INDUSTRIAL TRAINING

On

Comparison of hyperparameter optimization methods in ML

An industrial training project report submitted

to

MANIPAL ACADEMY OF HIGHER EDUCATION

For Partial Fulfillment of the Requirement for the

Award of the Degree

of

Bachelor of Technology

in

Information and Communication Technology

by

Chinmay Das

Reg. No. 190953132

Under the guidance of

Mr. Anup Kumar and Dr. rer nat. Björn Grüning,

Bioinformatics Group

Albert-Ludwig University of Freiburg, Germany

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

October 2022

CERTIFICATE

"official letterhead of the company and seal/signature by the mentor/HR"

This is to certify that **Mr. Chinmay Das** (Reg. No.: **190953132**), a student of 4th year B.Tech. (Information and Communication Technology) from Manipal Institute of Technology has completed his/her internship with us from 25-08-22 to 05-10-22.

During this tenure he has worked on project titled "**Comparison of hyperparameter optimization methods in ML**" and has successfully completed it to the best of his abilities.

His conduct and attendance have been good during this tenure.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Mr. Anup Kumar and Dr. rer nat. Björn Grüning for providing me with the opportunity to intern at the University of Freiburg. Also, a huge thanks to Mr. Anup Kumar for mentoring me and guiding me throughout the internship. He was extremely helpful and encouraged me to explore more and perform better throughout the internship. I often found myself surrounded by doubts and having no idea of how to proceed. These times were made so much easier because of the constant guidance. There were a lot of online resources that I referred to for learning and when facing doubts. I would also like to show my gratitude towards these online resources which make our lives in tech exponentially easier. I would also like to thank my family who helped me with all the non-technical resources required to successfully complete this internship and for being supportive along the way.

ABSTRACT

This project helps to get introduced to machine learning and different algorithms and methods used therein. The main objective of my project specifically was to compare and analyze the various search methods used in Hyperparameter Tuning. Any Machine Learning (ML) algorithm, be it classification, regression or any other, is performed via a set of instructions, using functions and algorithms. Sci-kit Learn is a library which has many such tools to help us perform these tasks efficiently and accurately. The methods provided all have some pre-defined parameters which take a set of different values. These different parameters are called Hyperparameters. To achieve maximum efficiency, it is important we find out the set of parameters that are the most optimal with regards to the nature of the task to be performed.

The process of finding out the most optimal set of parameters for a function is called Hyperparameter Optimization or Hyperparameter tuning. There are mainly three types of ways in which one can search the set of parameters which is the most optimal: Grid Search, Random Search, Bayesian Search. Our project compares these three methods and tells us which method to use in what case.

The tools and technologies used for this project are:

- Python version 3.0 and above
- IDE - PyCharm or Jupyter Notebook
- Conda Environment
- Libraries needed:
  - Sklearn
  - Numpy
  - Pandas
  - PMLB
  - Matplotlib

Contents

Details of the Organization

Albert Ludwig University of Freiburg, Germany

The University of Freiburg, or Albert Ludwig University of Freiburg in German, is a public research university with its main campus in Freiburg am Breisgau, Baden-Württemberg, Germany. As the second university in Austrian-Habsburg territory after the University of Vienna, the university was established by the Habsburg dynasty in 1457. With a long history of teaching the arts, social sciences, natural sciences, and technology, Freiburg is currently the fifth-oldest university in Germany. It has a strong academic reputation both domestically and abroad.

Chapter 1

Introduction

1.1 Brief introduction to the project and present-day scenario w.r.t the project

One of the most well-known branches of computer science in the current day and age is machine learning. Nearly every industry, including healthcare, finance, infrastructure, marketing, self-driving vehicles, recommendation systems, chatbots, social media, gaming, cyber security, and many more, use machine learning techniques.

1.2 Motivation

The control of a machine learning model's behavior requires hyperparameter optimization. Our predicted model parameters will yield less-than-ideal outcomes if our hyperparameters aren't properly tuned to minimize the loss function. This indicates that our model has additional flaws. It hence becomes a matter of utmost importance to tune our hyperparameters properly, and to do so efficiently. Out of all the different search methods, which one is the best fit for which scenario is something one must have a good idea of.

1.3 Problem Definition/Objectives

Since hyperparameters are unique to the specified algorithm, we are unable to determine their values from the data. For a given data collection, various hyperparameter values result in various model parameter values. Grid search, random search, and Bayesian optimization are automated techniques for hyperparameter tuning that are currently most in demand. We need to know which method gives us which kind of results. E.g., Grid search can be very accurate on tuning classification algorithms but can take a very long time to do so. We need a comparison between these three search methods on various algorithms, in order to analyze which one should we favor in what scenarios.

Chapter 2

Background Theory / Related Work

2.1 Machine Learning

Machine Learning (ML) is a form of artificial intelligence (AI). With the use of it, software programs may be able to predict outcomes more accurately without having to be explicitly instructed to do so. In Layman's terms, in order to predict new output values, machine learning algorithms use past data as input. Machine learning is significant because it aids in the creation of new products and provides businesses with a picture of trends in consumer behavior and operational business patterns. A significant portion of the operations of many of today's top businesses, like Facebook, Google, and Uber, revolve around machine learning. For many businesses, machine learning has emerged as a key competitive differentiation.

2.2 Hyperparameter Tuning and Search Methods

Parameters and hyperparameters are two terms widely used in machine learning. It's important to distinguish between these two. For the provided data set, a learning algorithm learns or predicts the model parameters before updating these values over time. On the other hand, we cannot determine the values of hyperparameters from the data since they are unique to the algorithm itself. To determine the model parameters, we employ hyperparameters. For a given data collection, various hyperparameter values result in various model parameter values. For example, some important hyperparameters that require tuning in neural networks are number of hidden layers, number of nodes/neurons per layer, learning rate, momentum, etc. The essential hyperparameters for tuning SVMs, for example, are C and Gamma.

Finding a set of ideal hyperparameter values for a learning algorithm and using this tuned algorithm on any data set is hyperparameter tuning. The model's performance is maximized by using that set of hyperparameters, which minimizes a predetermined loss function thus resulting in better outcomes with fewer errors. It should be noted that the learning algorithm attempts to discover the best solution within the constraints and optimizes the loss depending on the input data. Hyperparameters, however, precisely specify this configuration.

We need to know how to pick hyperparameters' ideal values now that we are aware of what they are and how crucial it is to tune them. Both manual and automated techniques may be used to determine these ideal hyperparameter values.

When manually tuning hyperparameters, we often start with the default suggested values or rules of thumb, then use trial-and-error to explore a variety of values. Manual tuning, however, is a tiresome and time-consuming method. When there are several hyperparameters with a broad range, it is not practicable.

Automated hyperparameter tuning techniques look for the best values using an algorithm. Grid search, random search, and Bayesian optimization are some of the automated techniques that are most widely used. Let's look more closely at these techniques.

Grid Search: Grid search is a kind of "brute force" approach to hyperparameter optimization. After constructing a grid of potential discrete hyperparameter values, we fit the model using every combination conceivable. An extensive process called grid search can identify the ideal set of hyperparameters. The disadvantage is that it moves slowly. It typically takes a lot of processing power and time to fit the model with every potential combination, which may not be accessible.

Random Search: The random search technique, as the name suggests, selects values at random rather than utilizing a preset set of values like the grid search method does. Each time a random combination of hyperparameters is tested, the

model's performance is recorded. It eventually returns the mixture that gave the greatest outcome after multiple repetitions. When we have numerous hyperparameters with somewhat big search domains, random search is acceptable. The advantage is that random search frequently takes less time than grid search to provide results that are equivalent. The output might not be the ideal mix of hyperparameters, which is one of its disadvantages.

Bayesian Optimization: According to his approach, finding the ideal hyperparameters is an optimization issue. This strategy takes the outcomes of the previous evaluation into account when selecting the subsequent hyperparameter combination. After that, a probabilistic function is used to choose the combination that will most likely produce the greatest outcomes. With this approach, a reasonable hyperparameter combination is found in a manageable number of rounds. The probabilistic model is founded on the findings of earlier evaluations. It calculates the likelihood that an objective function will be obtained for a given set of hyperparameters: P (result | hyperparameters). Because Bayesian optimization must be computed sequentially (where the subsequent iteration depends on the preceding one), it has the disadvantage of not allowing distributed processing when compared to grid search or random search.

2.3 Python libraries and Sci-kit learn

Python helps simplify sophisticated prediction technologies like AI algorithms and machine learning models. Its abundant machine learning-specific libraries and clean code give it the potential to move the emphasis away from the language and toward the algorithms. Additionally, it is reliable, intuitive, and fairly simple to master. With its frameworks, libraries, and community support, Python is genuinely excellent. Python is therefore the most used language for machine learning.

Python allows for the construction and operation of software solutions across a variety of platforms and operating systems. Linux, Windows, Mac, Solaris, and more are a few examples. Python machine learning programming is now much

more practical as a result. Another factor contributing to developers' preference for Python while creating ML applications.

Python draws on a broad collection of libraries and frameworks for machine learning applications. For instance:

- NumPy works with arrays and various matrices.
- Using the Matplotlib package, Python can produce static, animated, and interactive visualizations.
- A Python-based data visualization package called Seaborn provides users the ability to create eye-catching, high-quality visualizations (statistics).
- Sci-kit Learn: Scikit-learn is an open-source Python toolkit that uses a uniform interface to build a variety of machine learning, pre-processing, cross-validation, and visualization algorithms.
- PMLB (Penn Machine Learning Benchmarking): Once we build our model and are ready to compare and analyze, we need a dataset to benchmark our comparisons. PMLB library provides us with a huge number of benchmarking datasets, and this can also be installed and imported to our Python environment.

