reduces any correlations. This leads to an increase in range & variety.*
2. *K Nearest Neighbour*:K Nearest Neighbour is one of the simplest Machine Learning algorithms and it's based on the Supervised Learning technique. It assumes the similarity between the new case/data and available cases and it will put the new case into a

category that is most similar to these. The K-NN algorithm lets you categorize your data quickly. It stores all the available data to be easily retrievable and simply updates when new data appears. This is a non-parametric method/algorithm. This means it does not rely on understanding the "underlying structure of the data" and just focuses on predicting outcomes. It is also sometimes nicknamed a "lazy learner" because it relies more on less computationally expensive methods like remembering items seen in training classes than it does on understanding underlying relationships between items in the AI learns the categorization process by saving the dataset and waiting until there is a new dataset before classifying it. Earlier, it only saves the worker's data and when they got new data, they would classify that into a category too.

3. *Support vector machines:* A supervised learning technique known as an SVM is utilised to solve both classification and regression issues. To simply place fresh data points in the appropriate category in the future, the SVM method aims to establish the optimal line or A decision boundary that can divide the dimensional space into classes (hyperplane). To distinguish between the two classes of hyperplanes to distinguish between the two classes of data points.Due to this feature of SVM we can use this algorithm in the multiclass classiffication Our goal is to find a plane with the greatest margin—that is, the greatest separation between data points from both classes—is our goal. Maximizing the margin distance adds some support, increasing the confidence with which future data points may be classified.

As more classifiers do multiclass classification but as a general approach we will be selecting them as a baseline and from this paper Learning Based Methods for Code Runtime Complexity Prediction[5] | Springer Link they have got the most accuracy and good F1 score for these algorithms so we have selected their lead here. To get a fast and clear assumption we have used a cross-validation score on these algorithms.

## III. Implementation of Methods

This section discusses the implementation of experimental setup that is used for the classification of multiple categories of time complexity. There are many options available when it comes to choosing a platform for machine learning and data science. Two of the most popular choices are Google Colab and Jupiter Notebook. So, which one is the best for your needs?

Both Google Colab and Jupiter Notebook offer fast performance for machine learning and data science tasks. However, Google Colab is slightly faster than Jupiter Notebook, due to its use of GPUs and TPUs. Additionally, Google Colab offers more libraries and easier model creation than Jupiter Notebook.Google Colab offers more libraries and easier model creation than Jupiter Notebook. This makes it a great choice for beginners who want to get started with machine learning.Google Colab is a great choice for starters who want to get started with machine learning. It offers more libraries and easier model creation than Jupiter Notebook.

By using Sklearn we are going to implement the above algorithms on the preprocessed dataset. The implementation details of machine learning tools with the usage of cross validation score from sklearn to evaluate their accuracy on the training dataset is shown in the diagram.

| ALGORITHMS | Cross val score |
|---|---|
| Random Forest Classifier | 0.72519084 |
| KNeighbourClassifier | 0.67175573 |
| Support vector machine | 0.70229008 |

And from the score, we can select the **RandomForestClassifier** for providing maximum score in cross val score according to general approach for prediction and model creation. Whereas other algorithms were not providing very promising results. This practice shows us that ensemble models provide more accurate results on this dataset and by using them and tweaking their parameters we could achieve even more accuracy.

### *A. Ensemble approach*

As we can see that RandomForest did far better then the other algorithms in terms of the cross val score so taking that as an approach. We consider that if other ensemble algorithms such as Bagging,Boosting, stacking,Voting ,extra tree classifiers could probably provide a better solution for prediction of runtime complexity. This classification model is created by organizing the various layers of algorithms as shown in Figure 2.

