Ensemble Model of Core Clock Protein Cryptochrome (CRY1) Toxicity Prediction

HA W.M.ALEX

2024-12-10

Contents

1 Executive Summary

The Objective of Capstone Project is to develop a **Core Clock Protein Cryptochrome (CRY1) Toxicity Prediction Ensemble Model** that **Predict Toxicity of Molecules** using **Machine Learning** based on **core_clock_protein_cry** set. The Primary task is to Leverage this data to **Predict Toxicity of CRY1 Molecules/Instances**. Original data is Sourced and Loaded from **UCI Machine Learning Repository** [1]. Renamed as **ccpc_data** and **benchmark_d** dataset. The **ccpc_data** set is a **Non-Linear**, **High Dimensional** but **Small** dataset comprising with **171 of Molecules/Instances**, **1203 Molecular Descriptors/Features** and **1 Feature of Toxicity Class (Toxic or NonToxic)**. The **benchmark_d** dataset comprising with **171 Molecules**, **13 Molecular Descriptors/Features** and **1 Feature of Toxicity Class** for Benchmarking Purpose.

Molecular Descriptors of **core_clock_protein_cry** set for each Model were formulated from **1203 Molecular Descriptors** of **ccpc_data** set using **Molecular Descriptors Prioritization Method**. In **ccpc_data** set, **Zero** Values hold meaningful information relevant to its **Molecular Descriptors**. As such, these **Zero** Values are preserved in their original form and are not subject to any modification or elimination during Data Cleaning process. **Scaling and Normalization** are one of the key Pre-Processing steps in data preparation. **Scaling** ensures that all **Molecular Descriptors** are on the same **Scale**, while **Normalization** adjusts the data to a normal distribution.

Prior to Models Training, **Data Exploration Analysis(EDA)** were conducted which encompassing Data Cleaning, Data Pre-processing, Molecular Descriptors/Features Prioritization, Redundant Feature Elimination (RFE), Features Importance Selection, Redundant Molecular Descriptors Reducing and Principal Component Analysis (PCA). Model employs a **Ensemble Modelling Approach** using Multiple Models of **Gradient Boosted Machine (GBM), Random Forest, K-Nearest Neighbors (KNN) and Extreme Gradient Boosting (XGBoost)** augmented by **Hyper-Parameters Tuning** and **Cross Validation Method** to Optimize Configuration of the **Final Machine Learning Ensemble Model**. These Methods are instrumental in Optimizing magnitude of the Parameters to mitigate the **Risk of Over-fitting**.

The **core_clock_protein_cry** set is also Split into **train sets** and **test sets** for training and testing purpose. Each of the Four **Molecular Descriptors Prioritized Model** (GBM, Random Forst, KNN, XGBoost) will be trained separately with **Optimized Hyper-Parameters** and Evaluate on the **test sets**. **Predictions** will be made and an ensemble created from these Four **Models** to generate a **Majority Prediction of Toxicity Class**. The Ensemble Model **Performance** was evaluated using confusionMatrix function of Caret Package in R. Our ultimate Goal is to created a **Balanced and Robust Ensemble Model** that maximizes the **Performance**, as measured by our Evaluation Metrics such as **F1-Score, Accuracy, Precision and Recall**.

Performance of our Final Ensemble Model is as follow:

F1-Score=0.880, Accuracy=0.823, Recall=0.957, Precision=0.815, Sensitivity=0.957, Specificity = 0.583, P-value=0.021

Final **Ensemble model** combines **Strength** of GBM, Random Forest, KNN and XGBoost Algorithm to learn more complex patterns by **Analyzing Interactions of Molecular Descriptors**, making the **Model** increasingly more **Accurate**. The Model's **Performance** was Evaluated using the Metric such as **F1-Score, Accuracy, Precision and Recall**. The **reliability** and **trustworthiness** of the Model are Substantiated through incorporating **Cross-Validation Method** and **Hyper-Parameters Optimization** during Model development and training.

Optimal Machine Learning Ensemble Model achieved F-Score of 0.880 and Accuracy of 0.823, Signifying a Remarkable Outcome. Thus, I firmly assert our confidence in adopting the ultimate Ensemble Mode and Algorithm to construct the Core Clock Protein Cryptochrome (CRY1) Toxicity Prediction System.

[1] https://archive.ics.uci.edu/dataset/728/toxicity-2

2 Introduction

2.1 Core Clock Protein Cryptochrome (CRY1)

The **Core Clock Protein** refers to a group of Proteins that are central to the regulation of **Circadian Rhythms**, which are body's internal 24-hour cycles that influence various physiological processes. [2] The primary **Core Clock Proteins** include CLOCK(Clock Circadian Regulator) and BMAL1(ARNTL), which form a complex that activates the transcription of other Clock genes like Period (PER) and Cryptochrome (CRY).[3]

These Proteins work together in a feedback loop to maintain the **Circadian Rhythm**, which helps regulate sleep-wake cycles, metabolism, body temperature, and many other functions. Disruptions in these Proteins can lead to various health issues, such as sleep disorders and metabolic problems.

Cryptochrome (CRY) proteins play a crucial role in the regulation of the **Circadian Rhythms** by **interacting** with other **Core Clock Proteins**, There are generally two types, **"CRY1"** and "CRY2", each with its own specific functions, but both are essential for the proper functioning of the **Circadian Clock**.

"CRY" Proteins help regulate the feedback loop that maintains our **Circadian Rhythms** by inhibiting the activity of the "CLOCk-BMAL1" complex, thus controlling the transcription of other **Clock** genes. In simpler terms, they act like a brake in the system to ensure the cycle operates smoothly and on time.

2.2 Toxicity Prediction

2.2.1 Machine Learning Model

Machine Learning Model can **Predict** potential **Toxicity** of drug candidates by analyzing their **interactions** with various biological targets such as **Core Clock Protein cryptochrome (CRY1)** Molecules. This helps in early identification of harmful effects, reducing the risk of late-stage drug failures.

Two Datasets of **Core Clock Protein Cryptochrome (CRY1)** Molecules for Toxicity Prediction were sourced and loaded from **UCI Machine Learning Repository** [4]. Renamed as ccpc_data and benchmark_d dataset , which include **Molecules/Instances** and their associated **Molecular Descriptors** as well as **Toxicity Class**. The ccpc_data dataset comprising with **171 of Molecules/Instances**, **1203 Molecular Descriptors/Features/Compound** and **1 Feature of Toxicity Class**. The benchmark_d dataset comprising with **171 Molecules/Instances**, **13 Molecular Descriptors/Features** and **1 Feature of Toxicity Class** for Benchmarking purpose.

In **Toxicity** of Molecules/Instances **Model** Building, especially when working with our datasets from UCI Machine Learning Repository for **Classification** task, **Molecular Descriptors/Features** are commonly used to help distinguish **Toxicity of Molecules/Instances (Toxic or NonToxic)**. These **Molecular Descriptors** are derived from **Quantitative Structure-Activity Relationship (QSAR)**[5] and are used to create a comprehensive profile of each **Molecule/Instance/Compound**. By analyzing **interactions** of these **Molecular Descriptors/Features**, Machine Learning Models can be trained to classify **Toxicity** of these **Molecules** more accurately.

[2] journals.biologists.com
[3] www.genecards.org
[4] https://archive.ics.uci.edu/dataset/728/toxicity-2
[5] https://en.wikipedia.org/wiki/Quantitative_structure%E2%80%93activity_relationship

QSAR relies on the basic principle of chemistry that states the biological activity of any Compound is associated with the arrangement of atoms forming the Molecular structure. In other words, structurally related Molecules posses similar biological activities. This structural information can be defined in terms of a series of **Parameters** called **Molecular Descriptors**.

2.2.2 Molecular Descriptors/Features

In the context of **Molecular Descriptors/Features**, they comprise Relevant Features from the Chemical Compounds, such as Molecular Weight, Solubility and other Molecular Descriptors of Molecules/Instances. MATS3v and nHBint10 are examples of specific types of Descriptors used in cheminformatics to describe Molecular Properties:

MATS3v

- This stands for Moran Autocorrelation Topological Structure with lag 3, weighted by Van der Waals volumes. It is a type of Descriptor that considers the spatial arrangement of atoms within a Molecule [6]

nHBint10

- This represents the number of hydrogen bond acceptors within 10 bonds. It is a topological Descriptor that counts the number of hydrogen bond acceptors within a certain distance in the Molecular structure. [7]

These Descriptors might be Redundant, meaning they provide similar information and overlap with information provided by the other similar Descriptors. **Dimensionality Reduction Method/Feature Selection** Techniques are typically employed to identify and reduce **Redundant Molecular Descriptors**, improving the **Performance** of the **Machine Learning Models**.

2.2.3 Machine Learning Challenges

Applying Machine Learning to **CRY1 Toxicity Predictions** presents several Challenges:

Data Quality and Quantity

- High Quality, labeled data is essential for training effective Machine Learning Models.

Heterogeneity of Data

- Toxicity data is highly Heterogeneous, with variations between **Molecular Descriptors** and within **CRY1**. This complexity makes it challenging to develop Models that generalize well across different Molecular Descriptors.

High Dimensionality

- CRY1 datasets often include a vast number of **Molecular Descriptors/Features**, which can lead to **Over-Fitting** and computational Challenges.

[6] www.vcclab.org
[7] www.alvascience.com

Hence, Method/Techniques are adopt for **Dimensionality Reduction** as following:

Principal Component Analysis (PCA)

- Reduce the number of Features while retaining the most Important information.(i.e. **Vars Importance**)

Recursive Feature Elimination (RFE)

- Recursively remove the least important Features until a specified number of **Features** is reached.