2.4 Classification and Regression Methods

The Classification method, a Supervised Learning approach, is used to categorize fresh observations in light of training data. In classification, a software makes use of the dataset or observations that are provided to learn how to categorize fresh observations into various classes or groups. For instance, cat or dog, yes or no, 0 or 1, spam or not spam, etc. Targets, labels, or categories can all be used to describe classes.

Regression is a statistical research area that is essential to machine learning forecast models. It is useful for forecasting and predicting outcomes from data since it is used as a method to predict continuous outcomes in predictive modelling. Regression using machine learning often entails drawing a line of best fit through the data points. Unlike classification, regression predicts a value and not a class.

There are multiple algorithms which can be applied in ML to perform the tasks of classification and regression. In the project we take a look at four such algorithms. They are:

Decision Tree Algorithm: Decision Tree is a supervised learning method that can be applied to classification and regression issues; however, it is most frequently used to address classification issues. It is a tree-structured classifier, where internal nodes stand in for a dataset's features, branches for the decision-making process, and each leaf node for the classification result. The Decision Node and Leaf Node are the two nodes of a decision tree. While Leaf nodes are the results of decisions and do not have any more branches, Decision nodes are used to create decisions and have numerous branches. The given dataset's features are used to execute the test or make the decisions. The following algorithm can help you better understand the entire process:

Step 1: According to S, start the tree at the root node, which has the entire dataset.

Step 2: Utilize the Attribute Selection Measure to identify the dataset's top attribute (ASM) (Based on Gini Index and Information Gain).

Step 3: Separate the S into subsets that include potential values for the best qualities.

Step 4: Create the decision tree node that has the best attribute

Step 5: Use the selections of the dataset generated in step 3 to iteratively develop new decision trees. Continue along this path until you reach a point when you can no longer categorize the nodes and you refer to the last node as a leaf node.

Random Forest Algorithm: 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.

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. Rather 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 Working process can be explained in the below steps:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

SVM Algorithm: One of the most used supervised learning algorithms, Support Vector Machine, or SVM, is used to solve Classification and Regression problems. However, it is largely employed in Machine Learning Classification issues.

The SVM algorithm's objective is to establish the best line or decision boundary that can divide n-dimensional space into classes, allowing us to quickly classify fresh data points in the future. A hyperplane is the name given to this optimal decision boundary.

SVM selects the extreme vectors and points that aid in the creation of the hyperplane. Support vectors, which are used to represent these extreme instances, form the basis for the SVM method.

KNN Algorithm: K-Nearest Neighbor is a simple Machine Learning method that uses the Supervised Learning technique.

The K-NN algorithm assumes that the new case and the existing cases are comparable, and it places the new instance in the category that is most like the existing categories.

A new data point is classified using the K-NN algorithm based on similarity after all the existing data has been stored. This means that utilizing the K-NN method, fresh data can be quickly and accurately sorted into a suitable category.

Although the K-NN approach is most frequently employed for classification problems, it can also be utilized for regression.

Since K-NN is a non-parametric technique, it makes no assumptions about the underlying data.

The following algorithm can be used to describe how the K-NN works:

Step 1: Decide on the neighbors' K-numbers.

Step 2: Determine the Euclidean distance between K neighbors.

Step 3: Based on the determined Euclidean distance, select the K closest neighbors.

Step 4: Count the number of data points in each category among these k neighbors.

Step 5: Assign the fresh data points to the category where the neighbor count is highest. Our model is complete.

Chapter 3

Methodology

3.1 Basic Idea and Blueprint of Execution

The idea of the project is to compare the Bayesian hyperparameter optimization approach with traditional random and grid search approaches. The intelligent search in the constrained domain of hyperparameters for the ideal set of values of hyperparameters may improve the predictive power of Scikit-learn's supervised algorithms in a fewer number of iterations. We thus prepare an algorithm such that it can perform hyperparameter tuning using all three methods, namely Grid Search, Random Search and Bayesian Optimization, and then compare them on some metric.

For the classification algorithms, the methods can be easily compared by checking the accuracy score of each of them. For regression, however, there are other metrics we need to look at, such as F1 score, Precision and recall values, etc. For our project we have gone with the “r2” score of the algorithm, as the comparing metric.

Just giving good scores, be it accuracy or r2, is not enough for us and we need to also know whether our chosen search method is doing so efficiently. For this we find out the average time of execution of each tuning method and compare them against each other. To be able to visualize these comparisons, we will plot them on a graph so that it is easier to analyze.

3.2 Model selection

Since the project focuses on classification and regression, out of the many available models in sklearn, we need the ones that can be used for both

classification and regression. We, hence, choose the following four models to go ahead with: Decision Tree, Random Forest, SVM and KNN. Each of these have a different set of hyperparameters for our search methods to tune. Apart from that, the search methods can be compared based on the accuracy score for classification and r2 score for regression, and the comparison will have more contrast using the afore mentioned algorithms.

3.3 Evaluation metrics and calculations

For classification algorithms, we need to just check for the accuracy score as the evaluation metric. Accuracy score tells us whether the prediction made by our algorithm is the same as the actual class of the data/variable. For our algorithm to give out more accurate scores the hyper parameters need to be optimized. When we tune our hyperparameters using the search methods to be compared the one that gives us the most optimal set of hyperparameters would be the one for which our algorithm will be the most accurate.

For regression algorithms there is no such things as accuracy scores as regression algorithms give us a value and not a class and there is no way for us to directly measure how accurate the prediction was. We do, however, get few metrics which can help us compare as to which algorithm works the best. These metrics include the precision, recall values, F1 scores, R2 scores, etc. We have chosen the R2 score as our evaluation metric. The higher the R2 score the better is the regression algorithm and therefore the more optimized is the set of hyperparameters of the algorithm. We can, hence, say that the search method that gives us a higher R2 score value has tuned the hyperparameters better.

Apart from just a score, we also need to ensure that we are getting a decent score in an efficient manner, i.e., the time taken by a search method for hyperparameters optimization is also comparable to these scores.

For our project, since we are benchmarking for well over a hundred classification as well as regression datasets for each algorithm it is better, we average out the respective scores as well as the times of execution over all these datasets.

3.4 Method-wise average graphs

We make two graphs. In one graph we have grid search, random search and Bayesian search on X coordinates and the average scores on the Y coordinates. In the other graph we have average of times of execution on Y coordinates and grid search, random search, and Bayesian search on X coordinates. By plotting these two graphs it makes it easier for us to visualize which of the search methods is working better with regards to score and which one of the search methods is working better with regards to the time of execution.

Chapter 4

Testing / Results

This section describes the experiments to train and test the given dataset for all the classification and regression algorithms, as well as the performance of different search methods.

4.1 Setup

The models were built, and training and testing were implemented in Python version 3.7 using Scikit-learn package version 1.1.2 on a single Nvidia GTX GPU with 6 gigabytes of memory. The system memory is 16 gigabytes. All the datasets used in the project for benchmarking have been taken from the PMLB library. For getting the different models, we use the Scikit-learn package. To get all the arrays and random value sets, NumPy has been used. Pandas' library is used to convert the given list into a data series for better plotting of graphs. We have used the datetime module to get the start time and end time of a search to calculate its execution time. The matplotlib library version 3.5.3 is used for plotting all the graphs.

4.2 Model Training

Before training the model, we first need to get the dataset and split it into variables and targets. Along with that, we need to create a list of parameters which contains all the possible values of the hyperparameters which are to be tuned. For example, in RF classifier, the hyperparameters that need to be tuned are: 'n_estimators', 'criterion' and 'max_features'. We first create the parameter grid for all three search methods and pass the grid on to the method along with

other parameters such as the value of cv(Cross validation), verbose(tells us about the process ongoing), n_jobs(which cores to perform the task), and the scoring method. For random and bayes search methods we also pass a parameter named n _iter which tells the number of times the search process is to be iterated.

We then fetch the data from the dataset and store it in the format of X,y(variables and targets) and then split it into training and testing data. After this we fit the training data on to our search models. After the training data has been fit into the model, we then calculate the score for the test data and store it in the list corresponding to the specific dataset. We also calculate the time taken in fitting the training data and calculating the score of the testing data and store it in another list.

4.3 Statistical Computations

Once the training data has been fit, we calculate the respective score, be it accuracy or R2, of the model and we store this data in three different lists, one list each for grid, random and Bayesian search scores, with each value corresponding to one dataset. In order to understand the overall trend of the model and its optimization, we find out the average of these scores in the list. We save the average of each list, corresponding to a search method, in three variables. We also compute the time taken by each search method on a dataset and correspondingly save this time to three lists. We calculate the average of the time durations of each model's corresponding list (in seconds) and save them to three new variables.

