MSc Report

SCHOOL OF COMPUTING SCIENCE AND DIGITAL MEDIA

Student Name: **DESMOND DUMEBI OJEI**	Matriculation Number: **1419100**
Supervisor: **DR.CIPRIAN ZAVOIANU**	Second Marker: **Prof. John McCall**
Course: **MSc Data Science**	
Project Title: **Using Geometrical Characteristics To Predict The Performance Of Multi-Objective Evolutionary Algorithms**	
Project Investigation [CMM012] ☐ MSc Project [CMM013] ☒ (Tick as appropriate)	Start Date: 25/06/20 Submission Date: 26/08/20

CONSENT

I agree ☒
I do not agree ☐

That the University shall be entitled to use any results, materials or other outcomes arising from my project work for the purposes of non-commercial teaching and research, including collaboration.

DECLARATION

I confirm:

- **That the work contained in this document has been composed solely by myself and that I have not made use of any unauthorised assistance.**
- **That the work has not been accepted in any previous application for a degree.**
- **All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.**

Student Signature: Desmond Ojei	Date Signed: 20/06/2020

This project is greatly dedicated to my family whose hard work and commitment has enabled to reach this point of my life. I don't know where I would be without their effort and I can only hope that my own efforts and determination make them proud.

"The larger the island of knowledge the longer the shoreline of wonder ",

Ralph Sockman

Table of Contents

Nomenclature

CART	Classification and Regression Trees
EDA	Exploratory Data Analysis
IDE	Integrated Development Environment
KNN	K-Nearest Neighbour
MOEA	Multi-Object Evolutionary Algorithm
MOEA/D	Multi-Object Evolutionary Algorithm based on Decomposition
MO-ICOP	Multi-Objective Continuous Optimization Problems
MOO	Multi – Object Optimization
MOOP	Multi-Objective Optimization Problem
NSGA-II	Non-Dominated Sorting Genetic Algorithm II
SVM	Support Vector Machine

Tables of Figures

I Abstract

In the last decade, multi-objective evolutionary algorithms (MOEAs) have attracted a lot of research and they are still regarded as one of the hottest research areas in the evolutionary computation field to improve optimization search. However, one drawback of MOEAs is that optimization run can be very computationally expensive and time-consuming as several algorithm known to generally perform well need to be tried out a number of times in other to discover the best one for a new given problem.

This project focuses on constructing, implementing and evaluating geometric descriptors of seed sets that define multi-objective continuous optimization problems (MO-ICOPs) (i.e. convex and concave hull). The following parameters such as area and number of points of each geometric descriptor will be calculated and added to a dataset which consist of given values such as the dimensionality of the problem and the smoothness of the fitness landscape. The main focus will be to use machine learning to correctly predict the most suitable algorithm performance across a given set of problems. In the future, this can lead to algorithmic improvements and the successful applications of MOEA solvers on new problems with less tuning and computational intensity.

Keywords: multi-objective evolutionary algorithms - multi-objective continuous optimization problems - machine learning ·-convex hull - concave hull

II Acknowledgement

Firstly, I would like to thank the almighty God for the strength to carry out this project particularly during this challenging period engulfed in the world today.

Also, thanks to my project supervisor Dr.Ciprian Zavoianu who was outstanding with his support and effort by providing me with useful suggestions, whilst leading me through the right path when necessary during the entire project process. Lastly, I am grateful to my family and my course mates for their love and support towards me in full faith during this course.

1 Introduction

This chapter describes the motivation in conjunction with the aim and objectives of this project.

1.1 Motivation

Multi-Objective Optimization (MOO) simply means searching for the optimal solution values of multiple desired goals. This optimization process has grown immensely as the preferable approach when dealing with sustainability problems and decision making. Multi-objective optimization problems surface in many fields, which include economics, logistics and engineering when multiple optimal decisions require to be taken. For example, creating a new component might include minimizing the weight while the strength is maximized or making a business decision which might involve maximizing the expected return while minimizing the risk involved. The motivation of using the evolutionary multi-objective optimization is because in this optimization process when compared to the others is said to not require complex equations, which in turn simplifies the problem. Multi-Object Evolutionary Algorithms (MOEAs) are also widely used for solving real-world multi-objective problems. In the case of multiple conflicting objectives of a problem, MOEA can be applied. This creates a set of non-dominated solutions at the end of a run, known as the Pareto front. The main aim of MOEAs is to improve the convergence towards the Pareto front and diversify the solutions in the Pareto set.

1.2 Project Structure

- Chapter 1 reviews the project motivation, project aims and objective, research questions and finally the summary of the project investigation.
- Chapter 3 reviews the design methodology and implementation on the project.
- Chapter 4 reviews the result and analysis of the project.
- Chapter 5 reviews the discussion, conclusion and future work on the proposed project execution.
- Chapter 6 reviews the bibliography.
- Chapter 7 reviews the appendix of the project which consist of the project plan, RStudo screen shots of RStudio codes for each experiment.

1.3 Project Aim and Objectives

The aim of this project is to design, implementing and evaluate strategies to construct relevant geometric descriptors for MO-ICOPs (Zăvoianu, Lacroix, & McCall, 2020) (e.g., convex hull area and perimeter) and use this information to train and test using machine learning to correctly predict the most suitable algorithm performance across a given set of problems.

Objectives

- Designing and implementing geometric descriptors such as concave and convex hull for MO-ICOPs.
- Extracting information from these geometric descriptors such as number of points, area and perimeter.
- Executing Exploratory Data analysis (EDA) to visualize and gain more insight on the relationships between these features
- The features will be used to train and test a classification algorithm using machine learning to correctly predict the most suitable algorithm performance across a given set of problems.

1.4 Research Questions

The project is focused on evaluating the performances of machine learning models of two datasets, with and without geometric descriptor features. This is directed towards answering the two questions below:

- How did the trained models model perform when presented with an unseen dataset?
- How well the models are able to identify the most suitable MOEA algorithm for set of 1200 problems?

1.5 Summary of Project Investigation

In the world today due to an increasing competition among organizations in different fields, with the aim to maximize profits and minimize losses, there has been an increasing interest in application of MOEA in other to solve multiple conflicting objectives of a problem. The definition of the Multi-Objective Optimization Problems (MOOPs) aided to develop the aims and objectives that leads to the development and motivation of the project.

The literature overview was done to further understand the origin and the mechanism of optimization algorithms, types of optimization algorithms which brings this project to MOOP and methods which can be used to tackle these optimization problems such as the MOEA .Also, the preliminary investigation report further highlights the MO-ICOP dataset as proposed by

the authors (Zăvoianu, Lacroix, & McCall, 2020) which consist of 1200 problems and discusses using geometrical descriptors (i.e. concave and convex hull) to evaluate these problems. Finally, the next phase of the project will be using these geometric descriptors to predict the performance of the MOEAs.

2 Functional and Non-Functional Requirements

This chapter reviews the functional and non-functional requirements together with the project methodology proposed and lastly the legal, ethical issues and the risk considered with the project.

2.1 Functional Requirements

- The system will allow users to construct a geometric descriptors (Concave and Convex hull) with the seed sets that define MO-ICOPs.
- The system will allow users to extract information from these geometric descriptors.
- The system will allow users to create a new dataset consisting of the area of the geometric descriptors and number of points of the geometric descriptors.
- The system will allow tools for data exploration to visualize the relationships in the dataset in RStudio.
- The system allows training a classification (supervised) machine learning algorithm in RStudio to find the best algorithm which can accurately learn and classify the most suitable MOEAs such as NSGA-II (Deb, Pratap, Agarwal, & Meyarivan, 2002) and MOEA/D (Zhang & Li, 2007) based on the historical dataset.

2.2 Non-Functional Requirements for the code

- The application would be able to run in an environment with low computational resources.
- The code will be well commented to guide the user.
- The application can successfully be run in RStudio version of 1.2.5033.
- The project would be developed for 3 months from 20th May 2020 to 26th August 2020 and would be tested and evaluated.

2.3 Legal and Ethical Issues

This project does not pose any trace of risk to any individual. Also, I am taking full responsibility in maintaining academic honesty for each resource and library used in this project. Also, the software application used to develop this project are considered free and do not require licensing as no software will be used illegally. Finally, the main aim along the process of this project will be to further sharpen my skills in the subject area, as an aspiring data scientist and ensure the ethical rules are followed as highlighted by the British Computing Society.

2.4 Risk

This project was linked with potential risks and mitigation plans following the investigation report. However, one of the risks highlighted had to be reconsidered with an updated mitigation plan as the duration time of the project stretched.

Category/Sub -Category	Descriptions	Impact Descriptions	Mitigation Plan
People Experience	Lack of knowledge on ML Algorithm and EDA. Lack of Knowledge on R and ggplot. Lack of knowledge of Genetic Algorithms, Multi Object Optimization, Concepts of concave and convex hull	Failure to produce machine learning solutions and Visualisation. Failure to deliver working solution.	Undertake training, such as online training tutorials and in depth background research study.
Schedule/Time	The project was fixed for a duration of 3 months. However, due to some low computation efficiency, change of parameters and tuning of ML models, affected the duration of the project.	Inability to meet up deadlines on the due date.	In other to accommodate the low computation efficiency, and time taken to properly tune the ML models and analyse the result, a week extension of the project was requested to ensure the aims and objectives report were accomplished.
Technical Software	Loss of project work done due to system malfunctioning.	Failure to finish project and meet up deadline.	Ensure up to date back up storage of project in cloud for easy accessibility.

	Computational intensity which requires at least a core i5 processor, storage space of 256GB.	Time consuming , as time is essential and utilized in this project	Using a system with the specified details will make the work on the project efficiently.

3 Design Methodology and Implementation

The steps adopted to carry out this project are explained in this section. These steps include various techniques taken to collect data, process the data and finally apply machine learning methods to gain more insight of the data.

The Figure 1 below shows the structure of the project and the attributes in the separate datasets analysed.

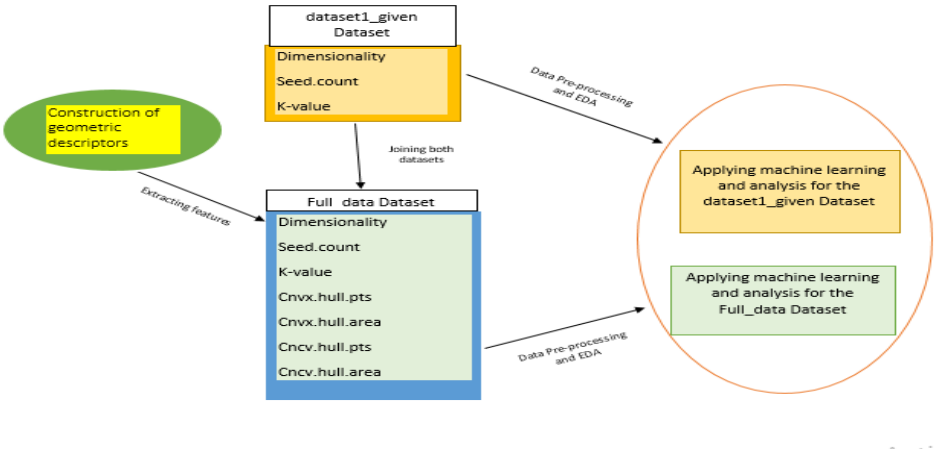

Figure 1: Project flow chart diagram

The steps adopted to carry out this project include:

- Understanding the project scope which will guide in the research to work on the challenges of the project.
- An iterative process for example the “for loop” was used in RStudio to iterate through the 4 directories containing csv.files. Each csv. files contained two separate csv.file of seed sets which represents x and y axis.
- Inside the iterative process, the separate csv.files were joined and used to construct geometric descriptors known as concave and convex hull. There are several libraries that can be used to generate these geometric descriptors.
- Following was the creation of a new dataset which was based on the parameters attained from the constructed geometric descriptors, including both the Dimensionality and “k” value (Smoothness) which will be provided by my supervisor. The attributes will be joined together in one csv.file and this will be explored using EDA techniques to visualize and gain insight on the computed dataset
- Also, background research and study on the most suitable machine learning algorithm and techniques to predict class of the new dataset more accurately.
- Finally, evaluating and analysing the performance of the trained model by using evaluation metrics.

3.1 Hardware and Software

This section highlights the hardware and software used in this project

3.1.1 Hardware

Below are the following specifications of Hardware used for the implementation of the various steps delineated in this project.

Processor: Intel(R) Core(TM) i7-3687U CPU @ 2.10GHz 2.60GHz

Installed memory (RAM): 4.00 GB (3.87 GB usable)

Local Disc (C :): 475GB

System type: 64-bit Operating System, x64-based processor

3.1.2 Software

The software in conjunction with the libraries used for this project are as follows:

RStudio 1.2.5033: R programming language is classified as a combination of software facilities for data manipulation, calculation and graphical display for its interactive statistical environment (Venables, Smith, & Team, 2020).These software facilities are the key reasons

this programming language is used in the project. Also, R is available for free whereby users can compile and run this language on several operating systems such as Mac OS X, Windows and Linux. Finally, I chose this language for this project to perfect my skills in R.

Integrated Development Environment (IDE)**:** Using RStudio which is known as an IDE, I was able to construct geometric descriptors, extract various parameters from these geometric descriptors and store these parameters in a csv.file, apply exploratory data analysis (EDA) and finally apply the most suitable machine learning algorithm to the data to evaluate the results.

3.1.3 R Libraries

R libraries and functions used for constructing the geometric descriptors, performing EDA and applying machine learning:

Splancs: This library generally is used for displaying and analysing spatial point of a patterned data. In this project the "npts" function which can only be used after loading up this library was used to compute the number of points for the geometric descriptors (i.e. convex and concave hull).

Concaveman: This library simply constructs a concave polygon based on set points given. The function used to perform this task is known as the "concaveman". Inside this function consist of parameters such as: points and concavity.

The "points" parameter simply defines the set points which you need to construct the concave hull which can be represented as a matrix of co-ordinates.