Redundant Features Reducing

- Eliminate highly correlated Features with cutoff 0.9. (i.e. **coefficient value $>$ 90%**)

Thus, prior to the Model Training, **Molecular Descriptors/Features Prioritization Method** will be **utilized** to Pre-Process and Select Molecular Descriptors/Features that are most relevant to Model for **Toxicity Prediction**.

3 Data Exploration Analysis (EDA)

3.1 Data Explorations

Data was sourced from **UCI Machine Learning Repository.**[8]. Two csv Files were loaded and renamed as **cryptochrome-d.csv** and **benchmark-13.csv**. The **cryptochrome-d.csv** file was read into **ccpc_data** dataframe, while **benchmark-13.csv** file was read into the **benchmark_d** dataframe.

The **ccpc_data** set, consisting of **171 rows and 1204 columns**, was Split into **df_ccpc_data** and **df_ccpc_tox**. As a result, **df_ccpc_data** set comprising of **171 Molecules/Instances** and **1203 Molecular Descriptors/Features.** was generated. And **df_ccpc_tox** set comprise **Single Toxicity Feature:Class of 171 Molecules/Instance.** Then **core_clock_protein_cry** list was created for Principal Component Analysis (PCA) and Model Training, incorporating both the Molecular Descriptors Matrix and Toxicity Vector.

Similarly, "benchmark_d" dataset consisting of **171 rows** and **14 columns.** set was split into **df_benchmark_d** and **df_benchmark_tox**. Hence, **df_benchmark_d** set comprising of **171 Molecules/Instances** and **13 Molecular Descriptors/Features.** was generated. And **df_benchmark_tox** set comprise **Single Toxicity Feature:Class of 171 Molecules/Instance.** . The **benchmark_core_clock_protein_cry** list was also created incorporating both the Molecular Descriptors Matrix and Toxicity Vector.

Redundant Feature Reduction was applied to **df_benchmark_d** set to eliminate **highly correlated Features**, using a **cutoff value** of **0.9**. This process resulted in the creation of **df_benchmark_d_reduced** set, which was subsequently utilized in the **Feature Importance** Techniques. And then **df_benchmark_d_reduced** set was transformed into a **Matrix** and **df_benchmark_d_tox_reduced$Class** was transformed into a **Toxicity Vector.**

Both **core_clock_protein_cry** and **benchmark_core_clock_protein_cry** were then randomly Split into 80% (train_set_x, train_set_y) and 20% (test_set_x, test_set_y). I use a 20% test set instead of 10% because our dataset is Small but High-Dimensional. This requires a sufficient amount of data to create a reliable test set for Evaluation purposes. Both train_set_x and test_set_x contained **Molecular Descriptors** and train_set_y and test_set_y contained **Toxicity Feature** ("Toxic" or "NonToxic") of Molecules/Instances.

Prior to **Model** training, the **Molecular Descriptors Prioritization Method** was applied for **Feature Selection**, reducing **dimensionality** by retaining the **Most Relevant Features**. **Feature Selection** will be performed using the Random Forest Algorithm, and the top Molecular Descriptors will be identified based on **Variable Importance.**

[8] https://archive.ics.uci.edu/dataset/728/toxicity-2

3.1.1 Dimensions - ccpc_data set

A) Dimensions of ccpc_data

Table 1: Dimension of ccpc_data set

Rows	Columns
171	1204

B) Dimensions of benchmark_d

Table 2: Dimension of benchmark_d dataset

Rows	Columns
171	14

3.1.2 Description of Columns/Features and Data Type

The 1 to 1203 Columns/Features - ccpc_data set

These Columns are **Molecular Descriptors** used in Quantitative Structure-Activity Relationship(QSAR) Models to describe Molecular properties. These **Molecular Descriptors/Features** of dataset are used in combination to capture **Core Clock Protein Cryptochrome (CRY1) Toxicity of Molecules/Instances.**

Examples of Molecular Descriptors/Features

- MDEC-23: This Descriptor evaluate Molecular distance edge Descriptors for C.[9]
- apol: This Descriptor calculates sum of atomic polarizabilities.
- MATS3v: This Descriptor measures the Molecular topology and shape. [10]
- nHBint10: This Descriptor counts the number of hydrogen bond acceptors in a Molecule.
- GATS8s: This Descriptor measures the similarity of a Molecule's properties at different distances lag 8 within the Molecule, weighted by I-state.z [11]

The "Class" Column - ccpc_data set

- This column contains either **Toxic or NonToxic** which represents Core Clock Protein Cryptochrome (CRY1) **Toxicity Class of Molecules/Instances.**

The **ccpc_data** set comprises a total of **1204 Columns** with **1203 Molecular Descriptors Features** and **1 Toxicity Class**

[9] docs.ochem.eu
[10] www.vcclab.org
[11] alvascience.com

Data Type of Column - ccpc_data set

Table 3: ccpc_data set

Column	Data_Type	Value
1 to 1203	Numeric	
Class	Character	Toxic or NonToxic

Description of Columns/Features - benchmark_d set

Column Names and Data Type

Table 4: benchmark_d dataset

	Data_Type
MDEC-23	numeric
MATS2v	numeric
ATSC8s	numeric
VE3_Dt	numeric
CrippenMR	numeric
SpMax7_Bhe	numeric
SpMin1_Bhs	numeric
C1SP2	numeric
GATS8e	numeric
GATS8s	numeric
SpMax5_Bhv	numeric
VE3_Dzi	numeric
VPC-4	numeric
Class	character

The benchmark_d dataset comprises a total of **14 Columns** with **13 Molecular Descriptors/Features** and **1 Toxicity class.**

3.1.3 Data Cleaning

R Codes check columns with "NA" values in "ccpc_data" set

```
# check and count number of missing value in ccpc_data set
na_results  <- ccpc_data[apply(is.na(ccpc_data),1,any),]
nrow(na_results)
```

```
## [1] 0
```

There are **NO missing value** in **ccpc_data** set.

R Codes check "Zeros" value in the "ccpc_data" set

```
# Create a logical matrix where TRUE represents zero value
zero_check <- ccpc_data == 0

# Sum the TRUE values for each column to count zeros
zero_count <- colSums(zero_check)

# Display sum of zeros of ccpc_data set
sum(zero_count)
```

```
## [1] 42659
```

In **ccpc_data** set, there are **42659 Zeros** value. It's important to note that these **Zeros** value are not Errors or Missing data. Instead, they carry Meaningful Information relevant to its **Molecular Descriptors.** Therefore, I am preserving all **zeros** value in the dataset and not altering them during the data cleaning process.

This Approach ensures that we maintain the integrity and accuracy of **ccpc_data** set, allowing for a more precise and insightful analysis.

3.1.4 Pre-Processing

3.1.4.1 Data Wrangling

Replace **hyphens** with **underscores** in all **Column Names/Molecular Descriptors** of **ccpc_data** set. The conversion of hyphens "-" to underscores "_" maintain consistency in data formats, also ensuring compatibility with R. This standardize **Column Names** is also making it easier to manage in subsequent analyses and ensuring uniformity across datasets. (e.g., **MDEC-23 to MDEC_23**)

Example: The Column Name of ccpc_data is "MDEC-23" before the Replacement.

```
# Display first 6 rows of column "MDEC-23" descriptors - column range [95:108]
head(ccpc_data[,99])
```

```
## # A tibble: 6 x 1
##   `MDEC-23`
##       <dbl>
## 1      60.2
## 2      44.5
## 3      37.5
## 4      40.6
## 5      52.7
## 6      52.4
```

R Codes that replace hyphens with underscores in the Columns Name

```
# Data Wrangling of ccpc_data for Molecular Descriptors/Features/Columns
#   1) convert hyphens "-" to under_scores "_"
#
colnames(ccpc_data) <- str_replace_all(colnames(ccpc_data), "^'|'$","")
colnames(ccpc_data) <- str_replace_all(colnames(ccpc_data), "-","_")

# Data Wrangling of benchmark_d for Molecular Descriptors/Features/Columns
#   1) convert hyphens "-" to under_scores "_"
#
colnames(benchmark_d) <- str_replace_all(colnames(benchmark_d), "^'|'$","")
colnames(benchmark_d) <- str_replace_all(colnames(benchmark_d), "-","_")
```

Example: The Column Name of ccpc_data is "MDEC_23" after the Replacement.

```
# Display first 6 rows of column "MDEC_23" descriptors - column range [95:108]
head(ccpc_data[,99])
```

```
## # A tibble: 6 x 1
##   MDEC_23
##     <dbl>
## 1    60.2
## 2    44.5
## 3    37.5
## 4    40.6
## 5    52.7
## 6    52.4
```

3.1.4.2 Dataset Splitting

ccpc_data set is splitting into **df_ccpc_data** set and **df_ccpc_tox** set by Column Categories.

benchmark_d set is also splitting into **df_benchmark_d** and **df_benchmark_d_tox** set by Column Categories.