4.4 Plotting and Comparisons

We now have the variables corresponding to the scores as well as the time durations of each search methods and we are now ready to plot them into a bar graph. We already have the x labels prepared, i.e., grid search, random search, and Bayesian optimization. We now need the y labels for which we arrange the three average scores and time durations correspondingly into a list. To get a better

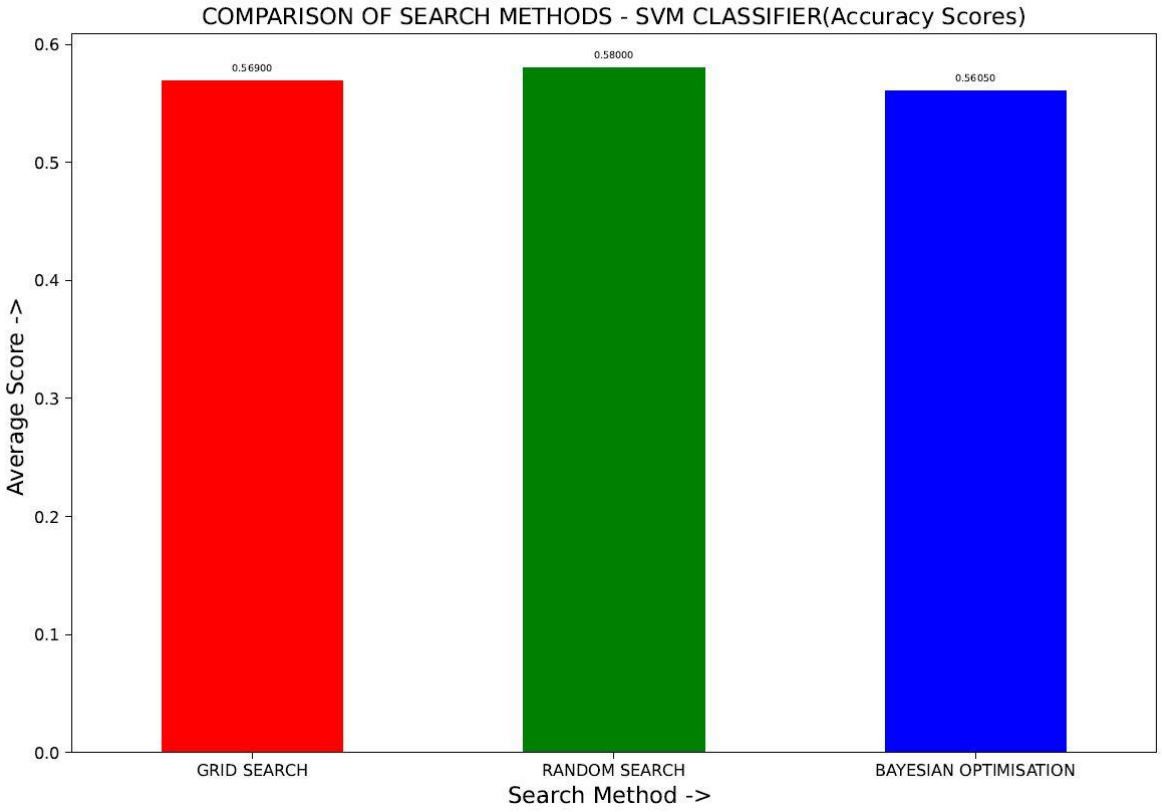
plot, we convert the lists into pandas series and then put them as the y labels of the respective graphs. We then define the font size, size of the figure, name of the labels, axes, and other important aspects of the graphs for a better and clearer understanding and visualization. We also declare a function which helps us to properly label the individual bars showing the exact value on top of each bar. We then plot the search method versus average scores and search method versus average time (in seconds) graphs with the aforementioned configurations.

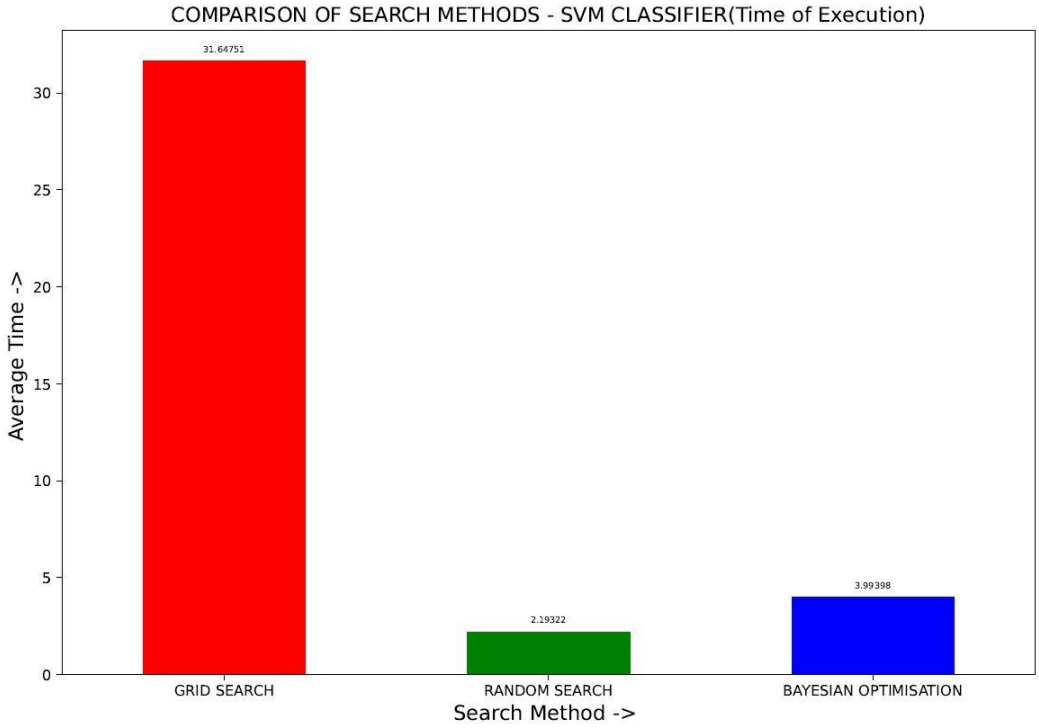
4.5 Results

The following are the average score vs method and the average time vs method graphs of all the classifier models, followed by those of regression models. The unit of the average time of execution of each model(y axis) of the second plot is seconds.

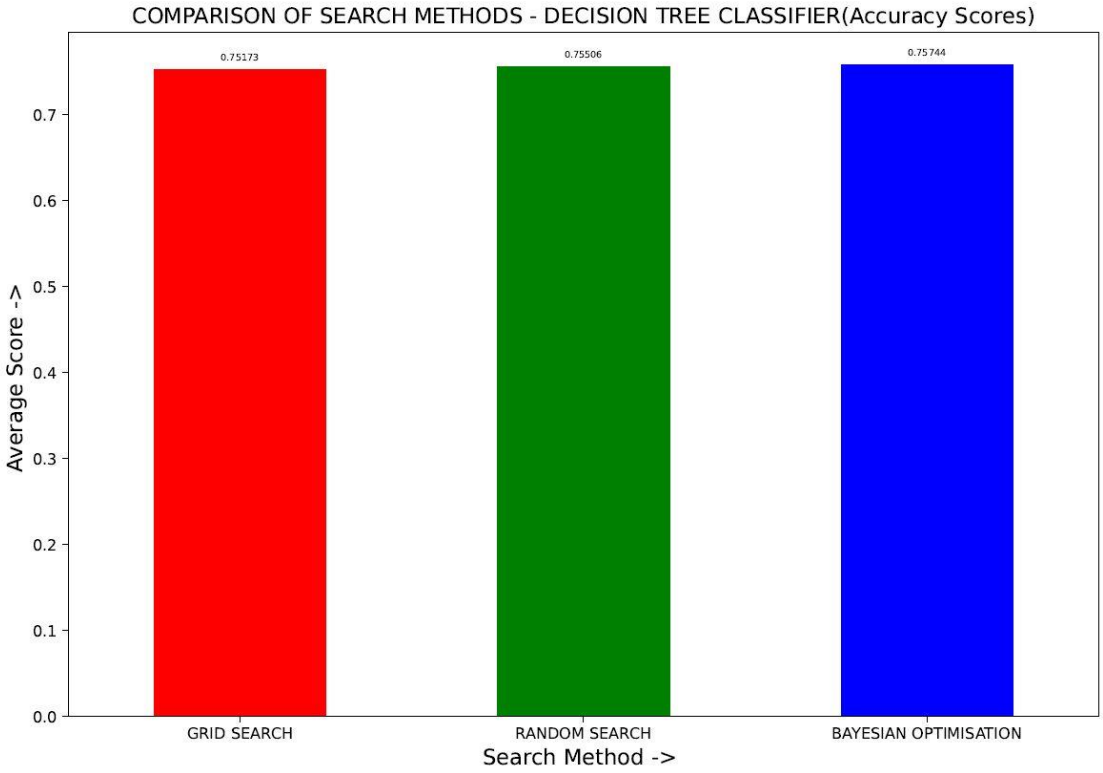
CLASSIFIER ALGORITHMS:

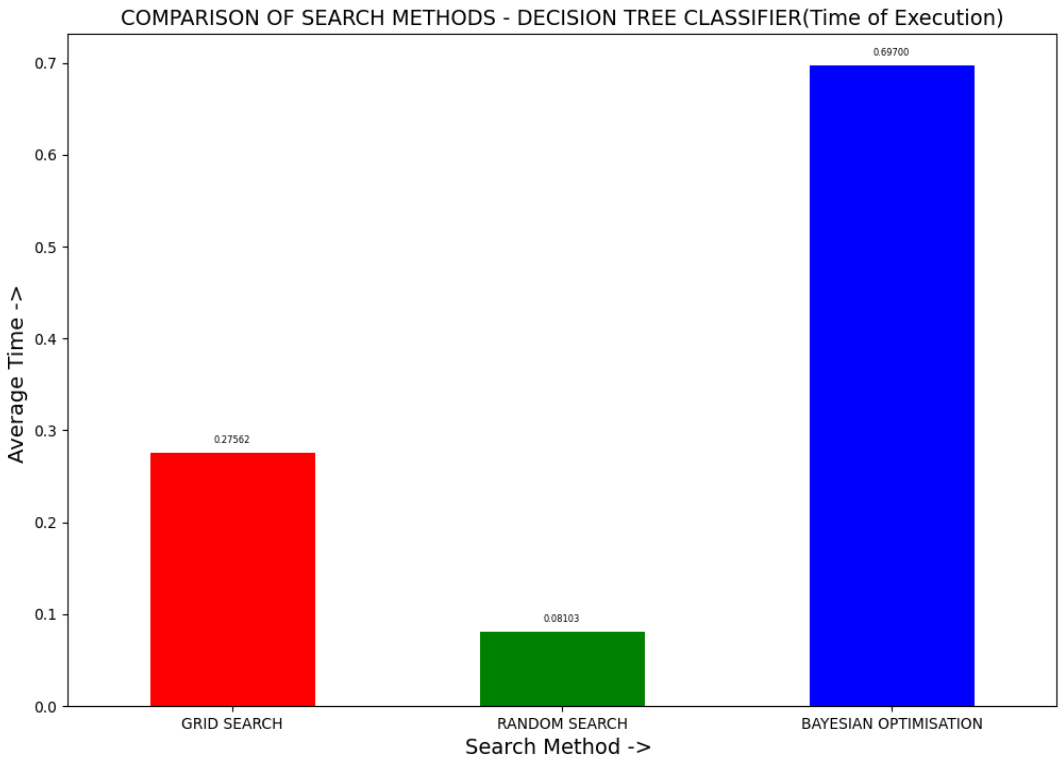
1.SVM Classifier:



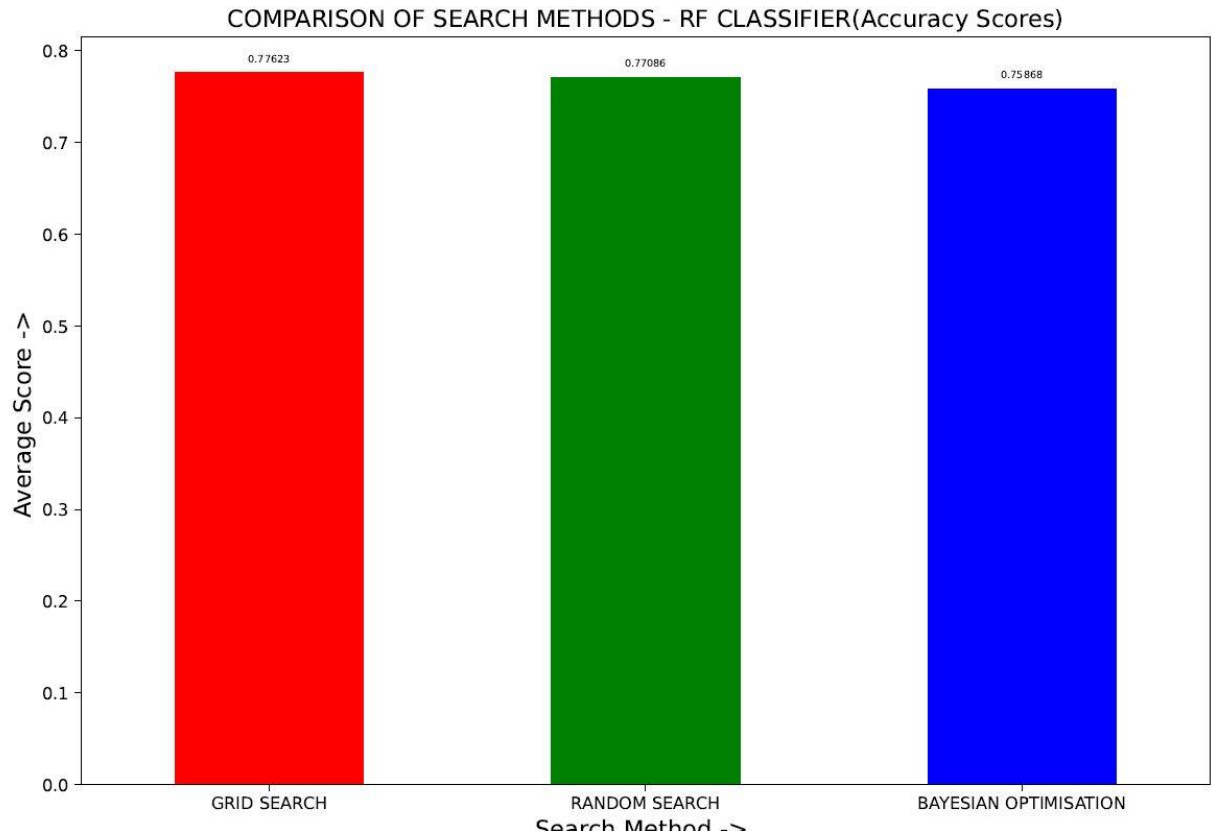


2.Decision Tree Classifier:

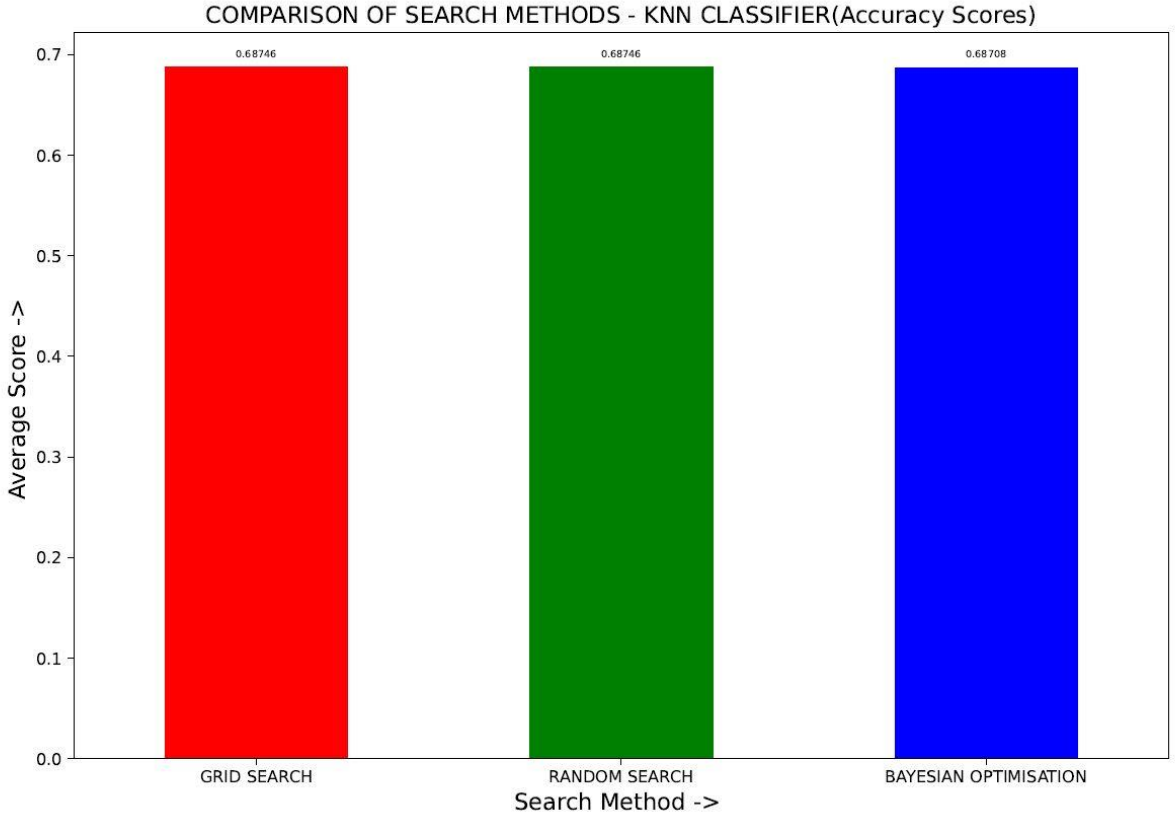


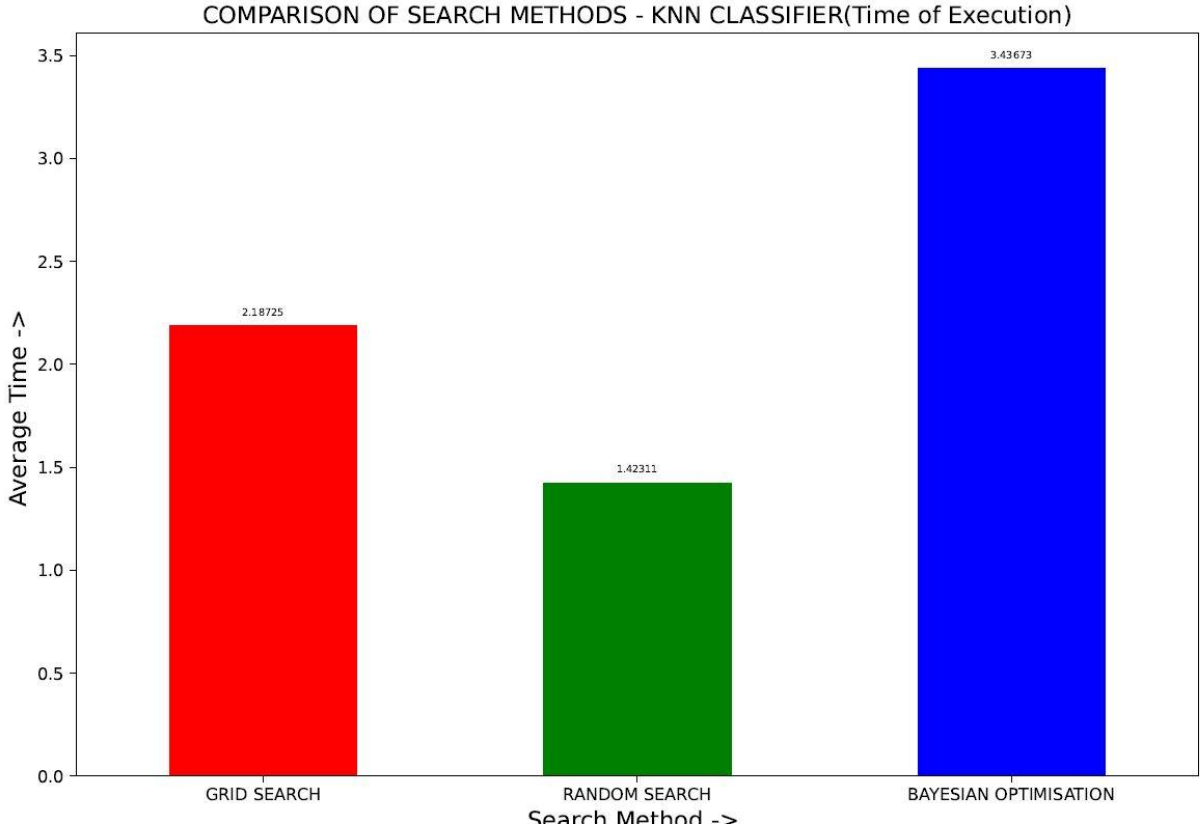


3.Random Forest Classifier:



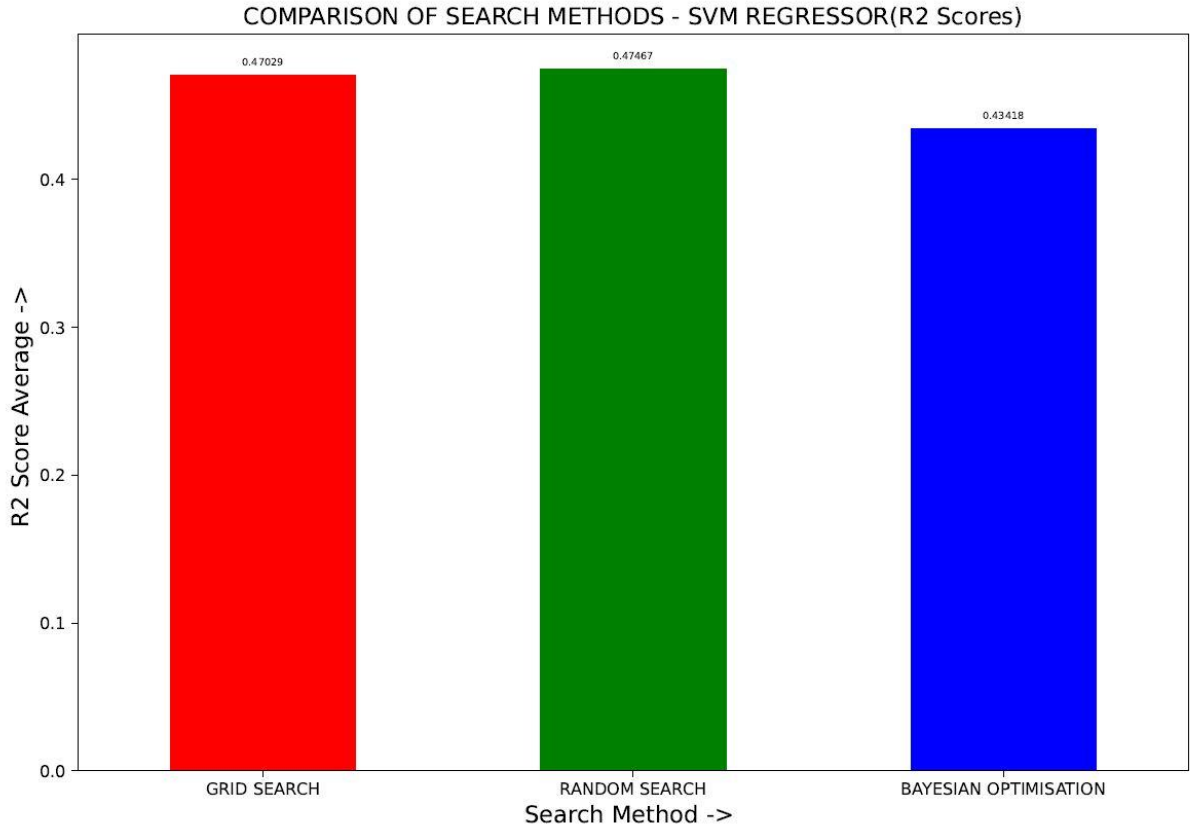
4.KNN Classifier:

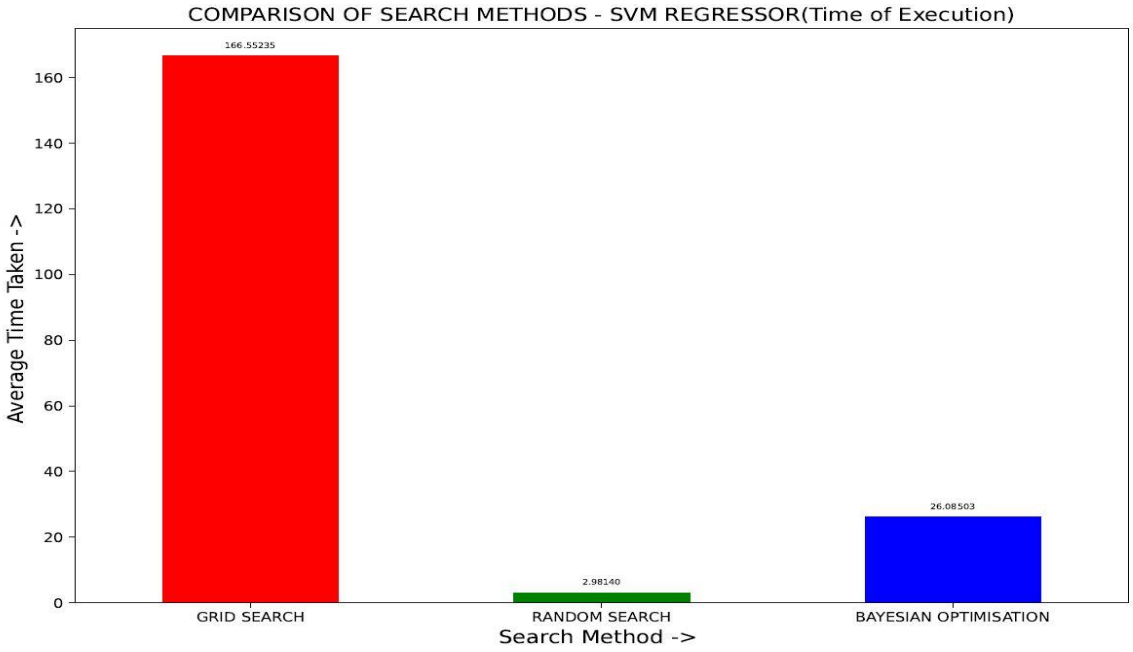




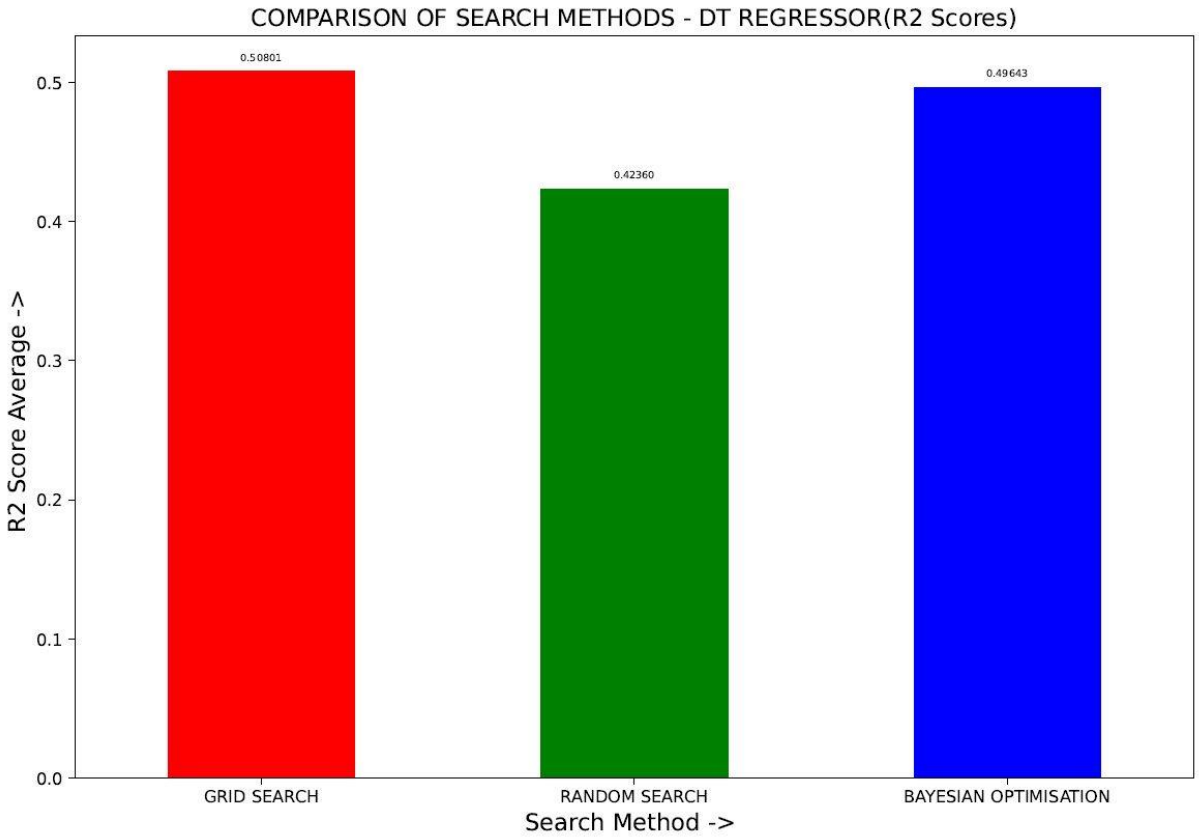
REGRESSION ALGORITHMS:

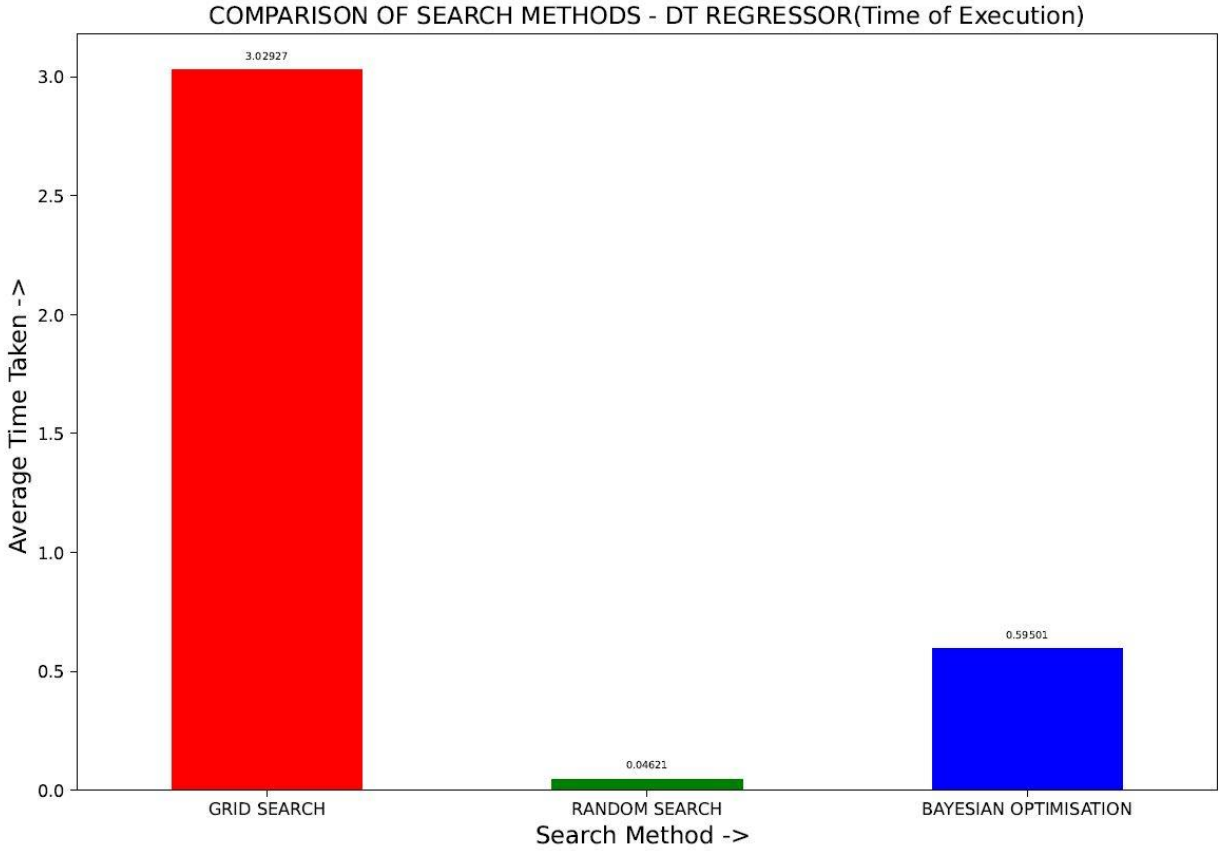
1.SVM Regressor:



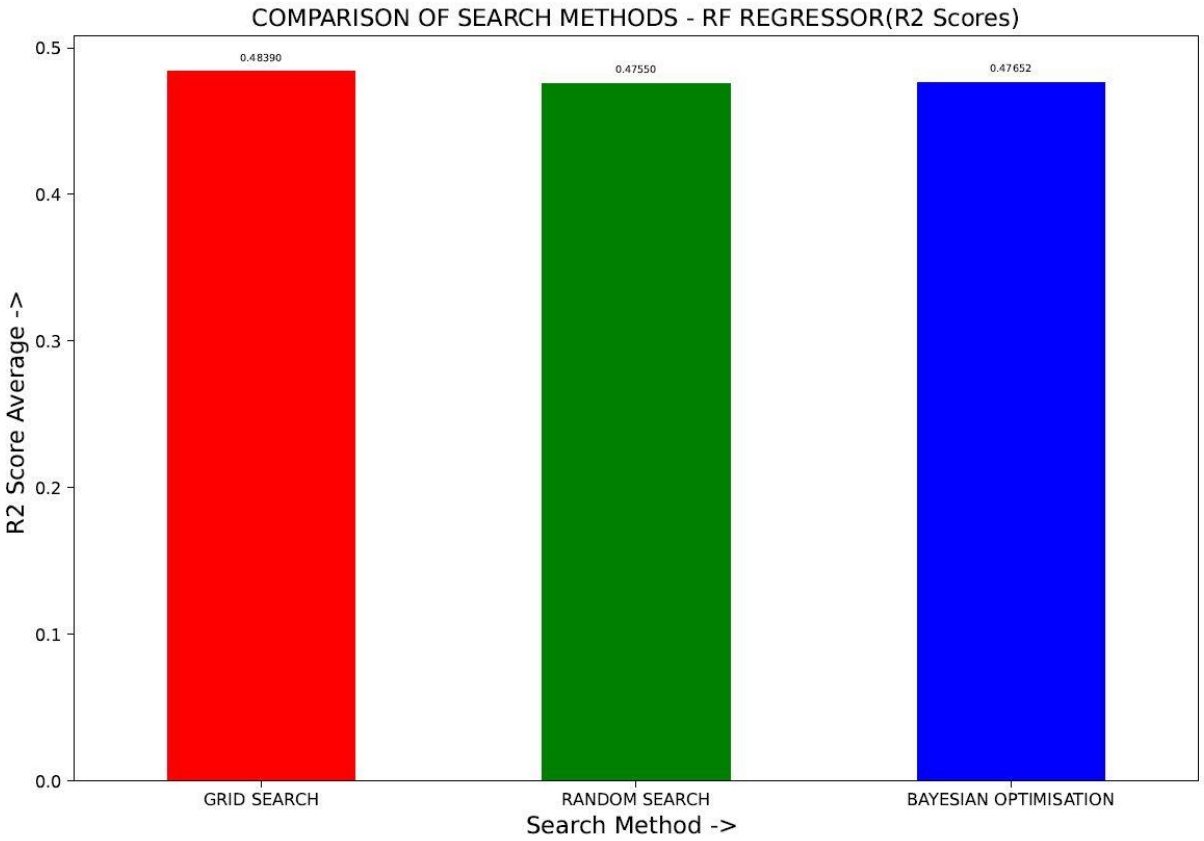


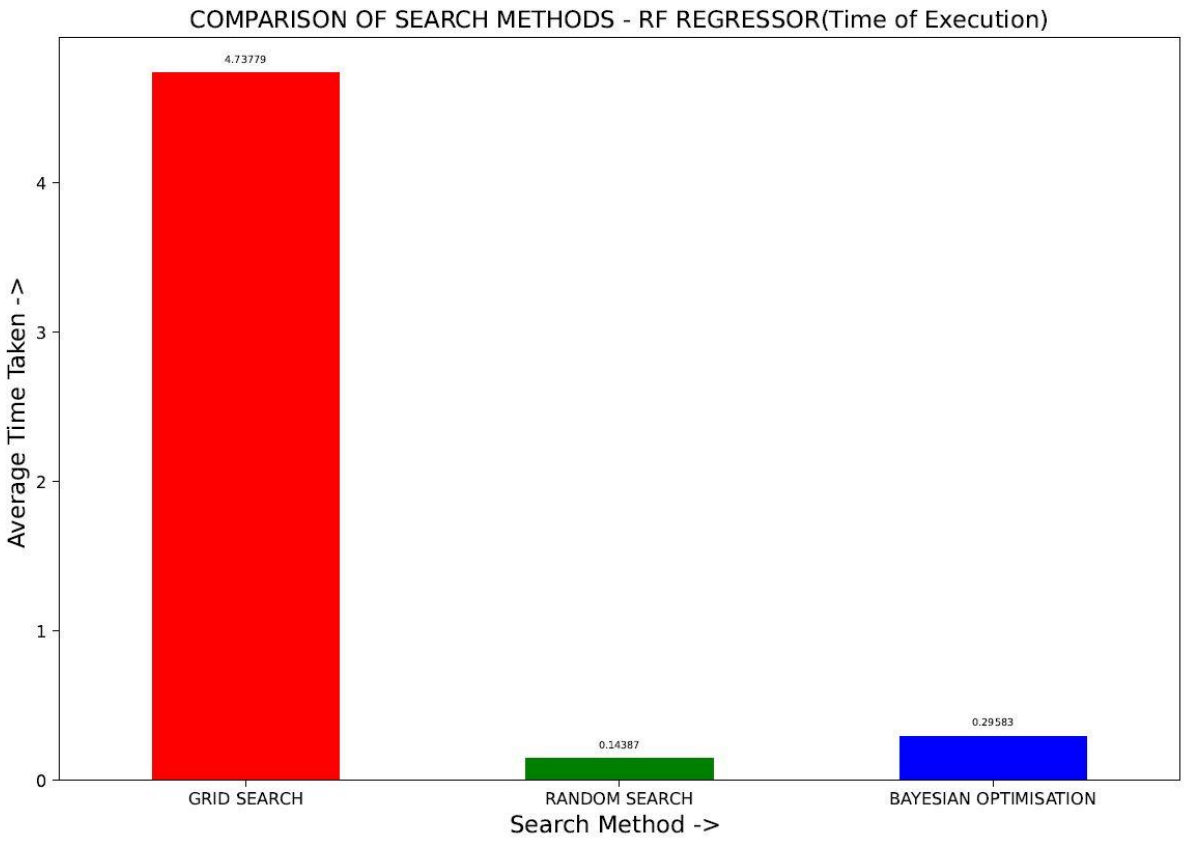
2.Decision Tree Regressor:



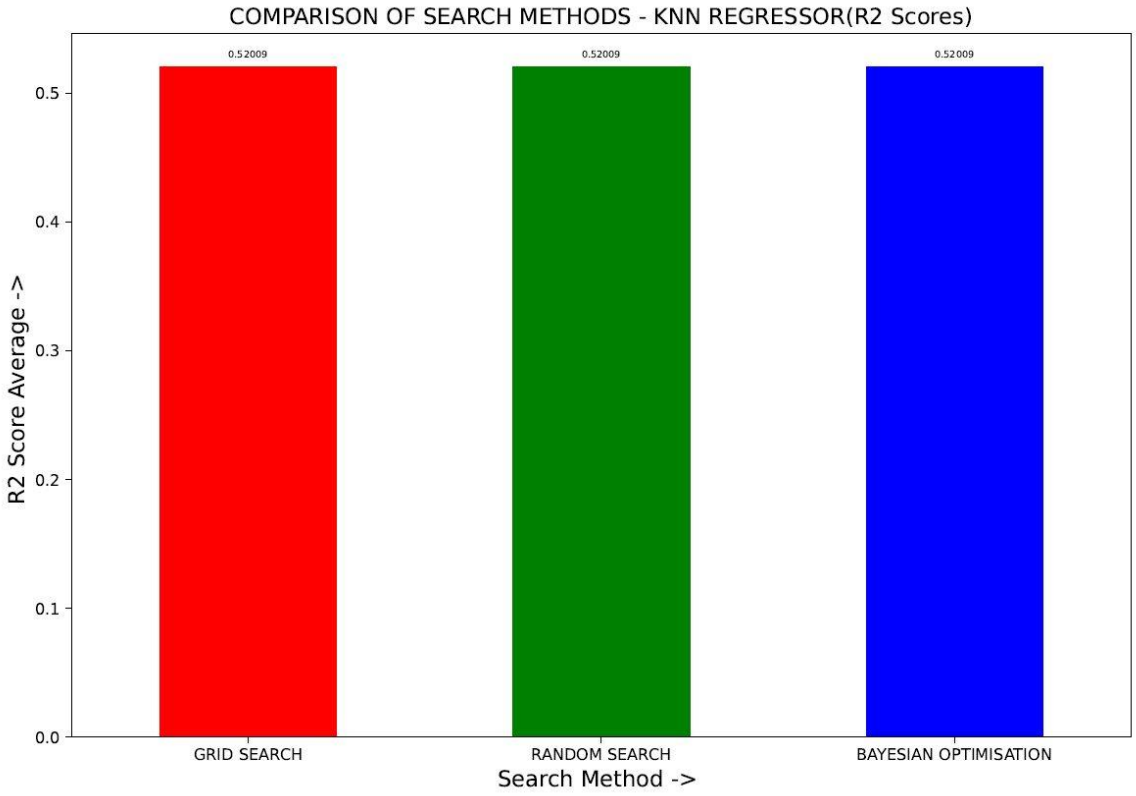


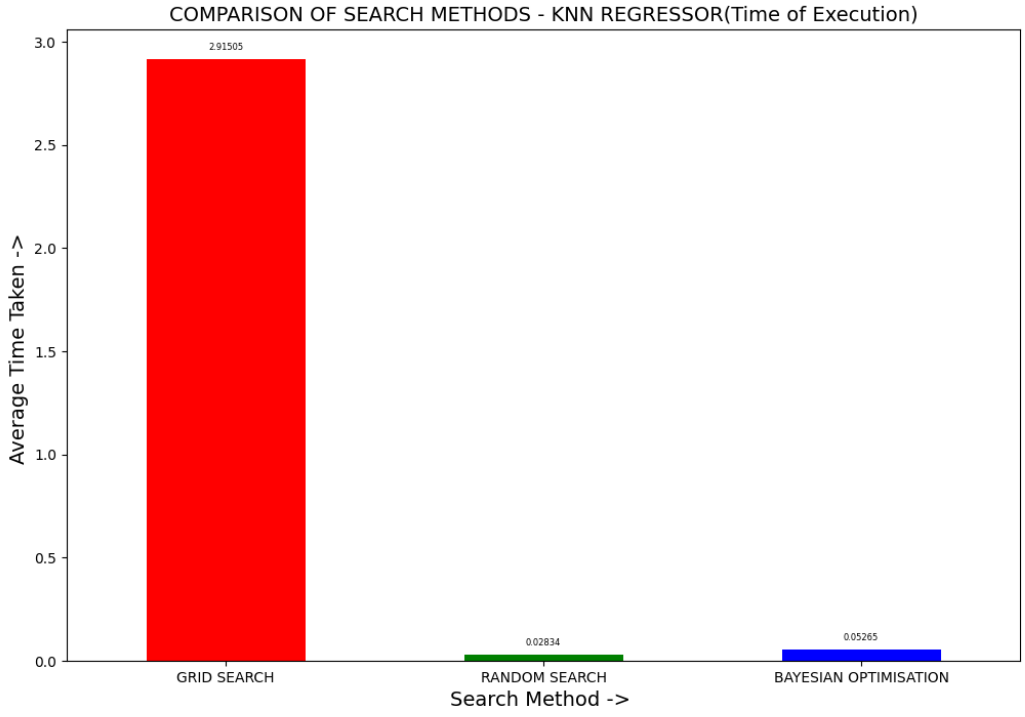
3. Random Forest Regressor:





4. KNN Regressor:





As we can see, in most cases, random search has been an obvious choice. In most of the models, the three search methods give us almost comparable or similar average score over the whole dataset. But it is quite visible that grid search takes much more time as well has really high computational costs, followed by Bayesian optimization and Random search not necessarily in order. When the time of execution is compared, Random search and Bayesian Optimization overshadow Grid search and there is mostly no competition at all. When we compare Random Search and Bayesian Optimization, however, things get interesting. Obvious expectation is that random search will take less time to execute but will perform as well in terms of score. But surprisingly, random search is, on an average, giving out scores comparable to Bayesian optimization, sometimes better, and taking less time as well. This is the case in most of the classifier models. In Regression algorithms, Bayesian optimization gives out the best combination of score and time of execution in most of the models. Random search follows and grid search takes much longer time and the scores, despite being comparable or even better than other two in some cases, cannot compensate for the high cost of execution.

Chapter 5

Conclusion and Discussion

In this project, we compared the three different methods of Hyperparameter optimization, based on their efficiency and accuracy. We first trained our models for classification and regression tasks respectively, along with defining the hyperparameter grid for the search methods to search and optimize. We then applied the three search methods on the same dataset, one after the other and kept track of the score and time taken by each method on each dataset. We then found out the average of these scores and time durations, so as to generalize the trend of the search methods over the classification/ regression algorithm. We then created two bar plots comparing the three search methods and analyzed them hence. We observed that each of these models make a very strong case for themselves given their best-case scenarios. We saw that when the time and cost of execution is of less concern and utmost importance is given to the method that gives us the best accuracy score, grid search excels in that regard. When it comes to taking way lesser time when compared to grid search and still giving accuracy scores approximately the same, and just a tad bit less, random search excels in that case. Whenever there is a case where the most efficient as well as fairly accurate tuning is needed, Bayesian Optimization is the way to go. Bayesian Optimization is notably better for regression algorithms and the difference is very eminent in the regression algorithms. While Grid Search takes a very long time and has very high computational costs, Bayesian Optimization gives us an R2 score very much comparable to that in way less time as well as has less computational costs.

Appendix

1. Description of the code repository

The codebase of the project is maintained on GitHub.

The link to the GitHub repository:

https://github.com/chinmaydas23/ml_search_methods_comparison

The repository contains separate codes implementing each of Grid, Random and Bayesian Search on four different classification and four different regression models, on all datasets of PMLB. Along with this a single algorithm, applying all three searches on the same datasets.

The “Plots” folder in the repository contains all the plots used in the report here, with each method and model named distinctively.

The repository can be used to run the main code and obtain the results mentioned above as follows:

- Install all the necessary packages from the requirements section and the codes mentioned in the repository.
- Add the "main.py" file to your local python environment and run it. That is all there is to be done.
- The file will then run the comparison of all the methods, taking in all four classification models first, and the regression models second.
- It will end once all the plots comparing the methods for each model has been saved to your project workspace.