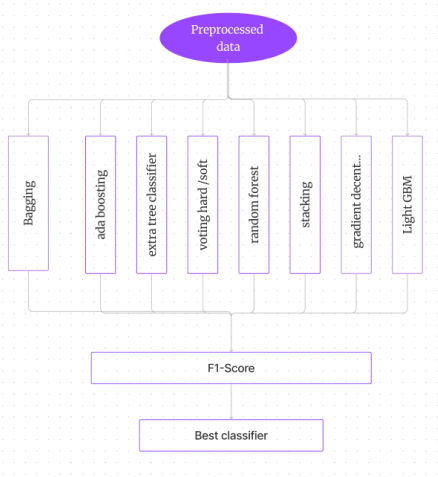



Fig. 2. Working diagram of the ensemble models for comparative analysis

*Extra Tree classifier: An extra tree classifier is a machine learning multiclass classifier that is trained using a set of data that is representative of the data that will be used to classify. The extra tree classifier is a fast, accurate, and efficient way to classify new data. it is a type of decision tree classifier, where the tree is generated using a random subset of the features, and the splits are chosen using a random subset of the data points. The extra tree classifier is more robust to overfitting than a traditional decision tree classifier and can be used to classify data with a high degree of accuracy. Since only one decision-making pathway is used, a single decision tree typically overfits the data it is learning from. Typically, predictions made from a single decision tree are inaccurate when applied to new data. By adding unpredictability, random forest models lessen the chance of overfitting With two major distinctions, Extra Trees is similar to Random Forest in that it generates multiple trees and separates nodes using random subsets of characteristics. It samples but does not bootstrap observations.*

*ADABoosting algorithm: Adaboosting is a machine learning technique that improves the performance of a multiclass classifier. Adaboosting works by combining several weak classifiers to create a stronger classifier. This technique is often used in a variety of different fields, such as computer vision(CV) and natural language processing(NLP). The improperly classified record in the first model is given priority as the first decision tree or model is constructed. Until we indicate the amount of base learners we wish to create, this process continues. It creates Stumps, which are trees made up of a start node and numerous leaf nodes. In AdaBoost, the placement of the stumps is crucial because the first stump's mistake affects how the others are formed. Let's say the algorithm used the aforementioned dataset to create three decision trees or stumps. The test dataset will go through every stump that the algorithm generated. It produces 1 output as it passes through the first stump. Once more, 1 is the output after going through the second stump. While passing through the third stump, it shows 0 as the output.*

*Bagging classifier: A particular kind of ensemble model called bagging trains each model on a subset of the data. Overfitting can be decreased using this method. A particular kind of ensemble model called bagging trains each model on a subset of the data. This method can be used to lessen overfitting and raise the ensemble's models' accuracy. A particular kind of ensemble model called bagging trains each model on a subset of the data. This method can be used to lessen overfitting and raise the ensemble's models' accuracy. Bagging can assist in preventing overfitting and enhancing overall accuracy by using a smaller data set for each model in the ensemble.*

*Light GBM (Light Gradient Boosting Machine): It is an open-source gradient boosting framework developed by Microsoft. It is designed to be faster and more efficient than other boosting algorithms by using a novel technique called Gradient-based One-Side Sampling (GOSS). The GOSS technique reduces the number of instances in the training set by keeping the ones that have large gradients and discarding the ones that have small gradients, which leads to faster training and less overfitting. The EFB technique bundles mutually exclusive features into a single feature to reduce memory usage and increase the speed of training.It can handle large-scale datasets and can be used for various machine learning tasks such as classification, regression, and ranking. It also supports distributed training. It is fast, as it can train models faster than other popular gradient boosting frameworks like XGBoost and CatBoost. Additionally, it has been shown to achieve better accuracy and have lower memory usage in certain scenarios .It is a powerful and efficient machine learning framework that can handle large-scale datasets and achieve high accuracy in a short amount of time.*

*XGBoost(Extreme Gradient Boosting):XgBoost is a powerful machine learning algorithm that has gained widespread popularity in recent years. It is a type of ensemble learning method that combines multiple decision trees to make more accurate predictions. It has been widely used in various fields such as finance, healthcare, and natural language processing due to its speed and accuracy. It can handle both regression*

*and classification problems and has been shown to outperform other popular machine learning algorithms like Random Forest and Neural Networks in certain scenarios. XGBoost incorporates several features such as regularization, parallel processing, and handling of missing values, which makes it a versatile and scalable machine learning algorithm. It is available in multiple programming languages and has been integrated into several popular machine learning frameworks, making it accessible to a wide range of users. Overall, XGBoost is a powerful tool for solving complex machine learning problems with high accuracy and efficiency.*