A) Column Categories

ccpc_data set:

- **Column Name** of Columns 1 to 1203: Contain **Name** of **1203 Molecular Descriptors**
- **Value** of Columns 1 to 1203 : Contain **Value** of **1203 Molecular Descriptors**
- **Value** of Column "Class" : Contain **Toxicity** Class (**Toxic or NonToxic**) of **Molecules/Instances**

benchmark_d set:

- **Column Name** of Columns 1 to 13: Contain **Name** of **13 Molecular Descriptors**
- **Value** of Columns 1 to 13 : Contain **Value** of **13 Molecular Descriptors**
- **Value** of Column "Class" : Contain **Toxicity** Class (**Toxic or NonToxic**) of **Molecules/Instances**

B) Generate following New Dataset from ccpc_data set:

df_ccpc_data set:

- **Column Name** of Columns 1 to 1203: Contain **Name** of **1203 Molecular Descriptors**
- **Value** of Columns 1 to 1203 : Contain **Value** of **1203 Molecular Descriptors**

df_ccpc_tox set:

- **Value** of Column "Class" : Contain **Toxicity** Class (**Toxic or NonToxic**) of **Molecules/Instances**

C) Generate following New Dataset from benchmark_d set:

df_benchmark_d set:

- **Column Name** of Columns 1 to 13: Contain **Name** of **13 Molecular Descriptors**
- **Value** of Columns 1 to 13 : Contain **Value** of **13 Molecular Descriptors**

df_benchmark_d_tox set:

- **Value** of Column "Class" : Contain **Toxicity** Class (**Toxic or NonToxic**) of **Molecules/Instances**

D) Dimension of df_ccpc_data set:

Table 5: Dimension of df_ccpc_data set

Rows	Columns
171	1203

E) Dimension of df_ccpc_tox set:

Table 6: Dimension of df_ccpc_tox set

Rows	Columns
171	1

F) Dimension of df_benchmark_d set:

Table 7: Dimension of df_benchmark_d set

Rows	Columns
171	13

G) Dimension of df_benchmark_d_tox set:

Table 8: Dimension of df_benchmark_d_tox set

Rows	Columns
171	1

3.1.4.3 Redundant Features Reducing - benchmark set

Eliminate highly correlated Features of df_benchmark_d with coefficient over 90%.

- **findCorrelation** is a function of Caret package in R, used to detect and remove highly correlated variables in df_benchmark_d set.
- **cutoff:** Parameter/Threshold for correlation. Variables with correlation higher than this value will be flagged.

R Codes that Eliminate Highly Correlated Features with cutoff=0.9

```
##############################################################################
#  Eliminate highly correlated features of df_benchmark_d (coefficient > 90%) #
##############################################################################

# Compute Correlation Coefficient
benchmark_correlation <- cor(df_benchmark_d)

# Find highly correlated features (cutoff = 0.9)
highly_correlated <- findCorrelation(benchmark_correlation, cutoff = 0.9)

# Remove the highly correlated features
df_benchmark_d_reduced  <- df_benchmark_d[, -highly_correlated]

# Assign df_benchmark_d_tox to Reduced toxicity dataset
df_benchmark_d_tox_reduced <- df_benchmark_d_tox
```

Dimension of df_benchmark_d_reduced set:

Table 9: Dimension of df_benchmark_d_reduced set

Rows	Columns
171	12

Column Name of Molecular Descriptors - df_benchmark_d_reduced set:

```
##  [1] "MDEC_23"    "MATS2v"     "ATSC8s"     "VE3_Dt"     "CrippenMR"
##  [6] "SpMax7_Bhe" "SpMin1_Bhs" "C1SP2"      "GATS8s"     "SpMax5_Bhv"
## [11] "VE3_Dzi"    "VPC_4"
```

3.2 Feature Selection

3.2.1 benchmark_core_clock_protein_cry set

R Codes that create benchmark_core_clock_protein_cry set for Feature Selection

```
##########################################################################
#   Create benchmark_core_clock_protein_cry set for Feature Selection   #
##########################################################################

# Transform df_benchmark_d_reduced to moecular_descriptors matrix
molecular_descriptors <- df_benchmark_d_reduced %>% as.matrix()

# Transform df_benchmark_d_tox_reduced$Class to toxicity vector
toxicity <- factor(df_benchmark_d_tox_reduced$Class)

# Generate list - benchmark_core_clock_protein_cry
benchmark_core_clock_protein_cry <- list(molecular_descriptors=molecular_descriptors,
                                         toxicity=toxicity )
```

Number of Molecules/Instances are in the Matrix

```
# Number of Molecules/Instances  are in the Matrix
nrow(benchmark_core_clock_protein_cry$molecular_descriptors)
```

```
## [1] 171
```

Number of Molecular Descriptors/Features are in the Matrix

```
# Number of Molecular Descriptors/Features are in the Matrix
ncol(benchmark_core_clock_protein_cry$molecular_descriptors)
```

```
## [1] 12
```

Proportion of the Molecules/Instances are "Toxic"

```
# Proportion of the Molecules/Instances are "Toxic"
mean(benchmark_core_clock_protein_cry$toxicity=="Toxic")
```

```
## [1] 0.3275
```

Proportion of the Molecules/Instances are "NonToxic"

```
# Proportion of the Molecules/Instances are "NonToxic"
mean(benchmark_core_clock_protein_cry$toxicity=="NonToxic")
```

```
## [1] 0.6725
```

3.2.2 Scaling and Normalizing

Scaling and Normalizing **benchmark__core__clock__protein__cry** set, an essential step in making sure the **Model** can Effectively learn Patterns.

A) Normalization

- Adjust **benchmark__core__clock__protein__cry** set to a common **Scale** without distorting differences in the ranges of values. This is especially important for **Molecular Descriptors/Features** with different units.

B) Standardization

- Transform the **benchmark__core__clock__protein__cry** set to have **Mean of Zero** and **Standard Deviation of One**.

R Codes for Scaling and Normalizing benchmark__core__clock__protein__cry set

```
#####################################################################
#  Scaling and Normalizing  benchmark_core_clock_protein_cry set  #
#####################################################################

# Compute Mean of each Column
col_means <- colMeans(benchmark_core_clock_protein_cry$molecular_descriptors, na.rm=TRUE)

# Compute Standard Deviations of each Column
col_sds <- colSds(benchmark_core_clock_protein_cry$molecular_descriptors, na.rm=TRUE)

# Subtract Column Means
benchmark_molecular_descriptors_centered<-sweep(benchmark_core_clock_protein_cry$molecular_descriptors,
                                                2, col_means, FUN="-")

# Divide by Column Standard Deviations
benchmark_molecular_descriptors_scaled <- sweep(benchmark_molecular_descriptors_centered,
                                                2 , col_sds, FUN="/")
```

List 6 rows of scaled and normalized benchmark__molecular__descriptors__scalad set

```
# List 6 rows of scaled and normalized benchmark_molecular_descriptors_scalad set
head(benchmark_molecular_descriptors_scaled)
```

```
##      MDEC_23  MATS2v   ATSC8s  VE3_Dt CrippenMR SpMax7_Bhe SpMin1_Bhs   C1SP2
## [1,]  2.0861 -0.2167  0.18642 -2.6180   -5.4287     0.9408     3.3408  0.7991
## [2,]  0.5322 -1.4460 -0.34070  0.3127    0.9638     0.7329     0.8670 -0.4900
## [3,] -0.1573  0.8756  0.85231 -2.4638    1.0085     0.2596    -0.2743  3.3774
## [4,]  0.1445  0.9392 -0.10972  0.2111    1.1521     0.2821    -0.2830  2.0883
## [5,]  1.3483 -0.9873 -0.18697  0.5871    0.9242     0.9852     1.5463 -0.4900
## [6,]  1.3133 -1.0167  0.02683 -2.2876    0.7097     0.1160    -0.3140 -0.4900
##       GATS8s SpMax5_Bhv  VE3_Dzi   VPC_4
## [1,] -0.1490     0.6506  0.19722  1.8832
## [2,]  1.1421     0.9124  0.22712 -0.5723
## [3,] -0.7000     1.1008 -0.03622  0.3582
## [4,] -0.2046     1.1626  0.10144  0.5599
## [5,] -0.7804     0.4852  0.37198  0.4452
## [6,] -0.4970     1.2078 -3.20379  0.1701
```

3.2.3 Partitions dataset

Partitions dataset into **80% training and 20% testing sets** as follow:

- Splits 80% of **benchmark_molecular_descriptors_scaled** Matrix into **train_set_x**
- Splits 20% of **benchmark_molecular_descriptors_scaled** Matrix into **test_set_x**
- Splits 80% of **benchmark_core_clock_protein_cry$toxicity** Class into **train_set_y**
- Splits 20% of **benchmark_core_clock_protein_cry$toxicity** Class into **test_set_y**

Given our dataset small size, we allocate 20% for testing rather than the typical 10%. This ensures an adequate amount of data for a reliable and effective evaluation.

R Codes Split dataset into train_set_x,train_set_y (80%) & test_set_x,test_set_y (20%)

```
#############################################################################################
#  Splits dataset into train_set_x, train_set_y (80%) & test_set_x, test_set_y (20%)  #
#############################################################################################
set.seed(1)

test_index <- createDataPartition(benchmark_core_clock_protein_cry$toxicity, times = 1,
                                  p = 0.2, list = FALSE)

test_set_x   <-  benchmark_molecular_descriptors_scaled[test_index,]
test_set_y   <-  benchmark_core_clock_protein_cry$toxicity[test_index]

train_set_x  <- benchmark_molecular_descriptors_scaled[-test_index,]
train_set_y  <- benchmark_core_clock_protein_cry$toxicity[-test_index]
```

3.2.4 Feature Importance

Feature Importance Technique ranks **Molecular Descriptors/Features** of **benchmark_core_clock_protein_cry** set based on their contribution to the Model's predictions using Random Forest Algorithm. **Feature Importance** can be derived from the average decrease in impurity or the mean decrease in Accuracy when a **Molecular Descriptors/Features** is excluded.