2.Code

2.1 main.py file

```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.model_selection import ParameterGrid
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier,
DecisionTreeRegressor
from skopt import BayesSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn import neighbors, metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
from pmlb import dataset_names, classification_dataset_names,
regression_dataset_names
from pmlb import fetch_data
import matplotlib.pyplot as plt
from datetime import datetime, timedelta

GridTestScore = []
RandTestScore = []
BayesTestScore = []

GridTimeTaken = []
RandTimeTaken = []
BayesTimeTaken = []

# Creating the parameter grids of each model. These parameters
will get optimized
RFparams = {'n_estimators': np.arange(5, 100, 5),
            'max_features': np.arange(0.1, 0.5, 0.05)}
DTCparams = {'criterion': ('gini', 'entropy'),
            'splitter': ('best', 'random'),
            'max_features': np.arange(0.1, 0.5, 0.05) }
DTRparams = {'criterion': ('squared_error', 'friedman_mse',
'absolute_error', 'poisson'),
            'splitter': ('best', 'random'),
            'max_features': np.arange(0.1, 0.5, 0.05), }
KNparams = {'weights': ('uniform', 'distance'),
                # 'n_neighbours' : np.arange(1,5,1),
                'algorithm' : ('auto', 'ball_tree', 'kd_tree',
```

```
'brute')
            }
SVparams = { 'kernel': ('linear', 'poly', 'rbf', 'sigmoid'),
          'C': [1,10,20],
          'degree': [3,8],
          'coef0': [0.01,10,0.5],
          'gamma': ('auto','scale') }

modelnames = ["RF CLASSIFIER", "SVM CLASSIFIER", "DT CLASSIFIER",
"KNN CLASSIFIER", "RF REGRESSOR", "SVM REGRESSOR",
              "DT REGRESSOR", "KNN REGRESSOR"]

#Selecting the model out of the four:
for i in range(8):
    if i == 0:
        model = RandomForestClassifier(random_state=0)
        params = RFparams
    elif i == 1:
        model = svm.SVC()
        params = SVparams
    elif i == 2:
        model = DecisionTreeClassifier(random_state=0)
        params = DTCparams
    elif i == 3:
        model = KNeighborsClassifier()
        params = KNparams
    elif i == 4:
        model = RandomForestRegressor(random_state=0)
        params = RFparams
    elif i == 5:
        model = svm.SVR()
        params = SVparams
    elif i == 6:
        model = DecisionTreeRegressor(random_state=0)
        params = DTRparams
    else:
        model = KNeighborsRegressor()
        params = KNparams

    if 0 <= i <= 3:
        # CLASSIFICATION Models:

        grid_search = GridSearchCV(model, params, cv=3,
scoring='accuracy', verbose=2, n_jobs=-1, error_score='raise')
        random_search = RandomizedSearchCV(model, params,
n_iter=10, cv=3, scoring='accuracy', verbose=2, n_jobs=-1)
        bayes_search = BayesSearchCV(model, params, n_iter=10,
cv=3, scoring='accuracy', verbose=2, n_jobs=-1)

        for classification_dataset in
classification_dataset_names[1:2]:
            # Read in the datasets and split them into
```

```
training/testing
            X, y = fetch_data(classification_dataset,
return_X_y=True)
            X_train, X_test, y_train, y_test =
train_test_split(X, y)
    elif i >= 4:
        # REGRESSION models:

        grid_search = GridSearchCV(model, params, cv=3,
scoring="r2", verbose=2, n_jobs=-1,error_score='raise')
        random_search = RandomizedSearchCV(model, params,
n_iter=5, cv=3, verbose=2, random_state=None, n_jobs=-1)
        bayes_search = BayesSearchCV(model, params, cv=3,
n_iter=5, scoring="r2", verbose=2, n_jobs=-1,random_state=None)

        for regression_dataset in regression_dataset_names[1:2]:
            X, y = fetch_data(regression_dataset,
return_X_y=True)
            X_train, X_test, y_train, y_test =
train_test_split(X, y)

    start_time = datetime.now()
    grid_search.fit(X_train, y_train)
    print("Best Params for Grid Search using "+modelnames[i]+"
model are:", grid_search.best_params_)
    print("Best Score for Grid Search using "+modelnames[i]+"
model is:", grid_search.best_score_)
    GridTestScore.append(grid_search.score(X_test,y_test))
    end_time = datetime.now()
    duration = str(end_time - start_time)
    print('Duration for Grid Search: {}'.format(end_time -
start_time))
    GridTimeTaken.append(duration)

    start_time = datetime.now()
    random_search.fit(X_train, y_train)
    print("Best Params for dataset By Random Search using
"+modelnames[i]+" model are:", random_search.best_params_)
    print("Best Score for Random Search using "+modelnames[i]+"
model is:", random_search.best_score_)
    RandTestScore.append(random_search.score(X_test,y_test))
    end_time = datetime.now()
    duration = str(end_time - start_time)
    print('Duration for Random Search: {}'.format(end_time -
start_time))
    RandTimeTaken.append(duration)

    start_time = datetime.now()
    bayes_search.fit(X_train, y_train)
    print("Best Params for dataset By Bayes Search using
"+modelnames[i]+" model are:", bayes_search.best_params_)
    print("Best Score for Bayes Search using "+modelnames[i]+"
```

```
model is:", bayes_search.best_score_)
    BayesTestScore.append(bayes_search.score(X_test,y_test))
    end_time = datetime.now()
    duration = str(end_time - start_time)
    print('Duration for Bayes Search: {}'.format(end_time -
start_time))
    BayesTimeTaken.append(duration)

    # Getting the average value of the scores list for each
search method
    gavg = round(sum(GridTestScore)/len(GridTestScore), 5)
    ravg = round(sum(RandTestScore)/len(RandTestScore), 5)
    bavg = round(sum(BayesTestScore)/len(BayesTestScore), 5)
    print(gavg)
    print(ravg)
    print(bavg)

    # Getting the average value of the time durations list for
each search method
    gtavg = (timedelta(seconds=sum(map(lambda f: float(f[0])*3600
+ float(f[1])*60 + float(f[2]), map(lambda f: f.split(':'),
GridTimeTaken)))/len(GridTimeTaken)))
    rtavg = (timedelta(seconds=sum(map(lambda f: float(f[0])*3600
+ float(f[1])*60 + float(f[2]), map(lambda f: f.split(':'),
RandTimeTaken)))/len(RandTimeTaken)))
    btavg = (timedelta(seconds=sum(map(lambda f: float(f[0])*3600
+ float(f[1])*60 + float(f[2]), map(lambda f: f.split(':'),
BayesTimeTaken)))/len(RandTimeTaken)))
    print(gtavg)
    print(rtavg)
    print(btavg)

    # Getting the plotting parameters correct and ready
    SMALL_SIZE = 6
    MEDIUM_SIZE = 10
    BIG_SIZE = 14
    BIGGER_SIZE = 20

    plt.rc('font', size=SMALL_SIZE)          # controls default
text sizes
    plt.rc('axes', titlesize=BIG_SIZE)     # fontsize of the axes
title
    plt.rc('axes', labelsize=BIG_SIZE)     # fontsize of the x and
y labels
    plt.rc('xtick', labelsize=MEDIUM_SIZE)     # fontsize of the
tick labels
    plt.rc('ytick', labelsize=MEDIUM_SIZE)     # fontsize of the
tick labels
    plt.rc('legend', fontsize=10)     # legend fontsize
    plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the
figure title
```

```
    # Defining the axes and values thereof
    xlabels = ['GRID SEARCH', 'RANDOM SEARCH', 'BAYESIAN
OPTIMISATION']

    # Getting the value of each bar to show on top of it
    def add_value_labels(ax, spacing=5):
        """Add labels to the end of each bar in a bar chart."""

        # For each bar: Place a label
        for rect in ax.patches:
            # Get X and Y placement of label from rect.
            y_value = rect.get_height()
            x_value = rect.get_x() + rect.get_width() / 2
            va = 'bottom'
            # Use Y value as label and format number with four
decimal places
            label = "{:.5f}".format(y_value)
            # Create annotation
            ax.annotate(
                label,                      # Use `label` as
label
                (x_value, y_value),         # Place label at end
of the bar
                xytext=(0, spacing),              # Vertically
shift label by `space`
                textcoords="offset points", # Interpret `xytext`
as offset in points
                ha='center',                # Horizontally center
label
                va=va)                      # Vertically align
label


    width = 0.40  # the width of the bars
    avgscores = [gavg, ravg, bavg]
    avg_series1 = pd.Series(avgscores)
    plt.figure(figsize=(12,8))
    ax = avg_series1.plot(kind='bar', color= ['red', 'green',
'blue'])
    ax.set_xlabel('Search Method -> ')
    ax.set_ylabel('Average Score -> ')
    ax.set_title("COMPARISON OF SEARCH METHODS - Accuracy Scores:
"+modelnames[i]+" ")
    ax.set_xticklabels(xlabels, rotation=0)
    add_value_labels(ax)
    plt.savefig(" AvgScoresPlot " + modelnames[i] + ".pdf",
dpi=100)
    plt.show()

    avgtimes = [gtavg.total_seconds(), rtavg.total_seconds(),
btavg.total_seconds()]
    avg_series2 = pd.Series(avgtimes)
```

```
    plt.figure(figsize=(12,8))
    ax2 = avg_series2.plot(kind='bar', color= ['red', 'green',
'blue'])
    ax2.set_xlabel('Search Method -> ')
    ax2.set_ylabel('Average Time -> ')
    ax2.set_title("COMPARISON OF SEARCH METHODS - Time of
Execution: "+modelnames[i] +" ")
    ax2.set_xticklabels(xlabels, rotation=0)
    add_value_labels(ax2)
    plt.savefig(" AvgTimesPlot " + modelnames[i] + ".pdf",
dpi=100)
    plt.show()
```

References

[1]. M.I. Jordan, T.M. Mitchell, Machine learning: Trends, perspectives, and prospects, Science 349 (2015) 255260. https://doi.org/10.1126/science.aaa8415.

[2]. N. Decastro-Garca, . L. Muoz Castaeda, D. Escudero Garca, and M. V. Carriegos, Effect of the Sampling of a Dataset in the Hyperparameter Optimization Phase over the Efficiency of a Machine Learning Algorithm, Complexity 2019 (2019). https://doi.org/10.1155/2019/6278908.

[3]. G. Luo, A review of automatic selection methods for machine learning algorithms and hyper-parameter values, Netw. Model. Anal. Heal. Informatics Bioinforma. 5 (2016) 116. https://link.springer.com/article/10.1007/s13721-016-0125-6

[4]. S. Sun, Z. Cao, H. Zhu, J. Zhao, A Survey of Optimization Methods from a Machine Learning Perspective, arXiv preprint arXiv:1906.06821, (2019). https://arxiv.org/abs/1906.06821

[5].PMLB: https://academic.oup.com/bioinformatics/article/38/3/878/6408434 https://github.com/EpistasisLab/pmlb

[6]. Scikit-learn: https://scikit-learn.org/stable/supervised_learning.html https://scikit-optimize.github.io/stable/auto_examples/sklearn-gridsearchcv-replacement.html

[7]. Exploring Bayesian Optimization**:** Breaking Bayesian Optimization into small, sizeable chunks. - Apoorv Agnihotri, Nipun Batra. https://distill.pub/2020/bayesian-optimization/

[8]. A Tutorial on Bayesian Optimization Peter I. Frazier July 10, 2018 https://arxiv.org/pdf/1807.02811.pdf