The "concavity" is a tuning parameter used to measure the degree of concavity. I tuned the concavity, using concavity from the range of 1-6 and I found out that when the concavity is below 2, it results to self-intersecting polygon which are not concave polygons. Furthermore, when the concavity was increased from 3 to 6 this resulted to more of a convex hull. After trial and testing the concavity of 2 was chosen as this represented the concave hull which will be used in this project.

magicfor: The "magicfor" packages stores printed values within a for loop function and by using the function known as "magic_result_as_dataframe", this simply organises this stored values in a data frame automatically.

ggplot2: This is a library which is used in R for creating graphics for visualization. The function in this library is called "ggplot". This was used in this project to visualise the proportions of the target class.

caret: This library is used for plotting and training both classification and regression models. This library was loaded in this project to run a classification model known as K-Nearest Neighbour (KNN) (Jabbar, Deekshatulua, & Chandra, 2013) and the Support Vector Machine (SVM) (Awad & Khanna, 2015).

DataExplorer: This library was used for data exploration in this project, with the use of functions such as: plot_missing.
The "plot_missing" function is used for checking for missing data on the dataset

corrplot: This library is used to graphically display correlation matrix of the attributes in the dataset set.

3.1.4 Construction of the geometric descriptors

This section consist of an in depth explanation on the steps taken to construct the geometric descriptors

3.1.4.1 Constructing the Convex and Concave hull

The complete R code for the construction of the geometric descriptors (i.e. convex and concave hull) can be found in the appendix Section 7.2. However, below are the steps taken to achieve these geometric descriptors in segments.
Firstly, as seen from the Figure 2 working directory was set using the "setwd" function inbuilt in RStudio and the "list.files" function was used in other to list only csv.files , which is the reason the parameter of this function kwon as "pattern" was set to "*.csv". Also other parameters such as "recursive" was set to "TRUE" in other for the listing to be recursive into directories and finally the "full.names" parameter was set to "TRUE" which simply commands R to add the directory path to the file names in other to give a relative path.

```
setwd("C:\\Users\\hp\\Desktop\\ProjectDevelopment\\Original DATASETS
PROJECT\\PPSN_2020-5a25aad3acf974a2e398d3c4b4b515d80c98e139\\benchmarkInstances"
)

all_files <- list.files(pattern = "*.csv", recursive = TRUE,full.names = TRUE)
```

Figure 2: Step 1 (Geometric Descriptor Construction)

As seen from the Figure 3, using an iterative process for example the “for loop” in RStudio to iterate through the 4 directories containing csv.files. Each csv.files contained two separate csv.file of seed sets which represents x and y axis.

In other for RStudio to recognise that these files are csv.files , the “read.csv” function was used inside the “for loop” with the “header” parameter in this function set as “FALSE”, as the csv.files did not contain columns. Furthermore, after the “read.csv” function is used and assigned to a variable called ”data”, This data variable was put into the “data.frame” function which simply converts the data into a dataframe which makes it easier to be converted into a matrix using the “data.matrix” function in RStudio.

```
for (files in all_files){

 data<-read.csv(files,header = FALSE) #Reading csv files
 data <- data.frame(data)


 #Constructing Convex hull
 paste0("Convex hull")

 set.seed(1)
 data <-data.matrix(data)
```

Figure 3: Step 2 (Geometric Descriptor Construction)

Following the data which is converted into a dataframe, this data is put into the “chull” function which is known as a generic convex-hull-based model selection technique. The job of this function simply finds the positions of the convex hull. The code “data[c(ch,ch[1])]” from the Figure 4 commands R to close the polygon from the data points which is stored in a variable known as “ coo_”. To elaborate on this, this is a way of representing the Gift wrapping or Jarvis Algorithm technique (Jarvis, 1973) , which is used to construct convex hull by starting from the point at the leftmost side on the plot or the point with the minimum value on the x axis and then wrapping the points in clockwise or counter clockwise direction . Furthermore, the “plot” function is used to plot the data and then the “line” function is used to outline the closed polygon which was stored to the variable “coo_”. Finally, the “npts” function is used to count the number of points on the boundary of the outlined polygon as seen in Figure 7 and Figure 7.

```
ch <- chull(data) # find positions of convex hull

coo_ <- data[c(ch,ch[1]),] # Closed Polygon

plot(data, pch=19,xlab ="x-axis", ylab ="y-axis")

lines(coo_, col="blue") #plot data

# finding the number of points on the Convex hull plot
 paste0("Number of points for Convex hull")
 print(npts(coo_))
```

Figure 4: Step 3 (Convex hull Geometric Descriptor Construction)

Constructing the concave hull was faster due to the fact that most work which involved conversion of the dataset to a dataframe and then to a matrix has already been done from the previous codes before the convex hull was constructed. However, Figure 5 shows the "concaveman" function which constructs a concave polygon based on set points given. This function consist of various parameters which include the "points" parameter which simply defines the set points which you need to construct the concave hull , in this case the "data " which is represented in a matrix of co-ordinates was put in that parameter. Also, following the other parameter known as the "concavity" which is a tuning parameter used to measure the degree of concavity. In this experiment this was set to 2 as this represented the concave hull. Also, when the value is below 2 this result to self-intersecting polygons and when the concavity is tuned from 3 and above, the shape of the results to more of a convex polygon. The "plot", "line" and "npts" functions were used similarly just as explained above when constructing the convex hull.

```
concv<- concaveman(points=data,concavity = 2)
plot(data,pch=19,xlab ="x-axis", ylab ="y-axis")
lines(concv,col="blue")
paste0("Number of points for Concave hull ")
print(npts(concv))
```

Figure 5: Step 3 (Concave hull Geometric Descriptor Construction)

The Figure 7 and Figure 7 below shows a constructed convex and concave from a random selected seed set of the same problem.

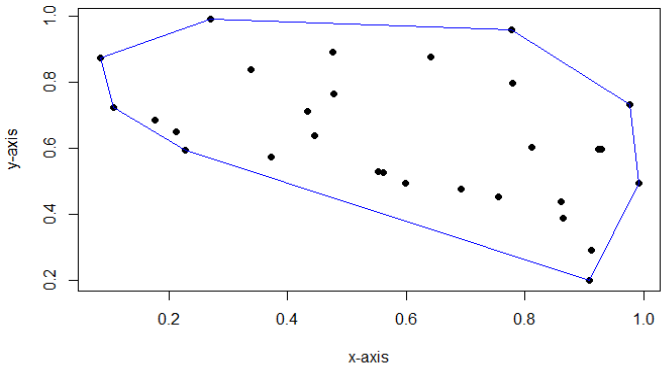

Figure 6: An example of a constructed of Convex Hull problem

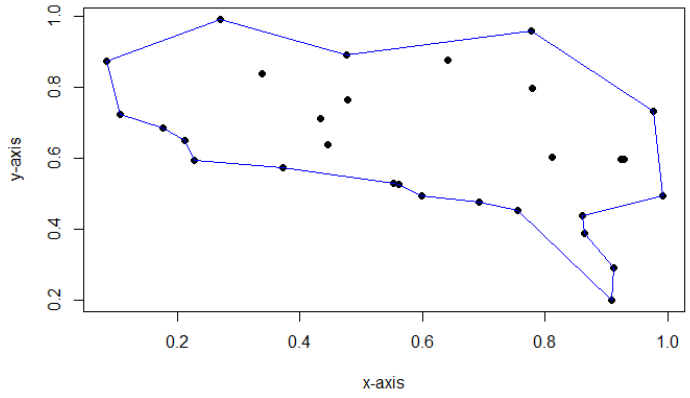

Figure 7: An example of a constructed of Concave Hull problem

Creating a function as seen in Figure 8 to calculate the area of Convex Hull which is built to work like the Shoelace theory by (Page, 2011) .

```
#using a function to find area of Convex Hull(Shoelace theory)
area<-function(coo_){
   coo_<-rbind(coo_,coo_[1,])
   x<-coo_[,1]; y<-coo_[,2]; lx<-length(x)
   sum((x[2:lx]-x[1:lx-1])*(y[2:lx]+y[1:lx-1]))/2
 }

print(area(coo_))
```

Figure 8: Step 4 (Convex hull Area Geometric Descriptor Construction)

The same function is created in to calculate the area of the concave hull.

```
#using a function to find area of Concave Hull
 area<-function(concv){
     concv<-rbind(concv,concv[1,])
     x<-concv[,1]; y<-concv[,2]; lx<-length(x)
     sum((x[2:lx]-x[1:lx-1])*(y[2:lx]+y[1:lx-1]))/2
   }

  paste0("Concave hull area ")
  print(area(concv))
```

Figure 9: Step 4 (Concave hull Area hull Area Geometric Descriptor Construction)

The Figure 10 below shows the use of the “magic_result_as_dataframe” function which compiles all the results in the “print” function in the “for loop” into a data frame which is assigned to a variable “df_1”. For example, in Figure 5 and Figure 5 as seen, the print function is used to print out the number of points and area obtained from each problem for both geometric descriptors. Those features result printed out are further compiled into a dataframe by the “magic_result_as_dataframe” function. Also, the name of the features were renamed and finally I used the “write.csv” function to store that compiled dataframe as a csv.file called “Project_2nddata.csv”.The first 10 rows of the dataset created based on the parameters attained from the geometric descriptors is shown in the figure below.

```
df_1<- magic_result_as_dataframe()      # get the result
colnames( df_1)

#| renaming the columns

 names(df_1)[names(df_1) == "npts(coo_)"] <- "Cnvx.hull.pts"
 names(df_1)[names(df_1) == "area(coo_)"] <- "Cnvx.hull.area"
 names(df_1)[names(df_1) == "npts(concv)"] <- "Cncv.hull.pts"
 names(df_1)[names(df_1) == "area(concv)"] <- "Cnv.hull.area"

 View(df_1)

setwd("C:\\Users\\hp\\Desktop\\ProjectDevelopment")

#Storing names on data frame with the write.csv function
write.csv (df_1, "Project_2nddata.csv")

View(df_1)
```

	files	data	Cnvx.hull.pts	Cnvx.hull.area	Cncv.hull.pts	Cnv.hull.area
1	./seedSets/d10/MOOP1/MOOP1_Ys.csv	*2 variables*	9	0.4481674	21	0.34710400
2	./seedSets/d10/MOOP10/MOOP10_Ys.csv	*2 variables*	9	0.6189835	48	0.50327022
3	./seedSets/d10/MOOP11/MOOP11_Ys.csv	*2 variables*	8	0.7299170	18	0.65369864
4	./seedSets/d10/MOOP12/MOOP12.csv	*2 variables*	8	0.4418448	39	0.41454148
5	./seedSets/d10/MOOP13/MOOP13_Ys.csv	*2 variables*	10	0.6046610	20	0.50011557
6	./seedSets/d10/MOOP14/MOOP14_Ys.csv	*2 variables*	8	0.3913147	55	0.08471285
7	./seedSets/d10/MOOP15/MOOP15_Ys.csv	*2 variables*	11	0.4463328	41	0.32125508
8	./seedSets/d10/MOOP16/MOOP16_Ys.csv	*2 variables*	10	0.7033688	32	0.63182160
9	./seedSets/d10/MOOP17/MOOP17_Ys.csv	*2 variables*	11	0.5383817	38	0.32931173
10	./seedSets/d10/MOOP18/MOOP18_Ys.csv	*2 variables*	11	0.4595743	42	0.16509514

Figure 10: Extracted feature Compiled dataset

3.1.5 Data Exploration and Machine Learning

This section consist of methods taken to perform EDA analysis and Machine learning on the datasets. Find the full R code for this section in the Appendix

3.1.5.1 Data Exploration for "dataset1_given" Dataset

In this section, I performed an EDA analysis and applied machine learning methods for the dataset provided by my supervisor and then merged his dataset with the previous dataset created which was set up from the parameters from the geometric descriptors. The aim was to study how the performance of the machine learning algorithm is affected when predicting the best algorithm for the last generation with and without the parameters of the geometric descriptors. Also, the decision of the algorithms which performed best on the 1200 problems was evaluated at the end of the optimizations runs using the hypervolume (Le & Landa-Silva, 2015)

The following steps taken to achieve this are as follows:

This steps explained below are for the dataset provided by my supervisor.

Firstly, I ensured the working directory is right by setting it up as seen in the Figure 11 below.

Figure 11: Step 1: (Data pre-processing for "dataset1_given")

The Figure 12 below shows the csv.file was loaded up using the "read.csv" and stored in a variable called "dataset1_given". Also, I used the code "dataset1_given_2<-dataset1_given[5]" to remove the last column from the dataset and store it in the same variable. I used the "head" function to print out the first 10 rows of the dataset which can be seen in Figure 13. Then the "colnames" function was used to show the columns names of the dataset. Also, when performing EDA analysis on a dataset in RStudio, it is advisable to know the dimension and structure of the dataset used, which is the reason the "dim" and "str" function were applied to this dataset. Finally, from the analysing the structure of the dataset, I found out that the target class known as "Best.AlgorithmLast.Generation" was stored as a character , therefore I converted this class target to a factor by using the "as.factor" function. This function ensured that the target class is separated in to 3 levels which in return makes the job easier for the machine learning classifiers to learn.