A) Model Optimization

- Fine-tune the Models to improve the **Performance**, involving **Hyper-Parameter Tuning** and **Cross-Validation.**
- Optimize the Models by adjusting Parameters as following:
  - Sequence **mtry** from 1 to 10
  - Set Number of Tree (**ntree**) equal to 500
  - Set Number of **Cross validation** equal to 5

R Codes select most relevant Descriptors from Vars Importance using Random Forest

```
#########################################################################################################
# Select Most Relevant Molecular Descriptors from Vars Importance as Benchmark using Random Forest Algo
#########################################################################################################

#####################################################
# Optimized Parameters of Random Forest Algorithm  #
#    * Sequence "mtry" from 1 to 10                 #
#    * Set Number of "tree" equal 500  (ntree=500) #
#    * Set Number of "cross validation" equal to 5 #
#####################################################
set.seed(9)

train_ccpc_benchmark <- caret::train(train_set_x, train_set_y , method="rf",
                                     tuneGrid=data.frame(mtry=seq(1:10)),
                                     ntree=500,
                                     trControl=trainControl(method="cv", number=5),
                                     importance= TRUE)
```

B) List Vars Importance

```
# Process and Generate Vars Importance of train_ccpc_benchmark
importance_rf_b <- varImp(train_ccpc_benchmark)
print(importance_rf_b)
```

```
## rf variable importance
##
##            Importance
## MDEC_23        100.00
## GATS8s          87.87
## SpMax7_Bhe      77.28
## C1SP2           57.73
## CrippenMR       43.08
## VPC_4           36.31
## SpMin1_Bhs      27.55
## SpMax5_Bhv      25.82
## VE3_Dzi         21.22
## ATSC8s          19.58
## MATS2v           5.93
## VE3_Dt           0.00
```

```
# Vars Importance of benchmark set
benchmark_importance_rf <- rownames(importance_rf_b$importance)[order(-importance_rf_b$importance[,1])]
#benchmark_importance_rf[1:12]
```

C) Select Top Pairwise Most Relevant Molecular Descriptors/Features.

MDEC_23 and GATS8s are selected as benchmark set.

R codes predict and compute overall accuracy using confusionMatrix function of caret package

```
# Compute Overall Accuracy for Molecular Descriptors/Feature Prioritization purpose
predict_ccpc_benchmark <- predict(train_ccpc_benchmark, test_set_x)

accuracy_ccpc_benchmark <- confusionMatrix(table(predict_ccpc_benchmark, test_set_y),mode="everything")

benchmark_accuracy <- accuracy_ccpc_benchmark$overall["Accuracy"]
```

The Overall Accuracy of 0.6571 will be adopted as benchmark Accuracy.

3.3 Principal Component Analysis (PCA)

In the context of **Principal Component Analysis (PCA)** for Predicting **Core Clock Protein Cryptochrome (CRY1) Toxicity.** The **core_clock_protein_cry** dataset includes various **Molecular Descriptors** of **Molecules/Instances.** These descriptors are essential for distinguishing **Toxicity Class (Toxic or NonToxic)** of the **Molecules.**

The primary goal of performing **PCA** is to reduce the **dimensionality** of the dataset from **1203 Molecular Descriptors** while retaining the most crucial information. This process transforms the Descriptors into a set of linearly uncorrelated components, known as **Principal Components (PCs)**, that capture the maximum **Variance** in the dataset. The **core_clock_protein_cry** dataset undergoes **Scaling, Normalization and Standardization** before **PCA** is performed using the **prcomp** function from the Caret package.

The results are summarized and **visualized** to understand the **principal components** (e.g., **PC1, PC2, PC3,..**) and their contributions to explaining the **Variance** in the data. Including **Molecular Descriptors** in **PCA** helps identify patterns and relationships that are crucial for distinguishing **Toxicity Class (Toxic or NonToxic)** in the dataset.

3.3.1 core_clock_protein_cry set

R Codes that create core_clock_protein_cry set for PCA

```
#############################################################################
#   Create core_clock_protein_cry set (1203 Molecular Descriptors) for PCA   #
#############################################################################

# Transform df_ccpc_data to molecular_descriptors Matrix
molecular_descriptors <- df_ccpc_data %>% as.matrix()

# Transform df_ccpc_tox$Class to toxicity vector as factor
toxicity <- factor(df_ccpc_tox$Class)

# Generate list - core_clock_protein_cry
core_clock_protein_cry <- list(molecular_descriptors=molecular_descriptors,
                               toxicity=toxicity )
# core_clock_protein_cry
```

Number of Molecules/Instances in the Matrix

```
# Number of Molecules/Instances/Column  are in the Matrix
nrow(core_clock_protein_cry$molecular_descriptors)
```

```
## [1] 171
```

Number of Molecular Descriptors/Features in the Matrix

```
# Number of Molecular Descriptors/Features/Predictors are in the Matrix
ncol(core_clock_protein_cry$molecular_descriptors)
```

```
## [1] 1203
```

Proportion of Molecules/Instances are "Toxic"

```
# Proportion of the Molecules/Instances are "Toxic"
mean(core_clock_protein_cry$toxicity=="Toxic")
```

```
## [1] 0.3275
```

Proportion of Molecules/Instances are "NonToxic"

```
# Proportion of the Molecules/Instances are "NonToxic"
mean(core_clock_protein_cry$toxicity=="NonToxic")
```

```
## [1] 0.6725
```

3.3.2 Scaling and Normalization

Scaling and Normalizing **core_clock_protein_cry** set , an essential step of **PCA** in making sure the **Model** can Effectively learn Patterns.

A) Normalization

- Adjust **core_clock_protein_cry** set to a common **Scale** without distorting differences in the ranges of values. This is especially important for **Molecular Descriptors/Features** with different units.

B) Standardization

- Transform the **core_clock_protein_cry** set to have a **Mean of Zero** and a **Standard Deviation of One.**

R Codes for Scaling and Normalizing core_clock_protein_cry set

```
#############################################################
#  Scaling and Normalizing  "core_clock_protein_cry" set  #
#############################################################

# Compute Mean of each Column
col_means <- colMeans(core_clock_protein_cry$molecular_descriptors, na.rm=TRUE)

# Compute Standard Deviations of each Column
col_sds <- colSds(core_clock_protein_cry$molecular_descriptors, na.rm=TRUE)

# Subtract Column Means
molecular_descriptors_centered <- sweep(core_clock_protein_cry$molecular_descriptors,
                                        2, col_means, FUN="-")

# Divide by Column Standard Deviations
molecular_descriptors_scaled <- sweep(molecular_descriptors_centered,
                                      2 , col_sds, FUN="/")
```

List 6 Rows of scaled and normalized molecular_descriptors_scalad set

```
# List 6 rows of scaled and normalized molecular_descriptors_scalad set
head(molecular_descriptors_scaled[,170:176])
```

```
##      R_TpiPCTPC   MATS1s   MATS1v    JGI10   MATS1c  MATS1e VR2_DzZ
## [1,]     1.8404  0.12944 -0.06358  0.02835 -0.09973 -0.5318 -0.2265
## [2,]     1.2775 -0.36007 -0.86010 -0.79793 -1.04840  0.3064 -0.2267
## [3,]     1.4219 -0.58687 -0.40303 -0.46742 -0.67468 -1.6525 -0.2256
## [4,]     1.3858 -1.01763  0.43719  0.13852 -0.25874 -1.8810 -0.2263
## [5,]     0.6496  0.84248 -0.02661  0.52411  0.79055  2.6592 -0.2268
## [6,]    -0.5452 -0.02394  1.37823  0.30377  0.22278 -0.6696  1.6171
```

3.3.3 Proportion of Variance (PCA)

To compute the **Proportion of Variance** explained by **PCA**, start by creating the Covariance **Matrix** to understand how different Features vary together. Next, use Eigenvalues and Eigenvectors to identify the **Principal Components.** Finally, select the **Principal Components** that capture the **most variance** to retain the most significant patterns in the data.

R Codes that compute Proportion of Variance explained by PCA

```
#####################################
#    PCA - Proportion of variance    #
#####################################

# PCA is performed using the "prcomp" function
pca_result_core_clock_protein_cry <- prcomp(molecular_descriptors_scaled)

# Create Dataframe with column pca_result_core_clock_protein$x
pca_data_core_clock_protein_cry <- data.frame(pca_result_core_clock_protein_cry$x)

# Add toxicity Class to the Dataframe
pca_data_core_clock_protein_cry$toxicity <- core_clock_protein_cry$toxicity

# Compute Proportion of Variance (Eigenvalues and Eigenvectors)
proportions_var_ccpc <-pca_result_core_clock_protein_cry$sdev^2/sum(pca_result_core_clock_protein_cry$s
#proportions_var_ccpc[1:42]
```

R Codes that generate Proportion of Variance explained by First Principal Component (PC1)

```
# Proportion of Variance explained by First Principal Component (PC1)
proportions_var_ccpc_first_pc <- proportions_var_ccpc[1]
proportions_var_ccpc_first_pc
```

```
## [1] 0.1957
```

R Codes compute No of Principal Components required to explain at least 90% of Variance