*Stacking:A sort of ensemble model known as stacking provides predictions by fusing the results of various models. This method can be applied to increase the precision of predictions. A sort of ensemble model known as stacking provides predictions by fusing the results of various models. This method can be applied to increase the precision of predictions. Stacking is a very effective method for increasing prediction accuracy when combined with a voting system. A sort of ensemble model known as stacking combines the forecasts of various models to increase accuracy. When combined with a voting technique, this method can be quite effective because it dramatically increases prediction accuracy.*

*Voting Hard/soft: A form of machine learning method called a multiclass classifier can be used to divide data into more than two classes. Voting classifiers come in two varieties: soft and hard. Hard classifiers only forecast the class that received the most votes. Soft classifiers are more forgiving and more likely to include a wider range of candidates.*

*A voting classifier uses machine learning to predict an output based on the class that has the highest favorable likelihood of being the outcome. Soft voting uses the average probability of all estimators, so the output accuracy will be higher.* Voting classifiers are simple to use and can significantly increase accuracy. They do have their limitations, though, as they can take a while to forecast and train.*

*GridsearchCV: GridSearchCV is a method for adjusting hyperparameters to find the best values for a particular model. As was already noted, the value of its hyper-parameters strongly influenced a model's performance. Considering that there is no way to determine the best possible values for hyper-parameters in advance, it is ideal to explore every conceivable value before deciding what the best ones are. We utilize GridSearchCV to automate the tweaking of hyper-parameters because doing it manually could take a lot of time and resources. The model selection package of Scikit-learn (or SK-learn) contains a method called GridSearchCV. This function aids in fitting your estimator (model) to your training set by looping through pre-defined hyper-parameters. The Scikit-Learn library must be installed on the.*

B. *Behavior of algorithms on dataset*

1. *As we are here trying to figure out which model will be the best for the prediction of runtime complexity. Let's see how the models are implemented and how they are working on the preprocessed dataset.*
2. *GridSearchCV: By using GridsearchCV we are hyper parameter tunning the extra tree classifier and random Forest so that we could get the best possible results for our model's working.*
3. *ExtraTreeClassifier: By using GridsearchCV we got the number of trees=50, criterion = entropy which indicates that the following algorithm is going to use entropy for its branching.*
4. *RandomForestClassifer: By using GridsearchCV we got num of trees= 110,class_weight to be balanced and criterion to be entropy as similar to the extra tree classifier.*
5. *AdaBoost classifier: We are using Randomforestclassifier created using gridsearchCV and extra tree classifier using gridsearchCV as a base estimator by using which it tries to correct itself till it is not done.*
6. *Bagging classifier: We are using RandomforestClassifier created by using GridsearchCV and providing n_estimators in a range from 50 to 500 with the step of 100 to get the best outcome which we received with 150 trees.*
7. *Stackingclassifier: This classifier is using ExtraTreeClassifier and RandomForestClassifier as its estimators and will be working on 70% of the preprocessed data.*
8. *Voting classifier: This classifier is using extra tree classifier and RandomForestClassifier as its base estimator and uses class weight balanced with criterion entropy. By using of voting classifier's Voting hyper parameter we can decide whether it to be hard or soft using which we have created it in 2 parts hard voting classifier and a soft voting classifier.*

## IV. Results and Discussion

This section discusses the results of the classification of the dataset used in this study.

### A. Datasets Used

Dataset- Midas research corcod-dataset[1] open to use and access created by researchers is used in this learning. The following dataset is divided into 80:20 training set and testing sets respectively.This runtime prediction dataset consists of a total 16 attributes . Which include following features

1. No_of_ifs:no of if presents in algorithms
2. no_of_switches: no of switches present in algorithm
3. no_of_loop: no of loops present in a algorithm
4. no_of_break: no of the breaks in an algorithm
5. priority_queue_present: if priority is present or not
6. no_of_sort:no of sort present
7. hash_set_present:hash set is present or not in algorithm
8. hash_map_present:hash map present or not
9. recursion_present:recursion present or not
10. nested_loop_depth:nested loop present or not
11. noOfVariables:no of variables present in algorithm
12. noOfMethods:no of methods used
13. noOfJumps:no of jumps done between statements in algorithm
14. noOfStatements:no of statement in an algorithm
15. complexity:time complexity of the algorithm (target feature)
16. file_name:filename to which algorithm is related.