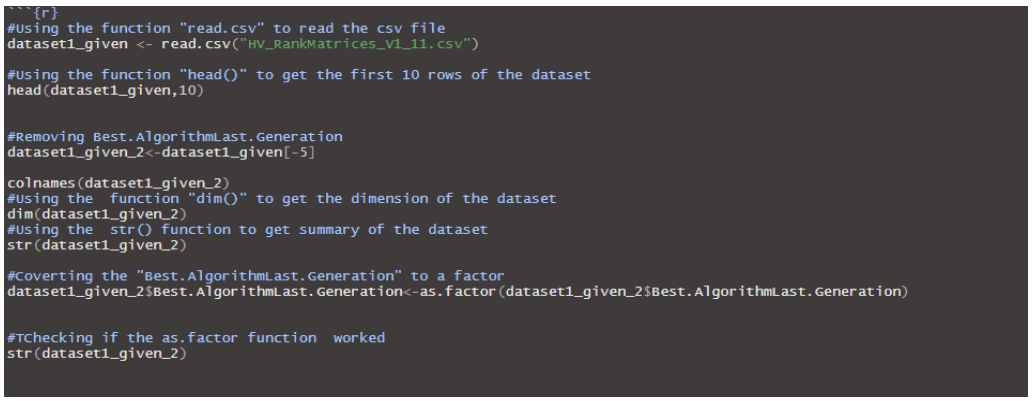

Figure 12: Step 2: (Data pre-processing for "dataset1_given")

	Dimensionality	seed.count	K.value	Best.AlgorithmLast.Generation	Best.Algorithm...Runtime.Average
1	10	91	1	MOEA/D	MOEA/D
2	10	91	2	equal_performance	MOEA/D
3	10	91	3	NSGA-II	NSGA-II
4	10	91	4	NSGA-II	NSGA-II
5	10	91	5	NSGA-II	NSGA-II
6	10	91	6	NSGA-II	NSGA-II
7	10	20	1	MOEA/D	MOEA/D
8	10	20	2	MOEA/D	MOEA/D
9	10	20	3	NSGA-II	NSGA-II
10	10	20	4	NSGA-II	NSGA-II

Figure 13: First 10 rows of the "dataset1_given"

In Figure 14 as seen, I used the “ggplot” function to visualize the class distribution of the dataset. The “geom_bar“ parameter in the “ggplot” function simply commands R to use a bar chart to graphically visualise the results. Furthermore, I used “table” function to count the distribution of the classes and finally the “prop.table function shows the proportion of each class in the dataset.

The Figure 15 shows the number of distribution for the 3 classes which include (MOEA/D, NSGA-II and equal_performance). As seen, from the total problems of 1200, the MOEA/D algorithm has the highest number of count (448) when compared to other classes. The NSGA-II has a count of (392) and finally both algorithm performed the same with a count of (360) out of the total problems of (1200).The correlation plot was not done for this dataset as there was not much attributes for the system to find correlations.

```r
#Class distribution
ggplot(dataset1_given_2, aes(Best.AlgorithmLast.Generation)) +
  geom_bar()
```

```r
table(dataset1_given_2$Best.AlgorithmLast.Generation)
prop.table(table(dataset1_given_2$Best.AlgorithmLast.Generation))
```

Figure 14: Step 3: (Data exploration for “dataset1_given”)

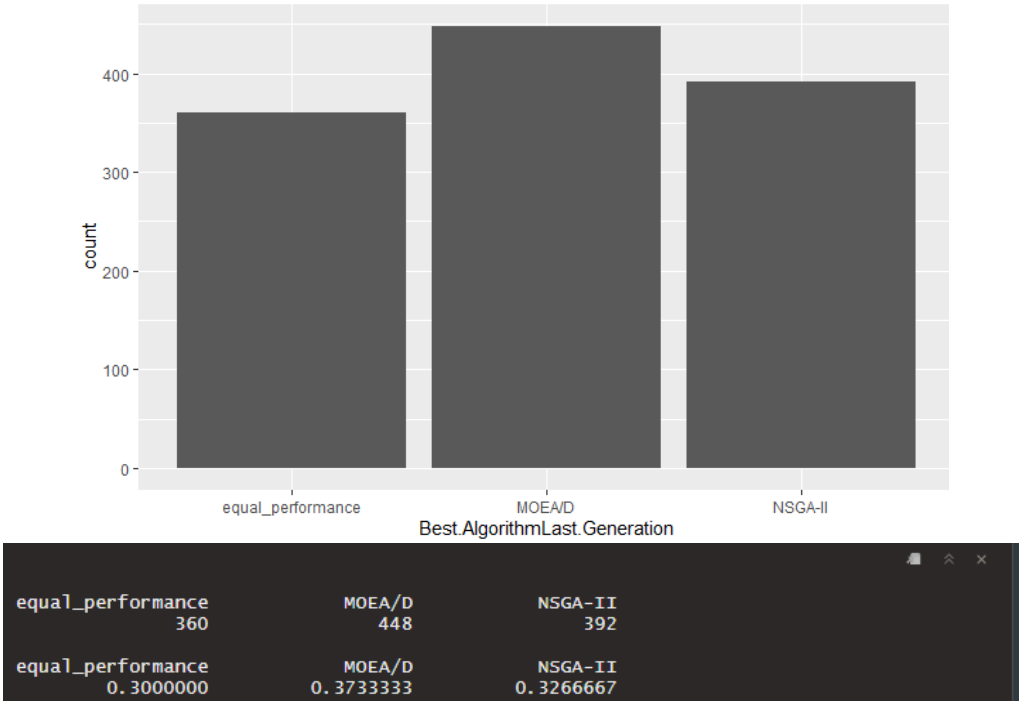

Figure 15: Class Distribution and proportions of "dataset1_given"

The "featurePlot" function was then used in other to show the boxplots of the independent variables as a variables as a function of the 3 classes (i.e. MOEA/D,NSGA-II and equal_performance) as seen in the

seen in the

Figure 16.

```
```{r}
par(mfrow=c(2, 2))
featurePlot(x=dataset1_given_2[,1:3],y=dataset1_given_2$Best.AlgorithmLast.Generation,
plot="box", scales=list(x=list(relation="free"), y=list(relation="free")), col ="blue")

```
```

Figure 16: Step 4:(Data exploration for "dataset1_given")

3.1.5.2 Data Exploration for "Full_data" Dataset

The "Full_data" dataset was formed by merging of both datasets shown in Figure 10 and Figure 13.

The class distribution analysis was not visualized on this dataset, as both datasets consist of the same target class.

Using the same functions such as "read.csv", "head", "colnames", "dim", "str" and "as.factor" as explained in Section 3.1.5.1. In this section, I used the "summary" function was used to gain a summary statistics for each attribute, as this dataset consisted of more numerical columns

added. Also, as seen from Figure 18, the “plot_missing” function which was used to scan through the dataset for any missing value. Finally, the “corrplot” was used in other to visualize the relationships between the attributes in the dataset.

```r
#Setting working directory
setwd("C:\\Users\\hp\\Desktop\\ProjectDevelopment\\Project_plots")

Full_data <- read.csv("Full_HV_RankMatrices_V1 (1).csv")
a<-head(Full_data,10)
View(a)

colnames(Full_data)
head(Full_data)
```

Figure 17:Step 1: (Data pre-processing "Full_data")

```r
dim(Full_data)

#Using the "str" function to check for the format of the dataset
str(Full_data)


#Changing the tarhet class to a factor
Full_data$BestAlgorithm.LastGeneration.<-as.factor(Full_data$BestAlgorithm.LastGeneratio
n.)

# Summary Statistics on each column
summary(Full_data)

#Checking for missing data on the dataset.. As seen below there is no missing value in
the dataset
plot_missing(Full_data)

#Using the correlation matrix to check for the correlation between the attributes in the
datset.
#The K-value has zero correlation to the any other attribute

Full_data_without_class <- cor(Full_data[-8])
corrplot(Full_data_without_class)
```

Figure 18: Step 2: (Data pre-processing "Full_data")

The Figure 19 as seen below shows the first 10 rows of this dataset.

	Dimensionality.d5.	seed.count	K.value	Cnvx.hull.pts	Cnvx.hull.area	Cncv.hull.pts	Cncv.hull.area	BestAlgorithm.LastGeneration.
1	10	91	1	9	0.6189835	48	0.5032702	MOEA/D
2	10	91	2	9	0.6189835	48	0.5032702	equal_performance
3	10	91	3	9	0.6189835	48	0.5032702	NSGA-II
4	10	91	4	9	0.6189835	48	0.5032702	NSGA-II
5	10	91	5	9	0.6189835	48	0.5032702	NSGA-II
6	10	91	6	9	0.6189835	48	0.5032702	NSGA-II
7	10	20	1	8	0.7299170	18	0.6536986	MOEA/D
8	10	20	2	8	0.7299170	18	0.6536986	MOEA/D
9	10	20	3	8	0.7299170	18	0.6536986	NSGA-II
10	10	20	4	8	0.7299170	18	0.6536986	NSGA-II

Figure 19: First 10 rows of the "Full_data" Dataset

The Figure 20 below shows the correlation plot of the attributes in the “Full_data” dataset.

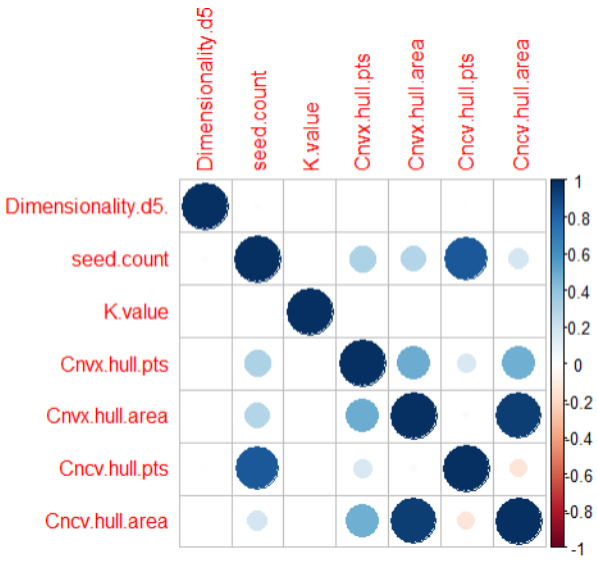

Figure 20: Correlation plot for the "Full_data"

The “featurePlot” function plot as seen in the Figure 21 below was used the same for this dataset as explained in Section 3.1.5.1

```
```{r}
#feature plot
#This shows the box plot for all the 6 variables as a function to the 2 class

par(mfrow=c(2, 2))
featurePlot(x=Full_data[,1:7], y=Full_data$BestAlgorithm.LastGeneration., plot="box",
scales=list(x=list(relation="free"), y=list(relation="free")), col ="blue")


```
```

Figure 21:Step 3: (Data pre-processing "Full_data")

3.1.5.3 Machine Learning methods for both the “dataset1_given” and “Full_data” Datasets.

In this section the below are classification algorithms which would be used to build a model that can successfully predict the most suitable MOEA algorithm for a set of MO-ICOPs.

• KNN (K-Nearest Neighbour)

• SVM (Support Vector Machines)

Note: These following classification models have their individual strengths and weaknesses depending on the dataset.

Evaluation metrics used in this project: Confusion Matrix: Accuracy, Recall

The procedure carried out for the machine learning section which is applied to both dataset is as follows:

For this section “caret” was used to easily compute the KNN and SVM classifiers.

- Firstly, the set seed was set to 1 for all models to ensure consistent reproducibility of results.
- The dataset was split to training and testing set of (~ 80/20 split).
- The method parameter is set to “knn”or “svm_linear” in the train function.
- Using the same classification model the “cv” K-fold cross-validation for both dataset.
- The “preprocess” function simplifies the dataset for the models as it was set to “centre and “scale”.
- Using “tuneGrid” argument was used to tune the “k” value for the KNN model and the “C” value for the SVM models to optimize the model as this has an effect on model performance.
- Validation of the model was achieved by using the Confusion matrix.
- Finding the variable importance of the four models created using the “varImp” function in RStudio.
- Computing the confidence interval based on the accuracy of the models

4 Results and Analysis

4.1 Class Distribution of both datasets

The Figure 22 below Class Distribution shows the number of distribution for the 3 classes which include (MOEA/D, NSGA-II and equal_performance).As seen, from the total problems of 1200, the MOEA/D algorithm has the highest number of count (448) when compared to other classes. The NSGA-II has a count of (392) and finally both algorithm performed the same with a count of (360) out of the total problems of (1200).It can be said that the most suitable algorithm to use for the 1200 problems will be the MOEA/D as this algorithm performed the best for most problems. This class distribution applies to both datasets as they use the same target class.

```
#Class distribution
ggplot(dataset1_given_2, aes(Best.AlgorithmLast.Generation)) +
  geom_bar()
```

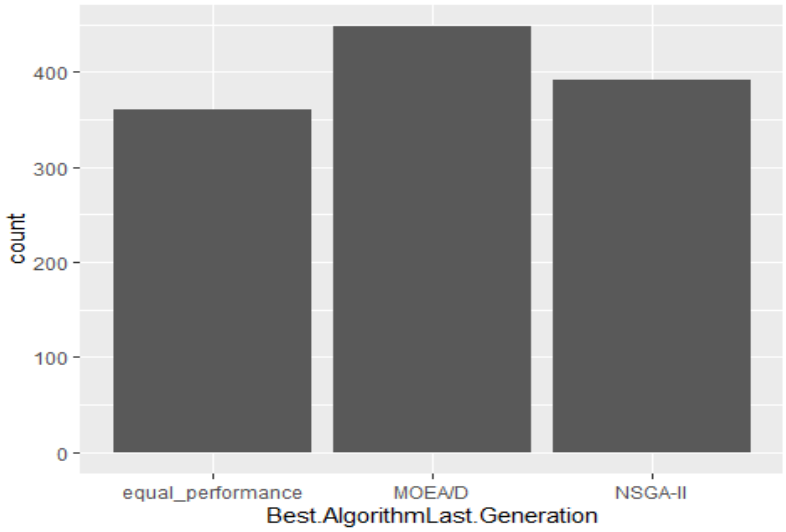

Figure 22: Class Distribution Analysis

4.2 Finding the proportion of the class distribution chart

The functions "prop.table" and "table" is used to show the percentage of the class distribution. As seen, the MOEA/D distribution is 37% of the dataset, the NSGA-II distribution is approximately 33% of the dataset and both had an equal performance of 30% of the dataset.

```
table(dataset1_given_2$Best.AlgorithmLast.Generation)

##
## equal_performance            MOEA/D           NSGA-II
##       360                       448               392

prop.table(table(dataset1_given_2$Best.AlgorithmLast.Generation))

##
## equal_performance            MOEA/D           NSGA-II
##         0.3000000         0.3733333         0.3266667
```

4.3 Feature plot of "dataset1_given" dataset.

Figure 23 below shows the feature plots of the independent variables as a function of the 3 classes (i.e.MOEA/D,NSGA-II and equal_performance).The boxplot of the target classes as seen from the "Dimensionality" attribute shows that when the dimensionality ranges from 5-20 ,The choice of class prediction usually tends to be "equal_Performance". This simply means that when the value of dimensionality ranges from 5-20, the chances of both "NSGA-II" and "MOEA/D" to perform equally for that problem is high. This "equal_performance" also have the highest interquartile range as it spreads out more than any other target class factors as this target class factor is said to be the most inconsistent factor. Also, still

considering the “Dimensionality” variable, when the dimensionality value ranges from 20-30 the chances of the best algorithm suitable for the problem will be the “MOEA/D”. Finally, when the dimensionality value is between 5-10, the chances of the best algorithm to use for that problem will the “NSGA-II” as this is the most consistent result for that range of dimensionality value due to the compact interquartile range. The "equal_perfromance also falls in this range as well but the NSGA-II is more consistent in this range of dimensionality values.