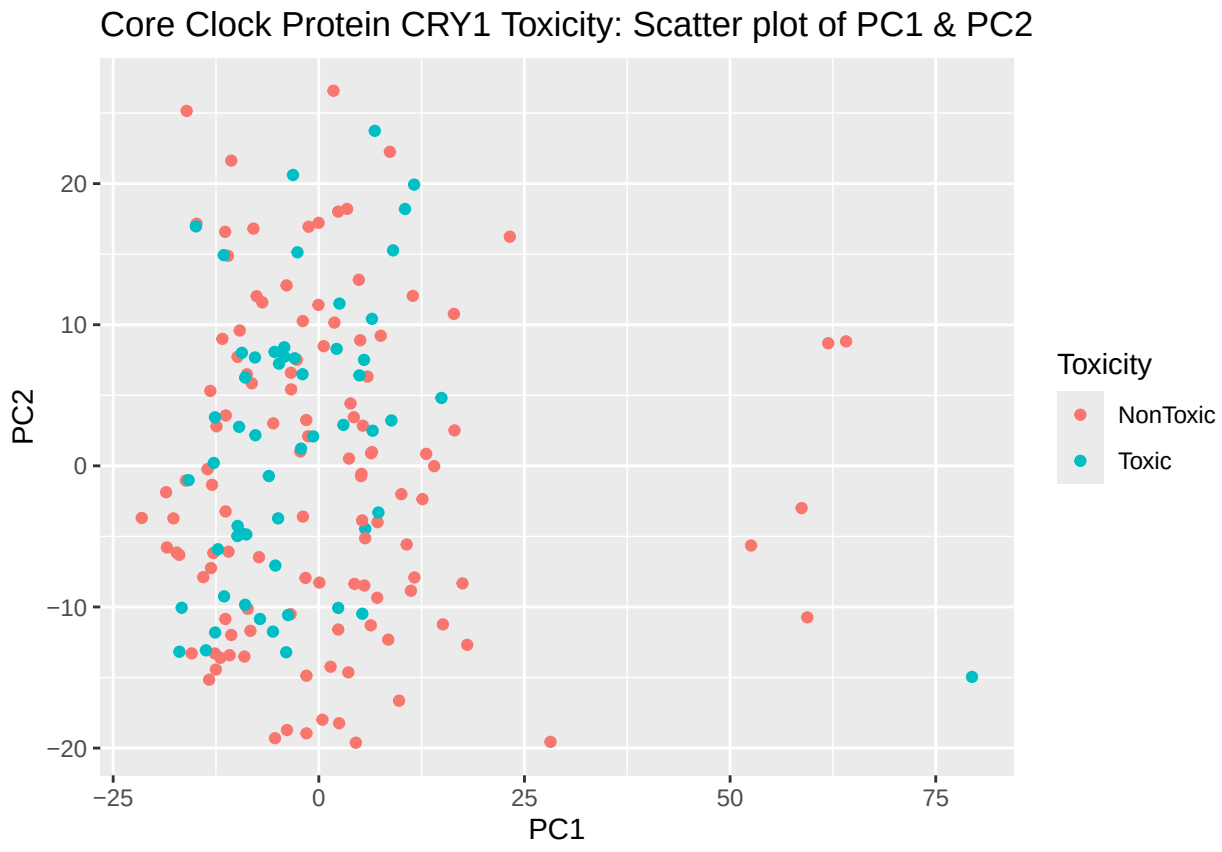
```
# Number of Principal Components are required to explain at least 90% of Variance
cumsum_proportions_var_ccpc   <- cumsum(proportions_var_ccpc)
no_pc_90percent_var_ccpc      <- which(cumsum_proportions_var_ccpc >= 0.9)[1]
```

42 Principal Components are required to explain at least 90% of Variance

PCA Scatter Plot visualizes the **First Two Principal Components** with color representing **Toxicity**, which can help in distinguishing between **Toxic and nonToxic** in the Reduced Feature Space.

R Codes that ggplot Principal Components with color representing Toxicity Class

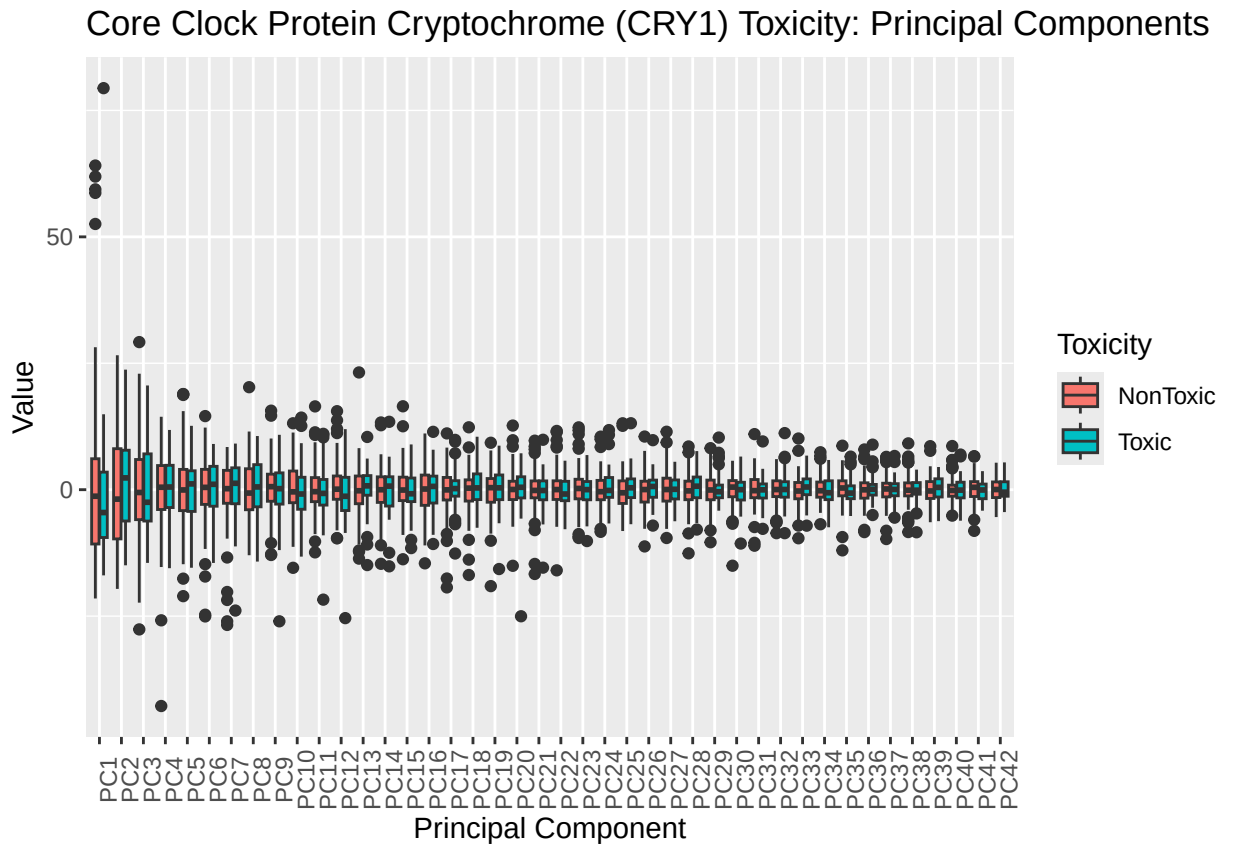
```
########################################
#   PCA - plotting Principal components   #
########################################

# Create data frame for plotting
plot_df_core_clock_protein_cry <-data.frame(PC1=pca_result_core_clock_protein_cry$x[,1],
                                            PC2=pca_result_core_clock_protein_cry$x[,2],
                                            Toxicity=core_clock_protein_cry$toxicity)


# ggplot First Two Principal Components (PC1 & PC2) with color representing Toxicity
ggplot(plot_df_core_clock_protein_cry,aes(x=PC1, y=PC2, color=Toxicity)) +
  geom_point() +
  labs(x="PC1", y="PC2", color="Toxicity") +
  ggtitle("Core Clock Protein CRY1 Toxicity: Scatter plot of PC1 & PC2")
```



PCA Box-Plot shows the distribution of the **Principal Components** grouped by **Toxicity**, helping to identify outliers and understand the spread of the data.

R Codes plot PCA Box-Plot show distribution of Principal Components grouped by Toxicity

```
################################
#  PCA - Plotting PC Box-plot  #
################################

plot_df_10 <- data.frame(pca_result_core_clock_protein_cry$x[,1:no_pc_90percent_var_ccpc])
plot_df_10$Toxicity <- core_clock_protein_cry$toxicity

melted_df_10 <- melt(plot_df_10, id.vars="Toxicity")

ggplot(melted_df_10, aes(x=variable, y=value, fill=Toxicity)) +
  geom_boxplot() +
  labs(x="Principal Component", y="Value", fill="Toxicity") +
  theme(axis.text.x=element_text(angle=90)) +
  ggtitle("Core Clock Protein Cryptochrome (CRY1) Toxicity: Principal Components")
```



Handling **1,203 Molecular Descriptors** of **Core Clock Protein Cryptochrome (CRY1) Molecules** can be challenging and may lead to **Model** Over-fitting. Over-fitting occurs when **Model** capture noise in the training data, which results in poor generalization to new data and difficulties in identifying patterns. **PCA** results do not display clear **Separations of Class** after plotting, it suggests that **PCA** alone is insufficient to differentiate **Toxicity Classes.**

In such cases, additional strategies should be employed to Enhance Classification **Performance.** Recommended approach is to integrate **PCA** with **Molecular Descriptors Prioritization Methods** and **Supervised Learning Algorithms. PCA** can serve as a Pre-Processing step to reduce **dimensionality.** Following this, a **Features Prioritization Method** can be applied to select a subset of the **Most Relevant Molecular Descriptors**, thereby reducing **Molecular Descriptors** and complexity. Finally, Supervised Learning Algorithms (e.g., **KNN,GBM,Random Forests,XGBoost**) can be **utilized** to classify the **Toxicity of Molecules/Instances.**

while **PCA** is a powerful tool, it may not always be the optimal choice for every dataset due to its **limitations.** Even though **PCA** alone does not yield clear **Classifications of Toxicity**, combining **Features Prioritization Methods** with an **Ensemble Model** of Supervised Learning Algorithms (e.g., KNN, GBM, Random Forests, XGBoost) may still **Effectively** learn and distinguish differences.