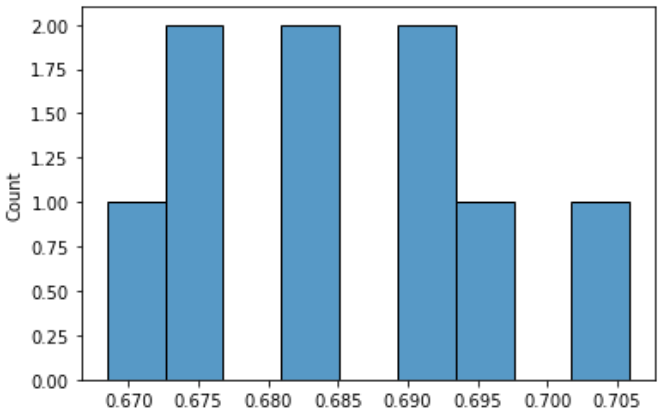
It contains 933 instances .

TABLE I. NUMBER OF INSTANCES IN TRAINING AND TESTING

| | **Number of Size after completion of the spliting of traing and testing** | |
|---|---|---|
| **Dataset** | **Text attribute** | **Title attribute** |
| Dataset | 187 | 746 |

### *B. Results*

The mentioned multiclass classifiers are trained and tested on the dataset for the prediction of time complexity . The following Table 1 shows the side by side study of the classifiers concerned with judging metrics and Table 2 provides results for the multiclass classifiers that are based on ensemble techniques. Fig. 4 to Fig. 7 shows the comparative performance of all the proposed classifiers in terms of accuracy and f1-score.




| index | CLASSIFIER | ACCURACY |
|---|---|---|
| 1 | KNN | 0.6928 |
| 2 | RANDOM FOREST CLASSIFIER | 0.6898 |
| 3 | SUPPORT VECTOR MACHINE | 0.6821 |
| 4 | LIGHT BM | 0.6737 |
| 5 | ADABOOSTING | 0.7058 |
| 6 | BAGGING | 0.7464 |
| 7 | EXTRA TREE CLASSIFIER | 0.6844 |
| 8 | STACKING | 0.7178 |
| 9 | VOTING HARD | 0.6898 |
| 10 | VOTING SOFT | 0.6951 |
| 11 | GradientBoostingClassifier | 0.7583 |
| 12 | xtreme gradient boosting | 0.7583 |

| SSIFIER | '1', | 'logn', | 'n', | 'n_square', | 'nlogn' |
|---|---|---|---|---|---|
| | 0.8571428 6 | 0.56 | 0.7230769 2 | 0.59649123 | 0.5974026 |
| OM FOREST SSIFIER | 0.727 | 0.545 | 0.772 | 0.677 | 0.625 |
| ORT TOR CHINE | 0.6388 | 0.375 | 0.70 | 0.71 | 0.6285 |
| OOSTING | 0.777 | 0.5 | 0.7417 | 0.666 | 0.653 |
| ING | 0.8048780 5 | 0.6 | 0.7908745 2 | 0.69491525 | 0.64935065 |
| A TREE SSIFIER | 0.7901234 6 | 0.6666666 7 | 0.7773584 9 | 0.70689655 | 0.64935065 |
| KING | 0.7848101 3 | 0.5714285 7 | 0.7698113 2 | 0.64957265 | 0.61538462 |

| | | | | | |
|---|---|---|---|---|---|
| light BGM | .075 | 0.6956 | 0.708 | 0.6804 | 0.48 |
| gradient decent boosting | 0.76 | 0.56 | 0.70 | 0.6530 | 0.52 |
| VOTING HARD | 0.8192 77 11 | 0.521739 13 | 0.763358 78 | 0.6440678 | 0.648648 6 |
| VOTING SOFT | 0.77 5 | 0.545454 55 | 0.773946 36 | 0.66666667 | 0.6 |