The boxplot of the target classes as seen from the K-value attribute shows that when the K-value ranges from 1-3, the chances of the best algorithm to use for that problem is the “MOEA/D”. On the other hand, when the value of the K-value ranges from 3-5, the chances of both algorithm performing the same (equal_Performance) increases and finally when the K-value falls between 3-6, the chances of the best algorithm to use for those problems will be the “NSGA-II”. As seen, the ranges of both the “equal_performance” and “NSGA-II” attributes intersect and the median of both factors is 4 and they both have a max value of 6. Furthermore, the interquartile range of the ’NSGA-II" spreads more than that of the “equal_perfromance” and “MOEA/D” attributes which makes it more inconsistent. This will result to both algorithm (equal_performnce) suitable for the problems which consist of the k-value from 3-5.The “seed.count” attributes simply shows that when the “seed.count” is high,NSGA-II will be more likely the most suitable algorithm. As seen from the boxplots chart below, the outliers will be ignored as these measurements are important for the analysis of the project. Finally, the Dimensionality value and the K-value have a huge effect on the choice of algorithm to use as seen from the boxplot analysis for the “dataset1_given” dataset.

```
#feature plot
#This shows the box plot for all the 6 variables as a function to the 2 cl
ass

par(mfrow=c(2, 2))
featurePlot(x=dataset1_given_2[,1:3],y=dataset1_given_2$Best.AlgorithmLast
.Generation, plot="box", scales=list(x=list(relation="free"), y=list(relat
ion="free")), col ="blue")
```

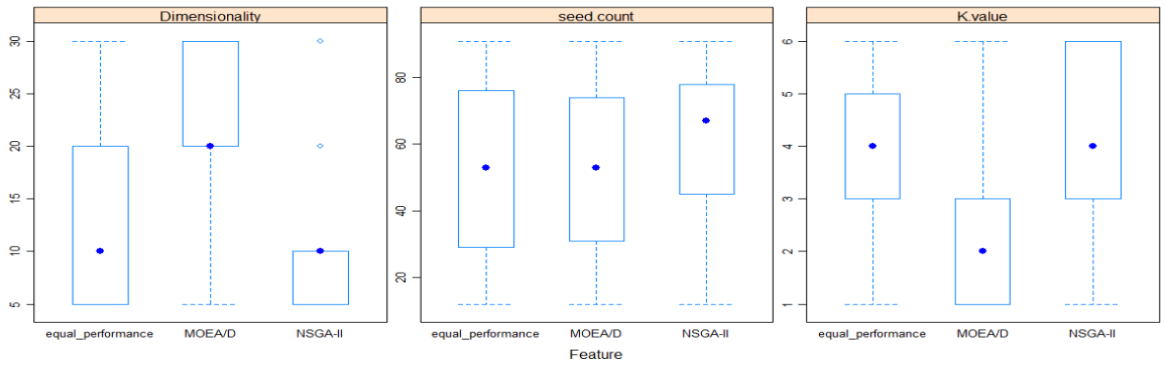

Figure 23: Feature Plot of the "dataset1_given dataset

4.4 Machine Learning Result and Analysis "dataset1_given" dataset

This section highlights the machine learning results and analysis attained from the "dataset1_given" dataset

4.4.1 Splitting dataset to training and testing

```
#Data is split 80% for training and 20% for testing
set.seed(1)

TrainingIndex <- createDataPartition(dataset1_given_2$Best.AlgorithmLast.G
eneration, p=0.8, list = FALSE)
Training <- dataset1_given_2[TrainingIndex,] # Training Set
Testing <- dataset1_given_2[-TrainingIndex,] # Test Set
```

4.4.2 Knn Model "dataset1_given" dataset (without geometric features)

```
set.seed(1)
tr <-trainControl(method="cv", number = 10)# using he cross validation met
hod with 10 folds
knn.model <- train(Best.AlgorithmLast.Generation~.,data=Training,method="k
nn",
            preProcess=c("center", "scale"),
            tuneGrid=expand.grid(.k=1:10),#testing k value from 1 to 10
            trControl=tr)
```

Figure 24 below shows values of "k" tested from 1 to 10 and the most suitable k-value chosen was 9, as this has the highest model optimization of 69.3% accuracy and 53.7% kappa.

```
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.

plot(knn.model)
```

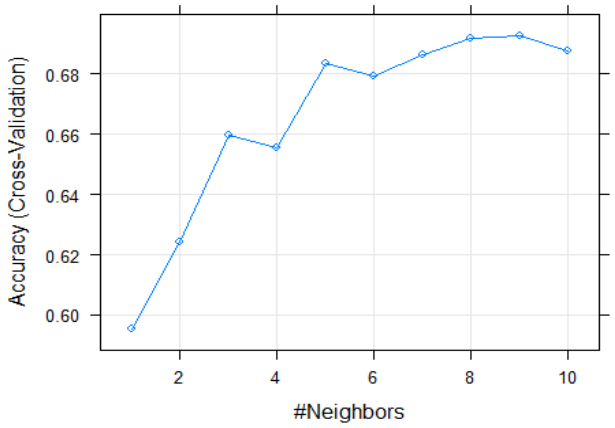

Figure 24: Accuracy(Cross-Validation) vs #Neighbours of the "dataset1_given" dataset (Knn)

4.4.3 Confusion Matrix Knn model "dataset1_given" dataset

- **Training Data of the "dataset1_given" dataset**

For the training data as seen below, the KNN model performed with a Testing - Accuracy: 0.7378 with 95% CI : (0.7087, 0.7653).

From the confusion matrix, This 176 instances are correctly classified as class "equal performance". 48 instances are classified as "equal performance" but are actually instances of the class "MOEA/D". 64 instances are classified as "equal_performance" but are actually instances of the class "NSGA-II".

56 instances are incorrectly classified as class "MOEA/D" but actually belong to the class of "equal_performance". 295 instances are correctly classified as class of "MOEA/D". 8 instances are classified as "MOEA/D" but are actually instances of the class "NSGA-II".

58 instances are incorrectly classified as class "NSGA-II" but actually belong to the class of "equal_performance". 18 instances are correctly classified as class of "MOES/D". 238 instances are correctly classified as class of "NSGA-II".

```
set.seed(1)
#Predicting the training and the test
trainKnn <- predict(knn.model ,Training)
testKnn <- predict(knn.model ,Testing)
#Confusion matrix
confusionMatrix(trainKnn,Training$Best.AlgorithmLast.Generation,mode="ever
ything")
```

```
## Confusion Matrix and Statistics
##
##                                 Reference
## Prediction          equal_performance MOEA/D NSGA-II
##   equal_performance               176     56      58
##   MOEA/D                           48    295      18
##   NSGA-II                          64      8     238
##
## Overall Statistics
##
##                Accuracy : 0.7378
##                  95% CI : (0.7087, 0.7653)
##     No Information Rate : 0.3736
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.605
##
##  Mcnemar's Test P-Value : 0.1905
##
## Statistics by Class:
##
```

```
##                      Class: equal_performance Class: MOEA/D Class: NSGA
-II
## Sensitivity                            0.6111        0.8217         0.7
580
## Specificity                            0.8306        0.8904         0.8
887
## Pos Pred Value                         0.6069        0.8172         0.7
677
## Neg Pred Value                         0.8331        0.8933         0.8
833
## Precision                              0.6069        0.8172         0.7
677
## Recall                                 0.6111        0.8217         0.7
580
## F1                                     0.6090        0.8194         0.7
628
## Prevalence                             0.2997        0.3736         0.3
267
## Detection Rate                         0.1831        0.3070         0.2
477
## Detection Prevalence                   0.3018        0.3757         0.3
226
## Balanced Accuracy                      0.7209        0.8560         0.8
233
```

- **Testing Data of the " dataset1_given" dataset**

For the unseen data as seen below which is also known as the test data, the KNN model performed with a Testing - Accuracy: 0.6695 with 95% CI : (0.6059, 0.7287)

This data consist of 3 predictors or classes as seen (equal_performance, MOEA/D).

From the confusion matrix, This 36 instances are correctly classified as class "equal_performance". 14 instances are classified as "equal_performance" but are actually instances of the class "MOEA/D". 22 instances are classified as "equal_performance" but are actually instances of the class "NSGA-II".

12 instances are incorrectly classified as class "MOEA/D" but actually belong to the class of "equal_performance". 75 instances are correctly classified as class of "MOEA/D". 2 instance is classified as "MOEA/D" but are actually instances of the class "NSGA-II".

23 instances are incorrectly classified as class "NSGA-II" but actually belong to the class of "equal_performance". 6 instances are correctly classified as class of "MOES/D". 49 instances are correctly classified as class of "NSGA-II".

Also, assuming the results from the "equal_performance" class are ignored as there is no right or wrong classification in this case because either the "MOEA/D" or "NSGA-II" can be

chosen as the most suitable algorithm for the problems . However, when it comes down to the classification of the “MOEA/D” or “NSGA-II” from the confusion matrix of the test/unseen data results below, the “MOEA/D” has a total of 89 instances with a correct classification accuracy of 84% of which 75 instances out of 89 samples predicted correctly “MOEA/D” to be the most suitable algorithm to be used. Also 12 instances were classified as “equal performance” which are partially incorrectly classified because only the “MOEA/D” is suitable for those problems and not both algorithms. Finally 2 instances were completely incorrectly classified, as the most suitable algorithm classified for the problem was “NSGA-II” while it was actually “MOEA/D”.

The “NSGA-II” has a total of 78 instances with a correct classification accuracy of approximately 63% of which 49 instances out of 78 predicted correctly “NSGA-II” to be the most suitable algorithm. Also 23 instances were classified as “equal performance” which are partially incorrectly classified because only the “NSGA-II” is suitable in for those problems and not both algorithms. Finally, 6 instances were completely incorrectly classified, as the most suitable algorithm classified for the problem was “MOEA/D” while it was actually “NSGA-II”.

```
set.seed(1)
confusionMatrix(testKnn,Testing$Best.AlgorithmLast.Generation,mode="everyt
hing")
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction          equal_performance MOEA/D NSGA-II
##   equal_performance                36     12      23
##   MOEA/D                           14     75       6
##   NSGA-II                          22      2      49
##
## Overall Statistics
##
##                Accuracy : 0.6695
##                  95% CI : (0.6059, 0.7287)
##     No Information Rate : 0.3724
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5013
##
##  Mcnemar's Test P-Value : 0.5367
##
## Statistics by Class:
##
```

```
##                      Class: equal_performance Class: MOEA/D Class: NSGA
-II
## Sensitivity                            0.5000        0.8427         0.6
282
## Specificity                            0.7904        0.8667         0.8
509
## Pos Pred Value                         0.5070        0.7895         0.6
712
## Neg Pred Value                         0.7857        0.9028         0.8
253
## Precision                              0.5070        0.7895         0.6
712
## Recall                                 0.5000        0.8427         0.6
282
## F1                                     0.5035        0.8152         0.6
490
## Prevalence                             0.3013        0.3724         0.3
264
## Detection Rate                         0.1506        0.3138         0.2
050
## Detection Prevalence                   0.2971        0.3975         0.3
054
## Balanced Accuracy                      0.6452        0.8547         0.7
396
```

- **SVM tunegrid trial for "dataset1_given" dataset**

```
set.seed(1)
#Train control function
tr <-trainControl(method="cv", number = 10)
#Fitting the model in the train function and setting method to "SVM"
#
tr <-trainControl(method="cv", number = 10)
svm.model_trial <- train(Best.AlgorithmLast.Generation~.,data=Training,
                   method="svmLinear",
                   preProcess=c("center", "scale"),
                   tuneGrid=expand.grid(C=c(0.00001, 0.0001, 0.001, 0.01,
0.1, 1, 10, 100, 1000, 10000)),trControl=tr)
```

Figure 25 shows the use of the 'tuneGrid" function for the SVM model, firstly I used a broad range of "C" value of (0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000) in other to find the good values. I found out the best value was at "1" which had the highest accuracy of 67.3% and kappa value of 50.5%.

```
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.

plot(svm.model_trial)
```

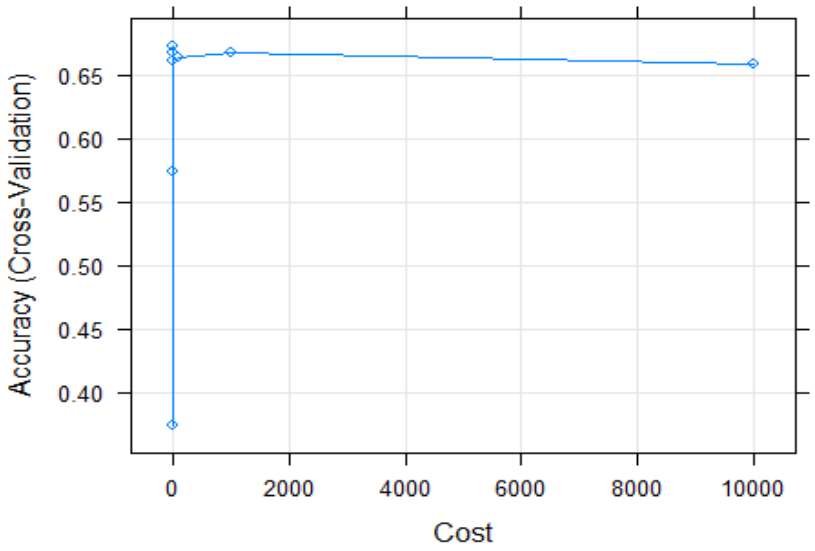

Figure 25: Accuracy(Cross-Validation) vs Cost for the "dataset1_given" dataset (SVM trial)