3.3.4 Partitions dataset

Partitions the dataset into **80% training and 20% testing sets** as follow:

- Splits 80% of **molecular_descriptors_scaled** Matrix into **train_set_x**
- Splits 20% of **molecular_descriptors_scaled** Matrix into **test_set_x**
- Splits 80% of **core_clock_protein_cry$toxicity** Class into **train_set_y**
- Splits 20% of **core_clock_protein_cry$toxicity** Class into **test_set_y**

Given our dataset's **Small Size but High Dimensionality**, we allocate **20% for testing** rather than the typical **10%.** This ensures an Adequate Amount of data for a **Reliable and Effective Evaluation.**

R Codes split dataset into train_set_x, train_set_y (80%) & test_set_x, test_set_y (20%).

```
#############################################################################################
#  Splits dataset into train_set_x, train_set_y (80%) & test_set_x, test_set_y (20%)  #
#############################################################################################

set.seed(1)

test_index <- createDataPartition(core_clock_protein_cry$toxicity,
                                  times = 1, p = 0.2, list = FALSE)

test_set_x   <-  molecular_descriptors_scaled[test_index,]
test_set_y   <-  core_clock_protein_cry$toxicity[test_index]

train_set_x  <- molecular_descriptors_scaled[-test_index,]
train_set_y  <- core_clock_protein_cry$toxicity[-test_index]
```

R Codes that compute Proportion of the train_set_y is "Toxic"

```
# Proportion of the train_set_y is "Toxic"
mean(train_set_y =="Toxic")
```

```
## [1] 0.3235
```

R Codes that compute Proportion of the test_set_y is "NonToxic"

```
# Proportion of the test_set_y is "NonToxic"
mean(test_set_y  =="NonToxic")
```

```
## [1] 0.6571
```

3.3.5 Feature Importance

Feature Importance Technique ranks **1203 Molecular Descriptors/Features** of **core_clock_protein_cry** set based on their contribution to the Model's predictions using **Random Forest Algorithm. Feature Importance** can be derived from the average decrease in impurity or the mean decrease in accuracy when a **Molecular Descriptors** is excluded. Identify and keep the **Most Important Molecular Descriptors**, Molecular Descriptors with Low Importance will be removed. Finally, select **Top Most Relevant Molecular Descriptors** as MD2 set.

A) Model Optimization

* Fine-tune Models to improve Performance,involving Hyper-Parameter tuning & Cross-Validation.

* Optimize the Models by adjusting Parameters as following:

    * Sequentially increment mtry from 1 to 10 with increment by 1
    * Set Number of Tree (ntree) equal to 500
    * Set Number of Cross validation** equal to 5

R Codes Select Most Relevant Descriptors from Vars Importance using Random Forest.

```
#################################################################################################
# Select Most Relevant Molecular Descriptors from Vars Importance using Random Forest Algorithm #
#################################################################################################

#######################################################################
#  Optimized Parameters of Random Forest Algorithm                    #
#   * Sequentially increment "mtry" from 1 to 10 with increment by 1  #
#   * Set Number of Tree ("ntree") equal to 500                       #
#   * Set Number of "Cross Validation" equal to 5                     #
#######################################################################
set.seed(9)

train_ccpc_1203 <- caret::train(train_set_x, train_set_y , method="rf",
                                tuneGrid=data.frame(mtry=seq(1:10)),
                                ntree=500,
                                trControl=trainControl(method="cv", number=5),
                                importance= TRUE)
```

B) List Vars Importance

R Codes that generate Vars Importance

```
# Process and Generate Vars Importance of train_ccpc_rf
importance_rf_1203 <- varImp(train_ccpc_1203)
print(importance_rf_1203)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 1203)
##
##            Importance
## MATS1e          100.0
## SaasC            98.5
## apol             97.7
## GATS8i           97.7
## SpMin5_Bhe       96.8
## MATS1p           95.8
## VE2_D            95.3
## MAXDN            94.8
## SpAD_Dt          94.3
## ATSC8p           94.3
## GATS5i           93.8
## GATS6e           93.6
## MLogP            93.6
## SpMax7_Bhv       93.0
## BIC0             92.8
## minwHBa          91.8
## Si               91.8
## BCUTc_1l         91.3
## GATS1p           91.3
## khs.tsC          91.0
```

C) Select Top 4 Most Relevant Molecular Descriptors as MD2 set.

R Codes that rank Vars Importance in descending order

```
# Vars Importance
ccpc_importance_1203 <- rownames(importance_rf_1203$importance)[order(-importance_rf_1203$importance[,1
#ccpc_importance_1203[1:4]
```

MATS1e, SaasC, apol, GATS8i are selected as MD2 set.

R Codes that predict and compute Overall Accuracy using confusionMatrix.

```
# Compute Overall Accuracy for Molecular Descriptors/Feature Prioritization Method
predict_ccpc_1203 <- predict(train_ccpc_1203, test_set_x)
accuracy_ccpc_1203 <- confusionMatrix(table(predict_ccpc_1203, test_set_y),mode="everything")
#accuracy_ccpc_1203
accuracy_ccpc_1203$overall["Accuracy"]
```

```
## Accuracy
##   0.6571
```

Overall Accuracy of 0.6571 is computed.

3.4 Molecular Descriptors/Features Prioritization

Molecular Descriptors/Features Prioritization involves pre-processing steps that **Enhance** data quality, **Reduce Redundant Molecular Descriptors**, and ensure the development of Robust and Reliable Models. Handling **Class Imbalance** in the data is crucial for constructing accurate predictive Models. **Feature Selection** plays a Significant role in reducing **dimensionality** by selecting only the **Most Relevant Molecular Descriptors/Features** for each **Model**, thus improving the Performance of **Toxicity Predictions.**

A) Formulating of MD1, MD2, RFE and MDU set are as follows:

MD1 set

* Select Top pairwise Molecular Descriptors generated from Vars Importance of Benchmark set using Random Forest Algorithm to formulate appropriate Molecular Descriptors "MD1" set. (Refer to Section 3.2.4 for Vars Importance of "MD1" set)

MD2 set

* Select Top ranked Molecular Descriptors/Features generated from Vars Importance of "1203 Molecular Descriptors" using Random Forest Algorithm to formula appropriate Molecular Descriptors "MD2" set. (Refer to Section 3.3.5 for Vars Importance of "MD2" set)

RFE set

* Perform RFE function using Caret Package in R Environment. It recursively removes the least important Features until specified Number of Features is reached as "REF" set.

MDU set

* The MD1", "MD2" and "RFE" set are used to formulate different combination of Molecular Descriptors

B) Formulation of the "MDU" set will be conducted in the RStudio environment as follows:

* Utilize trial and error strategy as methodology,offering flexibility in handling testing procedure.

* The accuracy of "MDU" sets will be tested using Machine Learning Algorithms such as KNN, GBM, Random Forest and XGBoost.

* The Accuracy of various "MDU" set combinations will be compared with the Benchmark Accuracy (0.6571) Sets exceeding the Benchmark will be selected and added to the "Candidate MDU" sets. Variances in Accuracy indicate interactions between Molecular Descriptors,which explain Variability among them.

* Prioritized Molecular Descriptors (PMD) from the "Candidate MDU" sets with the highest Accuracy will be selected as "PMD" Set for each Machine Learning Model.

* These Prioritized Molecular Descriptors of each Machine Learning Model will be applied for Final Model training.

C) For KNN Model, we will adopt Redundant Feature Elimination for Descriptors Selection.

```
* Recursive Feature Elimination (RFE) is a Feature Selection function of Caret package in R.
  RFE recursively removes the least important Features until specified number of Features is reached.
  Perform RFE to achieve better Performance and Enhance Robustness of the Models.
```

For each ML Algorithm (KNN, GBM, Random Forest, XGBoost), Prioritized Molecular Descriptors/Features Selection will be employed for Training the Model.

D) Summary of Molecular Descriptors Prioritization (MD1 Set , MD2 Set and RFE Set)

Table 10: MD1 ,MD2 and RFE Set - Molecular Descriptors

SET	Descriptors
MD1 Set	MDEC_23, GATS8s
MD2 Set	MATS1e, SaasC, apol, GATS8i
RFE Set	ZMIC1, ATSC8i, ATSC7p, MATS2s

E) Formation of MDU Set

- 1 or 2 Descriptors from MD1 Set + Any Combination of MD2 Set OR None + Any Combination of RFE Set of OR None

  (e.g., **MDU Set contains Molecular Descriptors MDEC_23, ZMIC1, ATSC8i**)

F) Summary of Prioritized Molecular Descriptors of each Model (PMD Set)

Table 11: PMD Set - Molecular Descriptors Prioritization

Model	Prioritized_Molecular_Descriptors
Random Forest	MDEC_23, GATS8s, MATS1e, SaasC
K-Nearest Neighbor(KNN)	MDEC_23, ZMIC1, ATSC8i
Gradient Boosted Machine (GBM)	MDEC_23, GATS8s, MATS1e, SaasC
Extreme Gradient Boosting (XGBoost)	MDEC_23, GATS8s, MATS1e, SaasC, apol, GATS8i

G) Definition of Prioritized Molecular Descriptors:

* MDEC_23: This Descriptor evaluate Molecular distance edge Descriptors for C.^[docs.ochem.eu]

* apol: This Descriptor calculates sum of atomic polarizabilities.