From the above graph we can see that classifiers like Random forest, stacking, voting hard, extra the classifier, voting soft, and bagging outperform knn, and SVM and we can say that these classifiers are not suitable for the prediction so for further analysis we take a look at f1 score classifiers among Random forest classifier, stacking voting hard, voting soft and bagging, etc some classifiers like voting soft are classifying some feature very effectively but are not classifying some categorical features with 0.54 f1 score and is not suitable for selection. Similarly voting hard, stacking, and Random forest have the same problem but we can see that among bagging an extra tree classifier we can see the adaboosting with extra tree classifier is outperforming and classifying all features more efficiently.

## V. Limitation

The main limitation of machine learning models is that they require a lot of data in order to work properly. The more data there is, the better the results are. For example, if we have a small dataset and want to use it to train our machine learning model, then it might not be as accurate as a model trained on a larger dataset.Overfitting of models over a small size dataset could be easily seen in the examples of voting hard and bagging classifiers which could be resolved by increasing the dataset with the algorithmic features having imbalance.The problem with the prediction problem is if new data is given where complexity is new. Then in such cases, the dataset will be not able to classify it and probably classify that with the wrong class.Bias in categories of complexity could be seen in the dataset where there is more number of the algorithm in logn complexity than 1. Which is reducing the accuracy of models and making them classify algorithms into the wrong class which could be removed by increasing the size of the dataset.

## VI. Conclusion and Future Scope

Runtime complexity prediction will help the researchers in getting an idea of how their algorithm is performing and could save their time of computation. For the prediction of time complexity, multiple machine learning algorithms are submitted as countermeasures throughout this study. The organized mechanism is used as an example for the usage of multiple classifiers for runtime complexity prediction. Experimental results states that the extra tree classifier outperforms its equally performing machine learning classification tools. The proposed approach achieved an accuracy of 70.28% which is higher than current methods proposed and F1 score of 0.77777778, 0.57142857, 0.74172185, 0.66666667, 0.65384615 is perfectly classifying the categories.

The results for the study of the samples is practically based on the dataset with similar studies and will be providing the results based on the approval of the samples for a similar dataset that is used in this case study. The further studies on the prediction for complexity can be extended based on the available dataset attributes to find out the accuracy of the model. This can be extended for a dataset containing more algorithms with various complexity or various feature. This learning only put forth the topic on the multiclass classification of complexities but in the future, it can extend for the prediction of time complexity of various algorithms having various features and it can also be put forth for the similar datasets like content and for those containing the algorithmic features as an attribute, etc.

## References


[1] dataset used in model creation created by concord research center and researcher https://github.com/midas-research/corcod-dataset/blob/master/README.md

[2] Algorithm Runtime Prediction: Methods & Evaluation Frank Hutter, Lin Xu, Holger H. Hoos, Kevin Leyton-Brown https://arxiv.org/abs/1211.0906

[3] A Convolutional Attention Network for Extreme Summarization of Source Code Miltiadis Allamanis, Hao Peng, Charles Sutton Proceedings of The 33rd International Conference on Machine Learning, PMLR 48:2091-2100, 2016. http://proceedings.mlr.press/v48/allamanis16.html

[4] A General Path-Based Representation for Predicting Program PropertiesUri Alon, Meital Zilberstein, Omer Levy, Eran Yahav https://arxiv.org/abs/1803.09544

[5] Mercem: Method Name Recommendation Based on Call Graph EmbeddingHiroshi Yonai, Yasuhiro Hayase, Hiroyuki Kitagawa https://arxiv.org/abs/1907.05690

[6] A Literature Study of Embeddings on Source CodeZimin Chen, Martin Monperrus https://arxiv.org/abs/1904.03061

[7] graph2vec: Learning Distributed Representations of Graphs Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, Shantanu Jaiswal https://arxiv.org/abs/1707.05005

[8] A system to grade computer programming skills using machine learningAuthors: Shashank Srikant, Varun Aggarwal Authors Info & Claims https://dl.acm.org/doi/10.1145/2623330.2623377