4.4.4 SVM model "dataset1_given" dataset (without geometric features)

```
set.seed(1)
#Train control function
tr <-trainControl(method="cv", number = 10)
#Fitting the model in the train function and setting method to "SVM"
#
tr <-trainControl(method="cv", number = 10)
svm.model <- train(Best.AlgorithmLast.Generation~.,data=Training,
                   method="svmLinear",
                   preProcess=c("center", "scale"),
                   tuneGrid=expand.grid(C=c(1:10)),trControl=tr)
```

After using a broad range of cost values, I used the cost value ranging from (1-10) and this clearly showed that the most suitable cost chosen still resulted to the value 1, as this has the highest model optimization of 67.3% accuracy and 50.4% kappa as seen from Figure 26.

```
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.

plot(svm.model)
```

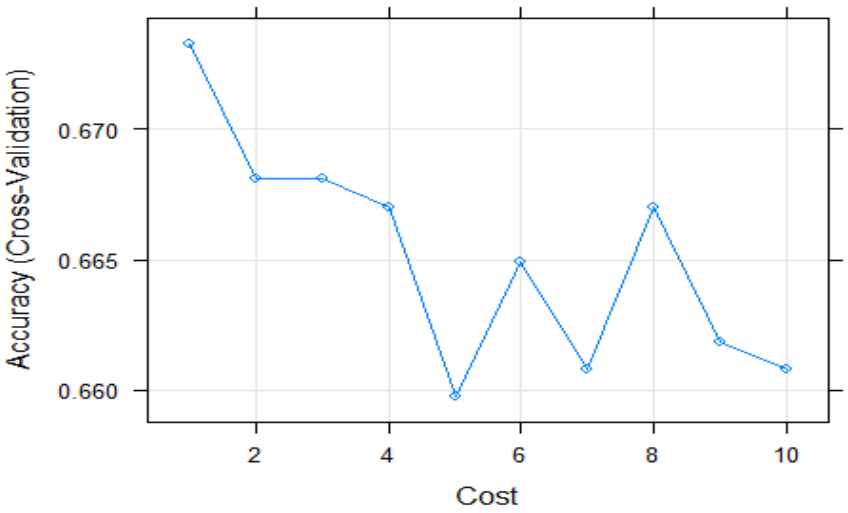

Figure 26: Accuracy (Cross-Validation) vs Cost for the "dataset1_given" dataset (SVM)

4.4.5 Confusion Matrix Analysis SVM model "dataset1_given" dataset

- **Training Data of the "dataset1_given" dataset**

For the training data as seen below, the SVM model performed with a Testing - Accuracy : 0.6785 with 95% CI : (0.6479, 0.7079)

This data consist of 3 predictors or classes as seen ("equal_performance", "MOEA/D" and "NSGA-II").

From the confusion matrix, This 110 instances are correctly classified as class "equal_performance". 73 instances are classified as "equal_performance" but are actually instances of the class "MOEA/D". 105 instances are classified as "equal_performance" but are actually instances of the class "NSGA-II".

47 instances are incorrectly classified as class "MOEA/D" but actually belong to the class of "equal_performance". 306 instances are correctly classified as class of "MOEA/D". 6 instances are classified as "MOEA/D" but are actually instances of the class "NSGA-II".

55 instances are incorrectly classified as class "NSGA-II" but actually belong to the class of "equal_performance". 23 instances are correctly classified as class of "MOES/D". 236 instances are correctly classified as class of "NSGA-II".

```r
set.seed(1)
#Predicting the training and the test
trainSVM <- predict(svm.model ,Training)
testSVM <- predict(svm.model ,Testing)
#Confusion matrix
confusionMatrix(trainSVM,Training$Best.AlgorithmLast.Generation,mode="ever
ything")
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           equal_performance MOEA/D NSGA-II
##   equal_performance               110     47      55
##   MOEA/D                           73    306      23
##   NSGA-II                         105      6     236
##
## Overall Statistics
##
##                Accuracy : 0.6785
##                  95% CI : (0.6479, 0.7079)
##     No Information Rate : 0.3736
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5126
##
##  Mcnemar's Test P-Value : 7.626e-07
##
## Statistics by Class:
##
##                      Class: equal_performance Class: MOEA/D Class: NSGA
-II
## Sensitivity                            0.3819        0.8524         0.7
516
## Specificity                            0.8484        0.8405         0.8
284
## Pos Pred Value                         0.5189        0.7612         0.6
801
## Neg Pred Value                         0.7623        0.9052         0.8
730
## Precision                              0.5189        0.7612         0.6
801
## Recall                                 0.3819        0.8524         0.7
516
## F1                                     0.4400        0.8042         0.7
141
## Prevalence                             0.2997        0.3736         0.3
267
## Detection Rate                         0.1145        0.3184         0.2
456
## Detection Prevalence                   0.2206        0.4183         0.3
611
## Balanced Accuracy                      0.6152        0.8464         0.7
900
```

- **Testing data for the "dataset1_given" dataset**

For the unseen data as seen below which is also known as the test data, the SVM model performed with a Testing - Accuracy : 0.6527 with 95% CI : (0.5887, 0.7129)

This data consist of 3 predictors or classes as seen ("equal_performance", "MOEA/D" and "NSGA-II").

From the confusion matrix, This 31 instances are correctly classified as class "equal_performance". 15 instances are classified as "equal_performance" but are actually instances of the class "MOEA/D". 26 instances are classified as "equal_performance" but are actually instances of the class "NSGA-II".

11 instances are incorrectly classified as class "MOEA/D" but actually belong to the class of "equal_performance". 77 instances are correctly classified as class of "MOES/D". 1 instance is classified as "MOEA/D" but are actually instances of the class "NSGA-II".

24 instances are incorrectly classified as class "NSGA-II" but actually belong to the class of "equal_performance". 6 instances are correctly classified as class of "MOEA/D". 48 instances are correctly classified as class of "NSGA-II".

As seen from the Confidence Intervals of both the training and testing data, there is no significant difference as they both overlap each other.

The "MOEA/D" has a total of 89 instances with a correct classification accuracy of approximately 87% of which 77 instances out of 89 samples predicted correctly "MOEA/D" to be the most suitable algorithm to be used. Also 11 instances were classified as "equal performance" which are partially incorrectly classified because only the "MOEA/D" is suitable for those problems and not both algorithms. Finally 1 instance was completely incorrectly classified, as the most suitable algorithm classified for the problem was "NSGA-II" while it was actually "MOEA/D".

The "NSGA-II" has a total of 78 samples with a correct classification accuracy of approximately 62% of which 48 samples out of 78 predicted correctly "NSGA-II" to be the most suitable algorithm. Also 24 samples were classified as "equal performance" which are partially incorrectly classified because only the "NSGA-II" is suitable in for those problems and not both algorithms. Finally, 6 samples was completely incorrectly classified, as the most suitable algorithm classified for the problem was "MOEA/D" while it was actually "NSGA-II".

```
set.seed(1)
confusionMatrix(testSVM,Testing$Best.AlgorithmLast.Generation,mode="everyt
hing")
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           equal_performance MOEA/D NSGA-II
##   equal_performance                31     11      24
##   MOEA/D                           15     77       6
##   NSGA-II                          26      1      48
##
## Overall Statistics
##
##                Accuracy : 0.6527
##                  95% CI : (0.5887, 0.7129)
##     No Information Rate : 0.3724
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4752
##
##  Mcnemar's Test P-Value : 0.2341
##
## Statistics by Class:
##
##                      Class: equal_performance Class: MOEA/D Class: NSGA
-II
## Sensitivity                            0.4306        0.8652         0.6
154
## Specificity                            0.7904        0.8600         0.8
323
## Pos Pred Value                         0.4697        0.7857         0.6
400
## Neg Pred Value                         0.7630        0.9149         0.8
171
## Precision                              0.4697        0.7857         0.6
400
## Recall                                 0.4306        0.8652         0.6
154
## F1                                     0.4493        0.8235         0.6
275
## Prevalence                             0.3013        0.3724         0.3
264
## Detection Rate                         0.1297        0.3222         0.2
008
## Detection Prevalence                   0.2762        0.4100         0.3
138
## Balanced Accuracy                      0.6105        0.8626         0.7
238
```

4.5 Machine Learning Result and Analysis "Full_data" dataset

The Figure 27 below shows that the dataset is complete and do not contain any missing values

```
#Checking for missing data on the dataset.. As seen below there is no miss
ing value in the dataset
plot_missing(Full_data)
```

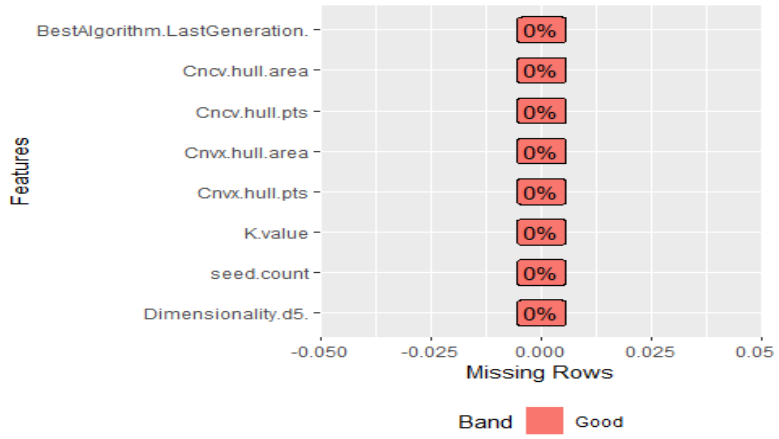

Figure 27: Plot showing number of missing values

Using the correlation matrix as seen in Figure 28 to check for the correlation between the attributes in the dataset. After carrying out some analysis on this correlation plot, I out that the K-value and Dimensionality which are classified as the most important attributes for predicting the target class has zero correlation to the any other attribute in the dataset. Also, the other correlations see in this plot seem to not provide any useable information meaningless simply not important.

```
Full_data_without_class <- cor(Full_data[-8])
corrplot(Full_data_without_class)
```

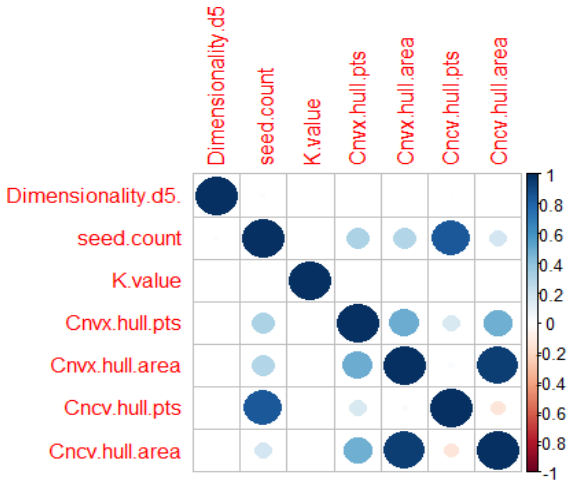

Figure 28: Correlation plot of "Full_data" dataset

4.5.1 Feature plot of "Full_data" dataset

Figure 29 below shows the feature plots of the independent variables as a function of the 3 classes (i.e. "MOEA/D", "NSGA-II" and "equal_performance"). The perfromance of the

“Dimensionality, K-value and seed.count value are not different from the dataset1_given” dataset. However, from the geometric descriptor features added such as “Cnvx.hull.pts” ,“Cnvx.hull.area” ,“Cncv.hull.pts”,“Cncv.hull.area” , the boxplots do not vary as much as the “Dimensionality and ”K-value" which makes it difficult for the machine to use this information to classify the 3 classes in the target class.

```
#feature plot
#This shows the box plot for all the 6 variables as a function to the 2 cl
ass

par(mfrow=c(2, 2))
featurePlot(x=Full_data[,1:7], y=Full_data$BestAlgorithm.LastGeneration.,
plot="box", scales=list(x=list(relation="free"), y=list(relation="free")),
col ="blue")
```

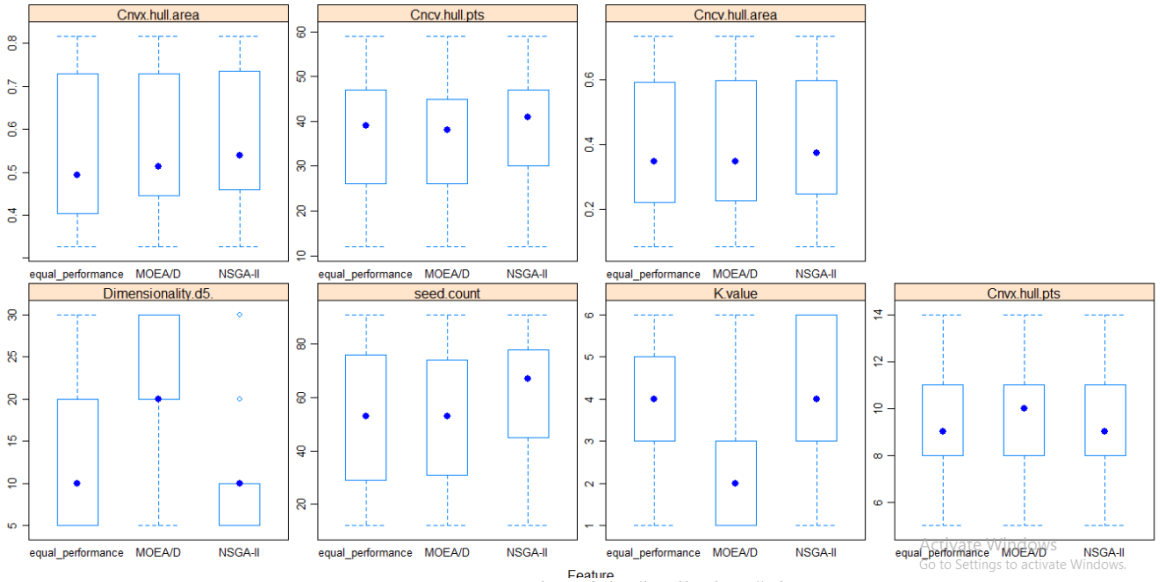

Figure 29: Feature Plot of the "Full_data" dataset

4.5.2 Splitting dataset to training and testing

```
set.seed(1)

TrainingIndex <- createDataPartition(Full_data$BestAlgorithm.LastGeneratio
n., p=0.8, list = FALSE)
Training2 <- Full_data[TrainingIndex,] # Training Set
Testing2 <- Full_data[-TrainingIndex,] # Test Set
```