* ATSC8i: Stands for Autocorrelation of Topological Structure,lag 8, weighted by Sanderson electronegativities. ^[www.vcclab.org]

* MATS1e: Stands for Moran Autocorrelation of Topological Structure, lag1, weighted by atomic Sanderson electronegativities.

* GATS8s: Measures the similarity of a Molecule's properties at different distances (lag 8) within the Molecule, weighted by I-state.^[www.alvascience.com]

* GATS8i: Stands for Geary Autocorrelation of Topological Structure, lag 8, weighted by atomic Sanderson electornegativities.

* ZMIC1: Stands for Z-modified Information Content Index(neighborhood symmetry of 1-order) ^[www.peer.com]

4 Machine Learning Modelling Approach and Algorithm

4.1 Modelling Approach

A) Data Pre-processing

I Scale and Normalize Molecular Descriptors/Features, Splitting the dataset into 80% training and 20% test sets for the Capstone Project. I choose a 20% test set instead of the 10% due to the dataset's small size but high dimensionality. This Approach ensures we have a sufficient quantity of data for Reliable testing purposes.

B) Hyper-Parameters Optimization and Cross Validation

I fine-tune each Model to improve Performance through Hyper-Parameter Tuning and Cross-Validation. By Optimizing Hyper-Parameters, I find the Optimal configuration of the Machine Learning Algorithm to achieve Accuracy, influencing how the Model updates its weights and converges to a solution.

Standard Parameters are carefully selected to Optimize the Model's Performance. Depending on Model complexity, I initially set up each Model with standard Hyper-Parameters to quickly gauge Performance and then adjust based on specific requirements. Fine-tuning with appropriate Hyper-Parameters during training ensures faster convergence, achieving an Optimized Model.

Cross-validation, using both 5-Fold and 10-Fold Method for different Model, is crucial for ensuring the Model's Robustness and Reliability. This Approach Balances Cross-Validation of the Model's performance with manageable run-time, maintaining Efficiency while achieving target Accuracy.

C) Machine Learning Modelling Approach

An Ensemble Model Approach will be implemented using the following Models:

* Random Forest
* Gradient Boosted Machine (GBM)
* K-Nearest Neighbor (KNN)
* Extreme Gradient Boosting (XGBoost)

Using Multiple Models of KNN, GBM, Random Forest and XGBoost for implementing an Ensemble Model as Machine Learning Modelling Approach is recommended, as combining Multiple Models can often yield better Accuracy than any Single Model.

Each Model will be trained separately with Optimized Hyper-Parameters and evaluate on the test set. Predictions will be made and an Ensemble created from these Four Models to generate a Majority Prediction of Toxicity Class. The Performance of theses Four Model and Ultimate Ensemble Model will be evaluated using the confusionMatrix function of Caret Package in R.

4.1.1 K-Nearest Neighbors Model (KNN)

Prioritized Molecular Descriptors (MDEC_23 + ZMIC1 + ATSC8i) were applied for training the KNN model.

K-Nearest Neighbors (KNN) is a Distance-based Learning Algorithm commonly used for Classification. Scaling and Normalization are essential for KNN to Perform Optimally because it relies on Distance Metrics. This method Predicts outcomes by calculating Distances between data points, making the Scaling and Normalization of Molecular Descriptors critical for Optimal Performance.

Hyper-Parameter Optimization:

A) k=seq(3, 31, 1) : Number of Neighbors for calculating Distances between data points

* By setting the value of "k" from 1 to 31 with increment by 1, an Optimal Number of Neighbors can be determined.

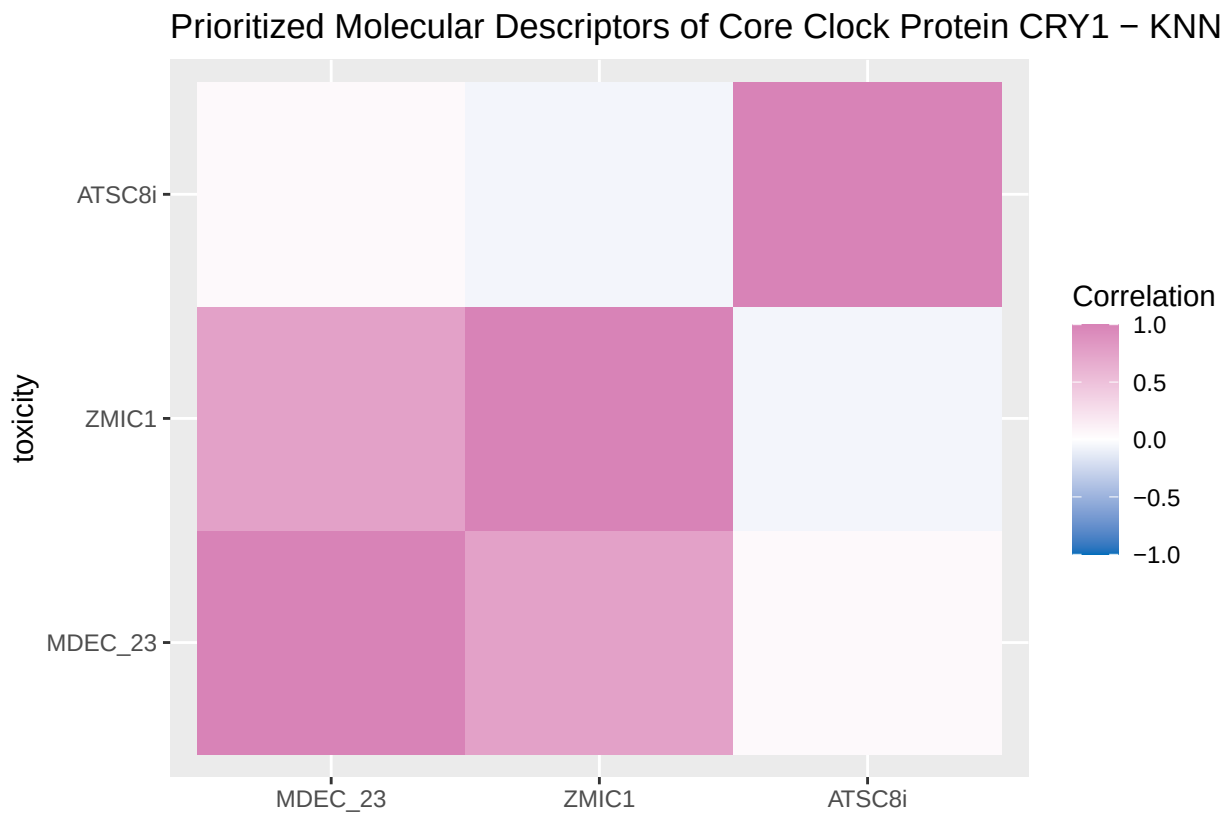
To enhance the model's Robustness and Reduce the risk of Over-fitting, 10-Fold Cross-Validation Techniques were employed.

Model achieved F1-Score of 0.851 and an Overall Accuracy of 0.800 by confusionMatrix.

The Best KNN Model is "23-Nearest Neighbors" Model. (k=23)

R Codes employ Prioritized Descriptors for KNN Training & ggplot Correlation Heat Map

```
#############################################################################################
# Apply Prioritized Molecular Descriptors on K-Nearest Neighbors Model (KNN) Training #
#############################################################################################

# Generate df_ccpc_data dataset comprising Prioritized Molecular Descriptors of KNN Model
df_ccpc_data <- ccpc_data  %>% select(MDEC_23, ZMIC1, ATSC8i, -Class)

# Compute Correlation coefficient of df_ccpc_data
cor_ccpc_data <- cor(na.omit(df_ccpc_data[, unlist(lapply(df_ccpc_data, is.numeric))]))

#############################################################################################
# Heat Map -  Prioritized Molecular Descriptors Correlation coefficient of KNN Model #
#############################################################################################
df_cor_ccpc_data <- melt(cor_ccpc_data)
colnames(df_cor_ccpc_data) <- c("molecular_descriptors","toxicity","value")

ggplot(df_cor_ccpc_data , aes(x=molecular_descriptors,y=toxicity, fill=value)) + geom_tile() +
  scale_fill_gradient2(low = "#006EBB", high = "#D883B7" , mid = "white",
                       midpoint = 0, limit = c(-1, 1), space = "Lab", name="Correlation") +
  labs(x="", title="Prioritized Molecular Descriptors of Core Clock Protein CRY1 - KNN")
```



R Codes create core_clock_protein_cry incorporating descriptors matrix & toxicity vector

```
# Generate molecular_descriptors Matrix and toxicity Vector
molecular_descriptors <- df_ccpc_data %>% as.matrix()
toxicity <- factor(ccpc_data$Class)

# Generate "core_clock_protein_cry" set for Normalization
core_clock_protein_cry <- list(molecular_descriptors=molecular_descriptors,
                               toxicity=toxicity )
```