- **SVM tunegrid trial for “Full_data” dataset**

```
set.seed(1)
#Train control function
tr <-trainControl(method="cv", number = 10)
#Fitting the model in the train function and setting method to "SVM"
```

```
#
tr <-trainControl(method="cv", number = 10)
svm.model2_trial <- train(BestAlgorithm.LastGeneration.~.,data=Training2,
                    method="svmLinear",
                    preProcess=c("center", "scale"),
                    tuneGrid=expand.grid(C=c(0.00001,0.0001,0.001,0.01,0.1,
1,10,100,1000,10000)),trControl=tr)
```

Figure 30 shows result from tuning the 'tuneGrid" function for the SVM model, firstly I used a broad range of "C" value of (0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000) in other to find the good values. I found out the best value was at "1" which had the highest accuracy of 68.2% and kappa value of 51.9%

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.

plot(svm.model2_trial)
```

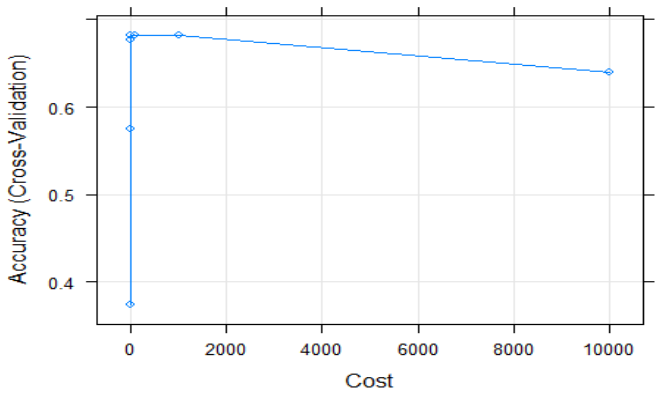

Figure 30: Accuracy(Cross-Validatio) vs Cost for the "Full_data" dataset (SVM trial)

4.5.3 SVM Model "Full_data" dataset (with geometric features)

```
set.seed(1)
#Train control function
tr <-trainControl(method="cv", number = 10)
#Fitting the model in the train function and setting method to "SVM"
#
tr <-trainControl(method="cv", number = 10)
svm.model2 <- train(BestAlgorithm.LastGeneration.~.,data=Training2,
                    method="svmLinear",
                    preProcess=c("center", "scale"),
                    tuneGrid=expand.grid(C=c(1:10)),trControl=tr)
```

After using a broad range of cost values, I used the cost value from range (1-10) and Figure 31 clearly showed that the most suitable cost chosen could be from 4-6 , as these values had the highest model optimization of 68.3% accuracy and 52.1% kappa value.

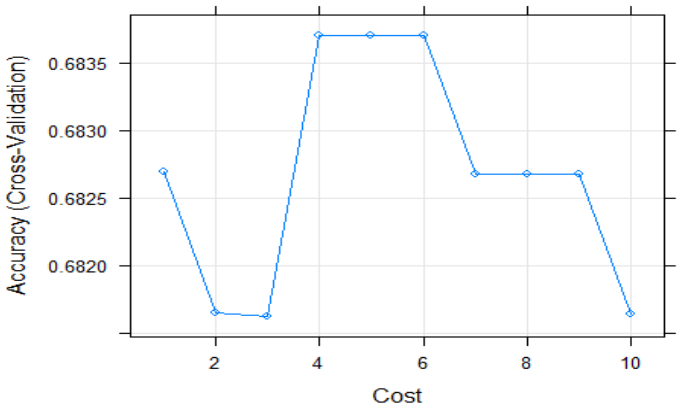

Figure 31: Accuracy(Cross-Validation) vs Cost for the "Full_data" dataset (SVM)

4.5.4 Confusion Matrix Analysis SVM model "Full_data" dataset

- **Training Data for the "Full_data" dataset**

For the training data, the SVM model performed with a Testing - Accuracy: 0.6889 95% CI: (0.6585, 0.718)

This data consist of 3 predictors or classes as seen ("equal_performance", "MOEA/D" and "NSGA-II").

From the confusion matrix, This 121 instances are correctly classified as class "equal_performance". 68 instances are classified as "equal_performance" but are actually instances of the class "MOEA/D". 99 instances are classified as "equal_performance" but are actually instances of the class "NSGA-II".

46 instances are incorrectly classified as class "MOEA/D" but actually belong to the class of "equal_performance". 302 instances are correctly classified as class of "MOEA/D". 11 instances are classified as "MOEA/D" but are actually instances of the class "NSGA-II".

54 instances are incorrectly classified as class "NSGA-II" but actually belong to the class of "equal_performance". 21 instances are correctly classified as class of "MOEA/D". 239 instances are correctly classified as class of "NSGA-II".

```
set.seed(1)
#Predicting the training and the test
trainSVM2<- predict(svm.model2 ,Training2)
testSVM2 <- predict(svm.model2 ,Testing2)
#Confusion matrix
confusionMatrix(trainSVM2,Training2$BestAlgorithm.LastGeneration.,mode="ev
erything")
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction          equal_performance MOEA/D NSGA-II
##   equal_performance               121     46      54
##   MOEA/D                           68    302      21
##   NSGA-II                          99     11     239
##
## Overall Statistics
##
##                Accuracy : 0.6889
##                  95% CI : (0.6585, 0.718)
##     No Information Rate : 0.3736
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5289
##
##  Mcnemar's Test P-Value : 0.0001271
##
## Statistics by Class:
##
##                      Class: equal_performance Class: MOEA/D Class: NSGA
-II
## Sensitivity                            0.4201        0.8412         0.7
611
## Specificity                            0.8514        0.8522         0.8
300
## Pos Pred Value                         0.5475        0.7724         0.6
848
## Neg Pred Value                         0.7743        0.9000         0.8
775
## Precision                              0.5475        0.7724         0.6
848
## Recall                                 0.4201        0.8412         0.7
611
## F1                                     0.4754        0.8053         0.7
210
## Prevalence                             0.2997        0.3736         0.3
267
## Detection Rate                         0.1259        0.3143         0.2
487
## Detection Prevalence                   0.2300        0.4069         0.3
632
## Balanced Accuracy                      0.6358        0.8467         0.7
956
```

- **Testing data for the "Full_data" dataset**

For the unseen data which is also known as the test data, the SVM model performed with a Testing - Accuracy : 0.6444 with 95% CI : (0.5801, 0.705)

This data consist of 3 predictors or classes as seen ("equal_performance", "MOEA/D" and "NSGA-II").

From the confusion matrix, This 33 instances are correctly classified as class “equal_performance”. 12 instances are classified as “equal_performance” but are actually instances of the class “MOEA/D”. 27 instances are classified as “equal_performance” but are actually instances of the class “NSGA-II”.

13 instances are incorrectly classified as class “MOEA/D” but actually belong to the class of “equal_performance”. 75 instances are correctly classified as class of “MOES/D”. 1 instance is classified as “MOEA/D” but are actually instances of the class “NSGA-II”.

25 instances are incorrectly classified as class “NSGA-II” but actually belong to the class of “equal_performance”. 7 instances are correctly classified as class of “MOES/D”. 46 instances are correctly classified as class of “NSGA-II”.

As seen from the Confidence Intervals of both the training and testing data, there is no significant difference as they both overlap each other.

Also, assuming the results from the “equal_performance” class are ignored as there is no right or wrong classification in this case because either the “MOEA/D” or “NSGA-II“ can be chosen as the most suitable algorithm for the problems . However, when it comes down to the classification of the “MOEA/D” or “NSGA-II” from the confusion matrix of the test/unseen data results below, the “MOEA/D” has a total of 89 instances with a correct classification accuracy of 84% of which 75 instances out of 89 instances predicted correctly “MOEA/D” to be the most suitable algorithm to be used. Also 13 instances were classified as “equal performance” which are partially incorrectly classified because only the “MOEA/D” is suitable for those problems and not both algorithms. Finally 1 instance was completely incorrectly classified, as the most suitable algorithm classified for the problem was “NSGA-II” while it was actually “MOEA/D”.

The “NSGA-II” has a total of 78 instances with a correct classification accuracy of approximately 59% of which 46 instances out of 78 predicted correctly “NSGA-II” to be the most suitable algorithm. Also 25 instances were classified as “equal performance” which are partially incorrectly classified because only the “NSGA-II” is suitable in for those problems and not both algorithms. Finally, 7 instances was completely incorrectly classified, as the most suitable algorithm classified for the problem was “MOEA/D” while it was actually “NSGA-II”.

```
confusionMatrix(testSVM2,Testing2$BestAlgorithm.LastGeneration.,mode="ever
ything")
```

```
## Confusion Matrix and Statistics
##
##                     Reference
## Prediction           equal_performance MOEA/D NSGA-II
##   equal_performance                33     13      25
```

```
##   MOEA/D                           12     75       7
##   NSGA-II                          27      1      46
##
## Overall Statistics
##
##                Accuracy : 0.6444
##                  95% CI : (0.5801, 0.705)
##     No Information Rate : 0.3724
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4636
##
##  Mcnemar's Test P-Value : 0.2021
##
## Statistics by Class:
##
##                      Class: equal_performance Class: MOEA/D Class: NSGA
-II
## Sensitivity                            0.4583        0.8427         0.5
897
## Specificity                            0.7725        0.8733         0.8
`
```

As seen from the Confidence Intervals of both the training and testing data, there is no significant difference as they both overlap each other.

4.5.5 Knn Model "Full_data" dataset (with geometric features)

```
set.seed(1)
tr <-trainControl(method="cv", number = 10)# using he cross validation met
hod with 10 folds
knn.model2 <- train(Best.AlgorithmLast.Generation~.,data=Training2,method=
"knn",
            preProcess=c("center", "scale"),
            tuneGrid=expand.grid(.k=1:10),#testing k value from 1 to 10
            trControl=tr)
```

Figure 32 below shows values of "k" tested from 1 to 10 and the most suitable k-value chosen was 10, as this has the highest model optimization of 68% accuracy and 51% kappa.

```
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 10.

plot(knn.model)
```

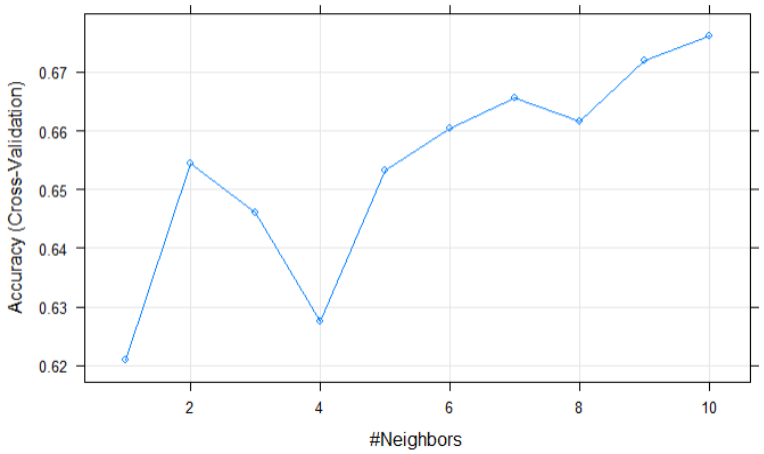

Figure 32: Accuracy (Cross-Validation) vs #Neighbours of the "Full_data" dataset (Knn)

4.5.6 Confusion Matrix Analysis KNN model "Full_data" dataset

- **Training Data for Full_data dataset**

For the training data as seen below, the SVM model performed with a Testing - Accuracy : 0.7575 with 95% CI : (0.7292, 0.7843)

This data consist of 3 predictors or classes as seen ("equal_performance", "MOEA/D" and "NSGA-II")

From the confusion matrix, This 151 instances are correctly classified as class "equal_performance".57 instances are classified as "equal_performance" but are actually instances of the class "MOEA/D".80 instances are classified as "equal_performance" but are actually instances of the class "NSGA-II".

26 instances are incorrectly classified as class "MOEA/D" but actually belong to the class of "equal_performance".319 instances are correctly classified as class of "MOES/D".14 instances are classified as "MOEA/D" but are actually instances of the class "NSGA-II".

40 instances are incorrectly classified as class "NSGA-II" but actually belong to the class of "equal_performance".16 instances are correctly classified as class of "MOES/D".258 instances are correctly classified as class of "NSGA-II".

```
set.seed(1)
#Predicting the training and the test
trainKnn2 <- predict(knn.model2,Training2)
testKnn2 <- predict(knn.model2,Testing2)
```

```
#Confusion matrix
confusionMatrix(trainKnn2,Training2$BestAlgorithm.LastGeneration.,mode="ev
erything")

## Confusion Matrix and Statistics
##
##                     Reference
## Prediction          equal_performance MOEA/D NSGA-II
##   equal_performance               151     26      40
##   MOEA/D                           57    319      16
##   NSGA-II                          80     14     258
##
## Overall Statistics
##
##                Accuracy : 0.7575
##                  95% CI : (0.7292, 0.7843)
##     No Information Rate : 0.3736
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6328
##
##  Mcnemar's Test P-Value : 1.511e-05
##
## Statistics by Class:
##
##                      Class: equal_performance Class: MOEA/D Class: NSGA
-II
## Sensitivity                            0.5243        0.8886         0.8
217
## Specificity                            0.9019        0.8787         0.8
547
## Pos Pred Value                         0.6959        0.8138         0.7
330
## Neg Pred Value                         0.8159        0.9297         0.9
080
## Precision                              0.6959        0.8138         0.7
330
## Recall                                 0.5243        0.8886         0.8
217
## F1                                     0.5980        0.8495         0.7
748
## Prevalence                             0.2997        0.3736         0.3
267
## Detection Rate                         0.1571        0.3319         0.2
685
## Detection Prevalence                   0.2258        0.4079         0.3
663
## Balanced Accuracy                      0.7131        0.8837         0.8
382
```

- **Testing data Full_data dataset**

For the unseen data as seen below which is also known as the test data, the SVM model performed with a Testing - Accuracy : 0.6569 with 95% CI : (0.593, 0.7169)

This data consist of 3 predictors or classes as seen (“equal_performance”, “MOEA/D” and "NSGA-II").

From the confusion matrix, This 30 instances are correctly classified as class "equal_performance".13 instances are classified as "equal_performance" but are actually instances of the class "MOEA/D".

29 instances are classified as "equal_performance" but are actually instances of the class "NSGA-II".7 instances are incorrectly classified as class "MOEA/D" but actually belong to the class of "equal_performance".77 instances are correctly classified as class of "MOEA/D".5 instance is classified as "MOEA/D" but are actually instances of the class "NSGA-II".

20 instances are incorrectly classified as class "NSGA-II" but actually belong to the class of "equal_performance".8 instances are correctly classified as class of "MOEA/D".50 instances are correctly classified as class of "NSGA-II".

As seen from the Confidence Intervals of both the training and testing data, there is no significant difference as they both overlap each other.

The “MOEA/D” has a total of 89 instances with a correct classification accuracy of approximately 87% of which 77 instances out of 89 predicted correctly “MOEA/D” to be the most suitable algorithm to be used. Also 7 instances were classified as “equal performance” which are partially incorrectly classified because only the “MOEA/D” is suitable for those problems and not both algorithms. Finally 5 instances was completely incorrectly classified, as the most suitable algorithm predicted for the problem was “NSGA-II” while it was actually “MOEA/D”.

The “NSGA-II” has a total of 78 instances with a correct classification accuracy of approximately 64% of which 50 instances out of 78 predicted correctly “NSGA-II” to be the most suitable algorithm. Also 20 instances were classified as “equal performance” which are partially incorrectly classified because only the “NSGA-II” is suitable in for those problems and not both algorithms. Finally, 8 instances were completely incorrectly classified, as the most suitable algorithm classified for the problem was “MOEA/D” while it was actually “NSGA-II”.

```
confusionMatrix(testKnn2,Testing2$BestAlgorithm.LastGeneration.,mode="everything")
```

```
## Confusion Matrix and Statistics
##
```

```
##                     Reference
## Prediction           equal_performance MOEA/D NSGA-II
##   equal_performance                30      7      20
##   MOEA/D                           13     77       8
##   NSGA-II                          29      5      50
##
## Overall Statistics
##
##                Accuracy : 0.6569
##                  95% CI : (0.593, 0.7169)
##     No Information Rate : 0.3724
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4808
##
##  Mcnemar's Test P-Value : 0.2462
##
## Statistics by Class:
##
##                      Class: equal_performance Class: MOEA/D Class: NSGA
-II
## Sensitivity                            0.4167        0.8652         0.6
410
## Specificity                            0.8383        0.8600         0.7
888
## Pos Pred Value                         0.5263        0.7857         0.5
952
## Neg Pred Value                         0.7692        0.9149         0.8
194
## Precision                              0.5263        0.7857         0.5
952
## Recall                                 0.4167        0.8652         0.6
410
## F1                                     0.4651        0.8235         0.6
173
## Prevalence                             0.3013        0.3724         0.3
264
## Detection Rate                         0.1255        0.3222         0.2
092
## Detection Prevalence                   0.2385        0.4100         0.3
515
## Balanced Accuracy                      0.6275        0.8626         0.7
149
```

4.6 Evaluation of Models

After applying EDA and using classification algorithms, I evaluate the individual performance of the algorithms. The evaluation is based on the accuracies obtained from the various models. Figure 33 shows that the accuracies of the models overlap each other, which means that there is no significant difference in accuracy performances.

```
#The "with_features" models represents the dataset1_given dataset
#The "without_features" models represent the Full_data daaset
```

```
Performances_of_model <- resamples(list(knn_Model_without_features = knn.m
odel,knn_Model_with_features=knn.model2,
SVM_without_features = svm.model , SVM_with_features = knn.model2))

dotplot(Performances_of_model)
```

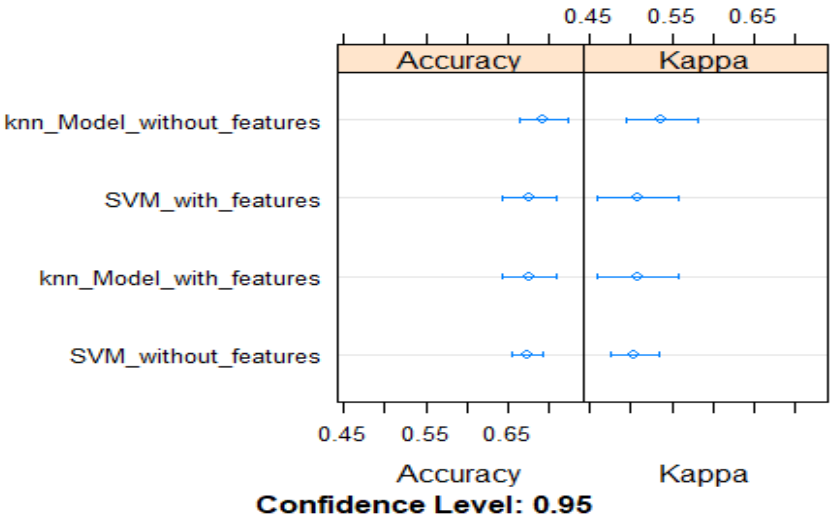

Figure 33: Plot showing the performance of the models based Accuracy

4.7 Variable importance for each model

In the results below, the "Dimensionality" and "K-value" attribute are the most important attributes used for classification followed by the "seed.count" which comes afterwards and the geometric descriptor features has a relatively low influence on the classification algorithms.

```
#Variable impotance for each model
varImp(svm.model)

## ROC curve variable importance
##
##   variables are sorted by maximum importance across the classes
##                equal_performance  MOEA.D NSGA.II
## Dimensionality             45.04 100.000 100.000
## K.value                    81.95  97.447  97.447
## seed.count                  0.00   3.821   3.821

varImp(svm.model2)

## ROC curve variable importance
##
##   variables are sorted by maximum importance across the classes
##                     equal_performance  MOEA.D NSGA.II
## Dimensionality.d5.             53.848 100.000 100.000
## K.value                        84.841  97.856  97.856
## seed.count                     16.029  19.238  19.238
## Cnvx.hull.area                 14.447   4.415  14.447
## Cnvx.hull.pts                  14.432  12.778  14.432
```

```
## Cncv.hull.pts                 13.944  14.349  14.349
## Cncv.hull.area                 7.151   0.000   7.151

varImp(knn.model)

## ROC curve variable importance
##
##   variables are sorted by maximum importance across the classes
##                equal_performance  MOEA.D NSGA.II
## Dimensionality             45.04 100.000 100.000
## K.value                    81.95  97.447  97.447
## seed.count                  0.00   3.821   3.821

varImp(knn.model2)

## ROC curve variable importance
##
##   variables are sorted by maximum importance across the classes
##                     equal_performance  MOEA.D NSGA.II
## Dimensionality.d5.             53.848 100.000 100.000
## K.value                        84.841  97.856  97.856
## seed.count                     16.029  19.238  19.238
## Cnvx.hull.area                 14.447   4.415  14.447
## Cnvx.hull.pts                  14.432  12.778  14.432
## Cncv.hull.pts                  13.944  14.349  14.349
## Cncv.hull.area                  7.151   0.000   7.151
```

5 Discussion and Reflection

This project comprises of using use two datasets with and without geometric features to predict the most suitable MOEA/'s for set of 1200 problems and compare their performances. Firstly, the models built in this report was developed in other to learn from a historical data containing the most suitable MOEA's used for about 1200 problems. The class target consisted of 3 target classes which include: "MOEA/D", "NSGA-II" and "equal_performance". This means the most suitable MOEA for the set of 1200 problems could either be NSGA-II, MOEA/D and both MOEA's could possibly have equal performance for some problems.

The datasets was split into training and test data with an ~ (80/20) split. This means 80% of the data was used for training and the 20% was used for testing. Furthermore, the confusion matrix was used to evaluate the performances of the 4 models in this report. From both the train and test results obtained as seen from the confusion matrix Sections (4.4.3, 4.4.5, 4.5.4 and 4.5.6) the models performed well when predicting the most suitable MOEA's for the problems to either be "MOEA/D" or "NSGA-II". However, the model did not perform well as much for predicting the problems which both MOEA's are both suitable (equal_prformance). This resulted to the decrease in the accuracy percentage of the models. However, if assumptions are made such as ignoring the "equal_performance" class results and focusing on how the model classified the "MOEA/D" and "NSGA-II", this is obvious that the model

performed well for both the training and test results as the degree of wrong classification is by far less than the one which are correctly classified. Also as seen in Figure 33 the performance accuracy of these models overlap each other, this simply means the there is no significant difference in the accuracy results of those models therefore not one model can be confidently said performs better than the others. Also, the variable importance as seen in Section 4.7 shows that the "Dimensionality" and K-value" are the most important attributes used in classifying the correct MOEA's for the given problems. Concisely, further analysis on the visual representation of these model performances as seen in Figure 33 shows the result obtained with these classification algorithms proved that adding geometrical descriptors to find the most suitable MOEA for a given set of Multi-Objective Continuous Optimization Problems (MO-ICOP) does not improve the predictive performance. In future research of this project, a geometric feature known as perimeter of the geometric descriptors (i.e. concave and convex hull) can be added to the dataset and used to predict the most suitable MOEA for the problems. Also, the equal_performance class could be replaced randomly with either "MOEA/D' or "NSGA-II" as this will definitely change the performance accuracy of the models.

The project required an understanding of the principles of Optimization algorithm, Genetic Algorithms, MO-ICOP along with an understanding of MOEA's and finally methods used to evaluate those algorithms. Understanding these principles were not straightforward at the beginning as majority of the knowledge was gathered during the investigation phase of this project. The process of constructing the geometric descriptors in RStudio involved a much steeper learning curve. Despite having previous experience in using RStudio and no experience in MOEA's and its evaluation methods, this required hard work and late nights to gain familiarity in other to construct those geometric descriptor of MO-ICOP iteratively. Also, acquiring knowledge of specific RStudio libraries and syntax continued throughout the duration of this project as some of the knowledge required for this project was beyond scope and required using online resources articles, community of bloggers and forum posts.
Through the work done in this project, the end goals and objectives were achieved. This included the relevant designs and methods that were made and the extensive background research and development that was carried out in other to create the best methods to achieve the various required tasks. Although along the line in the design phase which included tuning of the machine learning models on this project, some issues arose in which effective methods were applied to tackle these problems and as a result this ensured that the components comprising the models were not biased and produced the most suitable results.

6 Bibliography

- Awad, M., & Khanna, R. (2015). Efficient Learning Machines. *Support Vector Machines for Classification*, 39-66. doi:https://doi.org/10.1007/978-1-4302-5990-9_3

- Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms.* Chichester,UK: Wiley.

- Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation, 6*(2), 182–197.

- Jabbar, M., Deekshatulua, B., & Chandra, P. (2013). Classification of Heart Disease Using K-Nearest Neighbor and Genetic Algorithm. *International Conference on Computational Intelligence: Modeling Techniques and Applications (CIMTA) 2013*(10), 85 – 94.

- Jarvis, R. (1973). On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters 2*, 18-21. Retrieved [online], from https://www.sciencedirect.com/science/article/abs/pii/0020019073900203#!

- Le, K. N., & Landa-Silva, D. (2015). Hyper-volume Evolutionary Algorithm. *VNU Journal of Science: Comp. Science & Com. Eng.*, 10–32.

- Monfared, M. D., Mohades, A., & Rezaei, J. (2011). Convex hull ranking algorithm for multi-objective evolutionary algorithms. *Scientia Iranica, 18*(6), 1435-1442. Retrieved [online]', from https://www.sciencedirect.com/science/article/pii/S1026309811001672

- Moreira, A., & Santos, M. Y. (2007). Concave Hull: A K-Nearest Neighbours Approach For The Computation Of The Region Occupied By A Set Of Points. Retrieved [online], from https://pdfs.semanticscholar.org/2397/17005c3ebd5d6a42fc833daf97a0edee1ce4.pdf

- Page, J. (2011). Area of a polygon (Coordinate Geometry). Retrieved from https://www.mathopenref.com/coordpolygonarea.html

- Venables, W. N., Smith, D. M., & Team. (2020). An Introduction to R. *Notes on R: A Programming Environment for Data Analysis and Graphics*. Retrieved [online],, from https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

- Zăvoianu, A.-C., Lacroix, B., & McCall, J. (2020). Comparative Run-Time Performance of Evolutionary Algorithms on Multi-Objective Interpolated Continuous Optimisation Problems. *School of Computing Science and Digital Media, Robert Gordon University, Aberdeen, Scotland, UK*.

- Zhang Qingfu and Hui Li. (2007). MOEA/D: *A multiobjective evolutionary algorithm based on decomposition*. IEEE Transactions on evolutionary computation 11(6), 712-731

7 Appendix A

7.1 Project Plan

Using Geometrical Characteristics To Predict The Performance Of Multi-Objective Evolutionary Algorithms

Task Description	Task Duration(Days)	Task Start-Date	Task End-Date
Project Investigation and study	4	20 May 2020	24 May 2020
Geometric descriptor Construction	14	25 May 2020	08 June 2020
Project Literature / Bacground (report-writing)	16	09 June 2020	25 June 2020
Design Methodology	7	26 June 2020	03 July 2020
Motivation and Conclusion	5	15 July 2020	20 July 2020
Others(References, Acknowledgement Structure	4	21 July 2020	25 July 2020
Cross-Check (Reading Project Report)	2	26 July 2020	28 July 2020
Test and Evaluation Process	5	06 August 2020	11 August 2020
Demo and Poster of Project	3	12 August 2020	15 August 2020
Project Report Writing	10	16 August 2020	26 August 2020

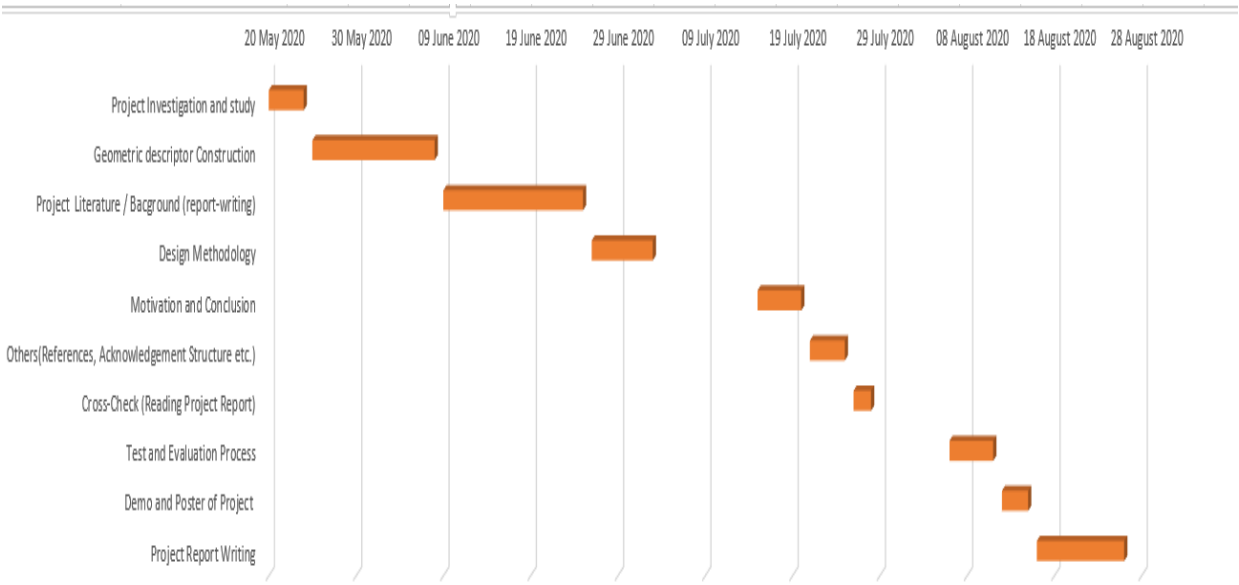

Figure 34:Project Plan Chart

7.2 Constructing Geometric Descriptors Code

```
setwd("C:\\Users\\hp\\Desktop\\ProjectDevelopment\\Original DATASETS
PROJECT\\PPSN_2020-5a25aad3acf974a2e398d3c4b4b515d80c98e139\\benchmarkInstances"
)

all_files <- list.files(pattern = "*.csv", recursive = TRUE,full.names = TRUE)
magic_for(print, silent = TRUE) # call magic_for()

for (files in all_files){


 data<-read.csv(files,header = FALSE) #Reading csv files
 data <- data.frame(data)


 #Constructing Convex hull
 paste0("Convex hull")

 set.seed(1)
 data <-data.matrix(data)

 ch <- chull(data) # find positions of convex hull

 coo_ <- data[c(ch,ch[1]),]

 plot(data, pch=19,xlab ="x-axis", ylab ="y-axis")

 lines(coo_, col="blue") #plot data

 # finding the number of points on the Convex hull plot
  paste0("Number of points for Convex hull")
  print(npts(coo_))

 #using a function to find area of Convex Hull(Shoelace theory)
 area<-function(coo_){
    coo_<-rbind(coo_,coo_[1,])
    x<-coo_[,1]; y<-coo_[,2]; lx<-length(x)
    sum((x[2:lx]-x[1:lx-1])*(y[2:lx]+y[1:lx-1]))/2
  }

 print(area(coo_))
```

```
#Constructing Concave hull
paste0("Concave hull ")

concv<- concaveman(points=data,concavity = 2)
plot(data,pch=19,xlab ="x-axis", ylab ="y-axis")
lines(concv,col="blue")
paste0("Number of points for Concave hull ")
print(npts(concv))

#using a function to find area of Concave Hull
area<-function(concv){
    concv<-rbind(concv,concv[1,])
    x<-concv[,1]; y<-concv[,2]; lx<-length(x)
    sum((x[2:lx]-x[1:lx-1])*(y[2:lx]+y[1:lx-1]))/2
  }

 paste0("Concave hull area ")
 print(area(concv))

 #put(files,Cnvx.hull.pts,Cnvx.hull.area, Cnv.hull.pts,Cnv.hull.pts )


}


df_1<- magic_result_as_dataframe()      # get the result
```

```
colnames( df_1)

names(df_1)[names(df_1) == "npts(coo_)"] <- "cnvx.hull.pts"
names(df_1)[names(df_1) == "area(coo_)"] <- "cnvx.hull.area"
names(df_1)[names(df_1) == "npts(concv)"] <- "cncv.hull.pts"
names(df_1)[names(df_1) == "area(concv)"] <- "cnv.hull.area"

View(df_1)

setwd("C:\\Users\\hp\\Desktop\\ProjectDevelopment")

#Storing names on data frame with the write.csv function
write.csv (df_1, "Project_2nddata.csv")

list.files(pattern ="*.csv")
```

Figure 35:Full code of geometric descriptors construction

7.3 Data Processing and Machine Learning R codes

This section contains codes used in Rstudio for the project.

7.3.1 Data Processing for "dataset1_given"

```
#Using the function "read.csv" to read the csv file
dataset1_given <- read.csv("HV_RankMatrices_V1_11.csv")

#Using the function "head()" to get the first 10 rows of the dataset
head(dataset1_given,10)


#Removing Best.AlgorithmLast.Generation
dataset1_given_2<-dataset1_given[-4]

colnames(dataset1_given_2)
#Using the  function "dim()" to get the dimension of the dataset
dim(dataset1_given_2)
#Using the  str() function to get summary of the dataset
str(dataset1_given_2)

#Coverting the "Best.AlgorithmLast.Generation" to a factor
dataset1_given_2$Best.AlgorithmLast.Generation<-as.factor(dataset1_given_2$Best.AlgorithmLast.Generation)


#TChecking if the as.factor function  worked
str(dataset1_given_2)
```

```
```{r}
#Class distribution
ggplot(dataset1_given_2, aes(Best.AlgorithmLast.Generation)) +
  geom_bar()
```
```

```
```{r}
table(dataset1_given_2$Best.AlgorithmLast.Generation)
prop.table(table(dataset1_given_2$Best.AlgorithmLast.Generation))
```
```

```
```{r}
par(mfrow=c(2, 2))
featurePlot(x=dataset1_given_2[,1:3],y=dataset1_given_2$Best.AlgorithmLast.Generation,
plot="box", scales=list(x=list(relation="free"), y=list(relation="free")), col ="blue")


```
```

Figure 36: Full code in Rstudio for Data Exploration of the "dataset1_given" dataset

7.3.2 Data Processing for “Full_data”

```r
#Setting working directory
setwd("C:\\Users\\hp\\Desktop\\ProjectDevelopment\\Project_plots")

Full_data <- read.csv("Full_HV_RankMatrices_V1 (1).csv")
x<-head(Full_data,10)
View(x)
colnames(Full_data)
```

```r

dim(Full_data)

#Using the "str" function to check for the format of the dataset
str(Full_data)


#Changing the tarhet class to a factor
Full_data$BestAlgorithm.LastGeneration.<-as.factor(Full_data$BestAlgorithm.LastGeneratio
n.)

# Summary Statistics on each column
summary(Full_data)

#Checking for missing data on the dataset.. As seen below there is no missing value in
the dataset
plot_missing(Full_data)

#Using the correlation matrix to check for the correlation between the attributes in the
datset.
#The K-value has zero correlation to the any other attribute

Full_data_without_class <- cor(Full_data[-8])
corrplot(Full_data_without_class)
```

```r
#feature plot
#This shows the box plot for all the 6 variables as a function to the 2 class

par(mfrow=c(2, 2))
featurePlot(x=Full_data[,1:7], y=Full_data$BestAlgorithm.LastGeneration., plot="box",
scales=list(x=list(relation="free"), y=list(relation="free")), col ="blue")
```

Figure 37: Full code in R studio for the "Full_data" dataset

7.3.3 KNN Model ”dataset1_given” dataset

```r
#After split, Training = 840 observations and 5 variables and Testing = 360 observation
and  7 variables.

#Data is split 70% for training and 30% Ffor testing
set.seed(1)

TrainingIndex <- createDataPartition(dataset1_given_2$Best.AlgorithmLast.Generation,
p=0.8, list = FALSE)
Training <- dataset1_given_2[TrainingIndex,] # Training Set
Testing <- dataset1_given_2[-TrainingIndex,] # Test Set
```

```r
set.seed(1)
tr <-trainControl(method="cv", number = 10)
knn.model <- train(Best.AlgorithmLast.Generation~.,data=Training,method="knn",
            preProcess=c("center", "scale"),
            tuneGrid=expand.grid(.k=1:10),#testing k value from 1 to 10
            trControl=tr)
```

```r
print(knn.model)
```

```r
plot(knn.model)
```

```r
set.seed(1)
#Predicting the training and the test
trainKnn <- predict(knn.model ,Training)
testKnn <- predict(knn.model ,Testing)
#Confusion matrix
confusionMatrix(trainKnn,Training$Best.AlgorithmLast.Generation,mode="everything")
```

```r
set.seed(1)
confusionMatrix(testKnn,Testing$Best.AlgorithmLast.Generation,mode="everything")
```

Figure 38: Full code in Rstudio for the Knn model (dataset1_given) dataset

7.3.4 SVM Model "dataset1_given"dataset

```
```{r}
set.seed(1)
#Train control function
tr <-trainControl(method="cv", number = 10)
#Fitting the model in the train function and setting method to "SVM"
#
tr <-trainControl(method="cv", number = 10)
svm.model <- train(Best.AlgorithmLast.Generation~.,data=Training,
                   method="svmLinear",
                   preProcess=c("center", "scale"),
                   tuneGrid=expand.grid(C=c(1:10)),trControl=tr)


```
```{r}
print(svm.model)

```
```{r}
plot(svm.model)

```
```{r}
set.seed(1)
#Predicting the training and the test
trainSVM <- predict(svm.model ,Training)
testSVM <- predict(svm.model ,Testing)
#Confusion matrix
confusionMatrix(trainSVM,Training$Best.AlgorithmLast.Generation,mode="everything")

```
```{r}
set.seed(1)
confusionMatrix(testSVM,Testing$Best.AlgorithmLast.Generation,mode="everything")


```
```

Figure 39: Full code in Rstudio for the SVM model (dataset1_given) dataset

7.3.5 KNN Model. “Full_data” dataset

Figure 40:Full code in Rstudio for the KNN model (Full_data) dataset

7.3.1 SVM Model. “Full_data” dataset

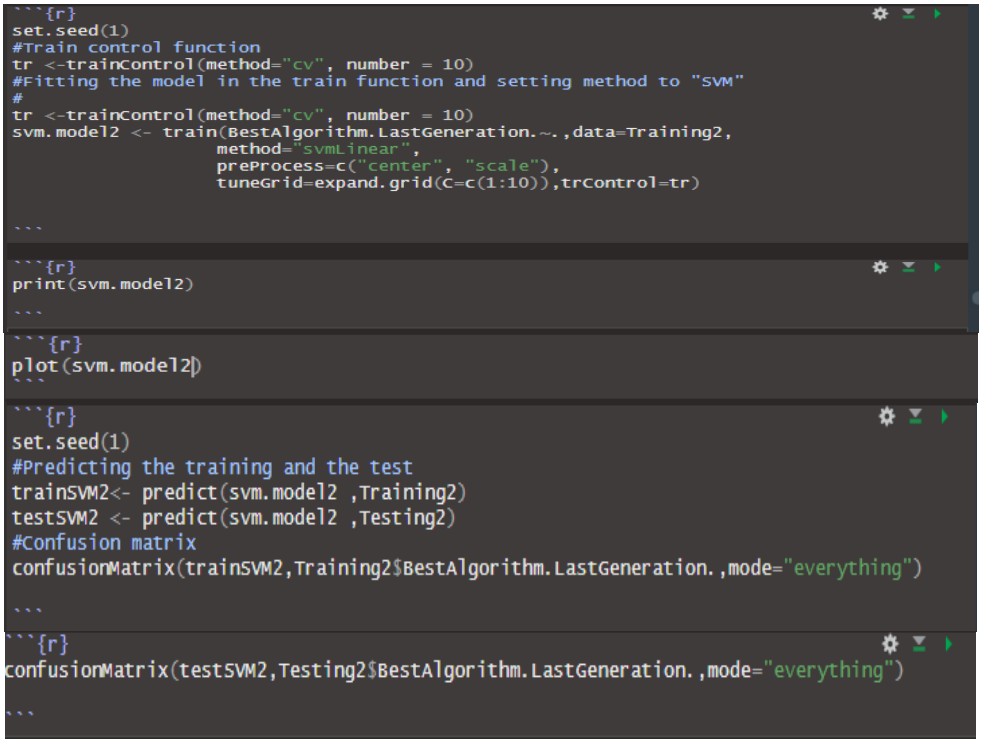

Figure 41:Full code in Rstudio for the SVM model (Full_data) dataset