R Codes that List last 6 rows of KNN Model

```
# Display last 6 rows of df_ccpc_data- KNN
tail(df_ccpc_data)
```

```
## # A tibble: 6 x 3
##   MDEC_23 ZMIC1 ATSC8i
##     <dbl> <dbl>  <dbl>
## 1   31.6   60.5 -17.7
## 2   13.6   44.0  -7.86
## 3   33.2   74.6 -10.7
## 4   28.4   67.3 -37.4
## 5   26.4   65.5  19.2
## 6    9.88  35.7   6.54
```

R Codes for scaling and normalization of KNN Model

```
#########################################################################
#  Scaling and Normalizing data for K-Nearest Neighbors Model (KNN)  #
#########################################################################

# Compute Mean of each Column
col_means <- colMeans(core_clock_protein_cry$molecular_descriptors, na.rm=TRUE)


# Compute Standard Deviations of each Column
col_sds <- colSds(core_clock_protein_cry$molecular_descriptors, na.rm=TRUE)


# Subtract Column Means
molecular_descriptors_centered <- sweep(core_clock_protein_cry$molecular_descriptors,
                                        2, col_means, FUN="-")

# Divide by Column Standard Deviations
molecular_descriptors_scaled <- sweep(molecular_descriptors_centered,
                                      2 , col_sds, FUN="/")
```

R Codes that List 6 rows of normalized KNN Model

```
# Display first 6 rows of molecular_descriptors_scaled - KNN
head(molecular_descriptors_scaled)
```

```
##      MDEC_23   ZMIC1   ATSC8i
## [1,]  2.0861  2.0171 -0.46708
## [2,]  0.5322  1.3578 -0.25912
## [3,] -0.1573 -0.3346  0.74669
## [4,]  0.1445 -0.2164  0.40247
## [5,]  1.3483  0.6953 -0.08054
## [6,]  1.3133  2.3638 -0.19781
```

R Codes split 80% training set & 20% test set of datset for KNN Model Training

```
##############################################################################
#  Generate 80% training set and 20% test set for K-Nearest Neighbors Model  #
##############################################################################
set.seed(1)

test_index <- createDataPartition(core_clock_protein_cry$toxicity, times = 1, p = 0.2, list = FALSE)

test_set_x   <-  molecular_descriptors_scaled[test_index,]
test_set_y   <-  core_clock_protein_cry$toxicity[test_index]

train_set_x  <- molecular_descriptors_scaled[-test_index,]
train_set_y  <- core_clock_protein_cry$toxicity[-test_index]
```

A) Hyper-Parameters Optimization of K-Nearest Neighbors Model (KNN)

1) Fine-tune the Models to improve the performance, involving Hyper-Parameter tuning and Cross-Validation.

2) Optimize the Models by setting parameters as following:

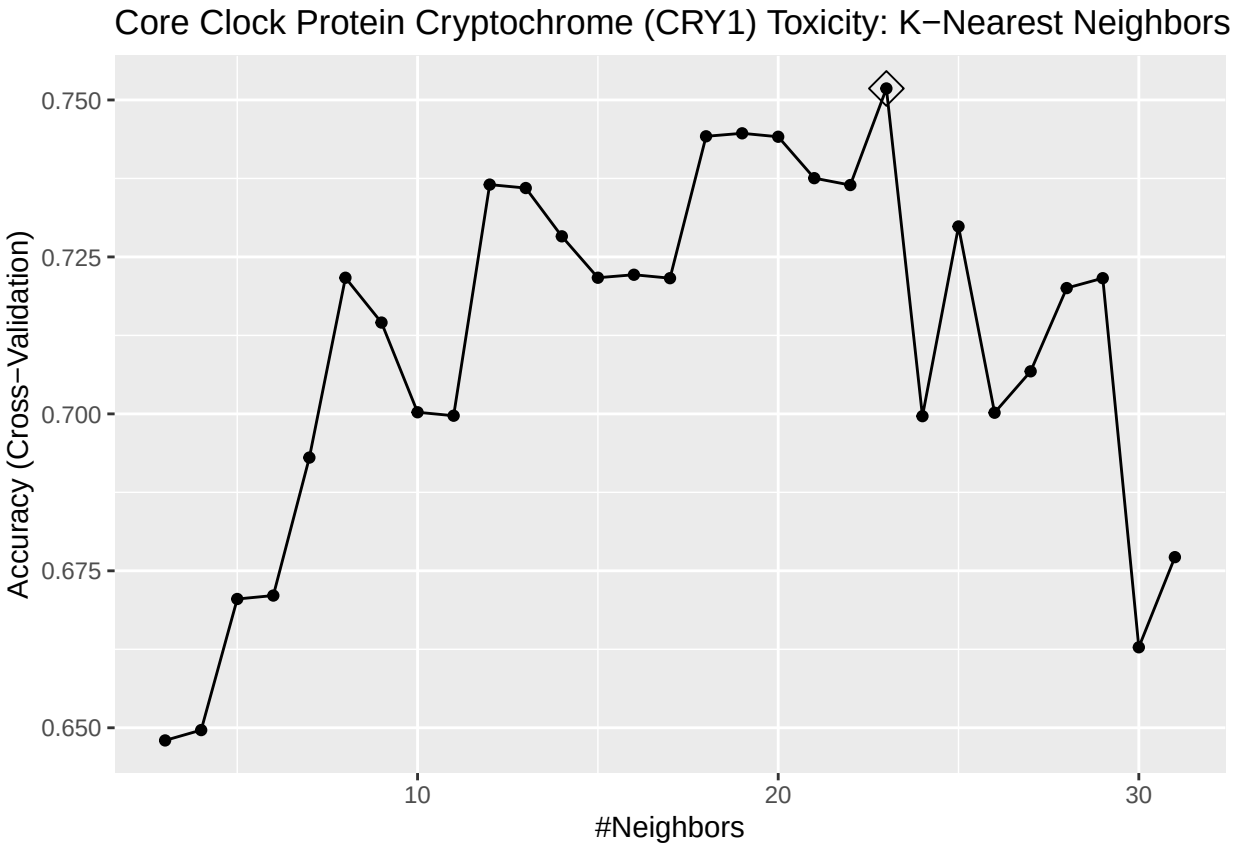
   * Sequentially increment "k" from 3 to 31
   * Set Number of "Cross validation" equal to 10

R Codes for training KNN Model using Optimized Parameters

```
#######################################################
#      K-Nearest Neighbors Model (KNN) Training       #
#######################################################

#######################################################
# Optimize Parameters of K-Nearest Neighbors Model  #
#  * Sequentially increment  "k" from 3 to 31       #
#  * Set Number of "cross validation" equal to 10   #
#######################################################
set.seed(7)

control <- trainControl(method="cv", number=10)

train_ccpc_knn <- caret::train(train_set_x, train_set_y ,
                               method = "knn",
                               tuneGrid = data.frame(k = seq(3, 31, 1)),
                               trControl = control)
```

R Codes ggplot no of Neighbors vs Accuracy and list k and Final Model of KNN

```
# ggplot number of Neighbors vs Accuracy
ggplot(train_ccpc_knn, highlight = TRUE) +
  ggtitle("Core Clock Protein Cryptochrome (CRY1) Toxicity: K-Nearest Neighbors model (KNN)")
```




```
#Display the Best "k" Number of Neighbors for KNN Model
train_ccpc_knn$bestTune
```

```
##     k
## 21 23
```

```
#Display Final Model of KNN
train_ccpc_knn$finalModel
```

```
## 23-nearest neighbor model
## Training set outcome distribution:
##
## NonToxic    Toxic
##       92       44
```

R Codes predict and compute F1-Score and Accuracy for KNN Model using confusionMatrix

```
# Makes Prediction on test set and Compute Overall Accuracy of KNN Model
predict_ccpc_knn <- predict(train_ccpc_knn, test_set_x)
accuracy_ccpc_knn <- confusionMatrix(table(predict_ccpc_knn, test_set_y),mode="everything")
accuracy_ccpc_knn$byClass["F1"]
```

```
##     F1
## 0.8511
```

```
accuracy_ccpc_knn$overall["Accuracy"]
```

```
## Accuracy
##      0.8
```

4.1.3 Random Forests Model

Prioritized Molecular Descriptors (MDEC_23 + GATS8s + MATS1e + SaasC) were applied for training Random Forest Model.

Random Forest is an ensemble method that builds multiple **Molecular Descriptors Decision Trees** and combines their outputs to achieve more **Accurate** and Stable Predictions. During training, it constructs a multitude of **Decision Trees** and uses their output to determine the class of the individual **Molecular Descriptors Trees.** The idea behind this method is that a group of weak learners (Decision Trees) can collaborate to form a strong learner. This Approach Significantly reduces the **risk of Over-fitting** compared to relying on a single Molecular Descriptors Decision Tree.

Hyper-Parameter Optimization

A) ntree=500: Number of Trees to grow in the Forest.

```
* Increasing the value of "ntree" in the Model can enhance Stability and Accuracy.  However, it also
  raises computational demands.  I begin by setting the value of "ntree" to a large number(e.g. 200).
  Then I Fine-Tune this value by monitoring the validation error as the Number of Trees increases.
```

B) mtry=seq(1:10): Number of Molecular Descriptors/Features to consider when looking for Best Split in each node.

```
* The Hyper-Parameter "mtry" adjust the number of variables randomly sampled as candidates at each
  Split when building the Trees in the Model. This Parameter controls the randomness and complexity
  of the Model.  Tuning "mtry" helps find the Optimal Number of Molecular Descriptors/Features,
  Balancing between Under-fitting and Over-fitting. By setting the value of "mtry" from 1 to 10,
  Optimal Number of Neighbors can be determined.
```

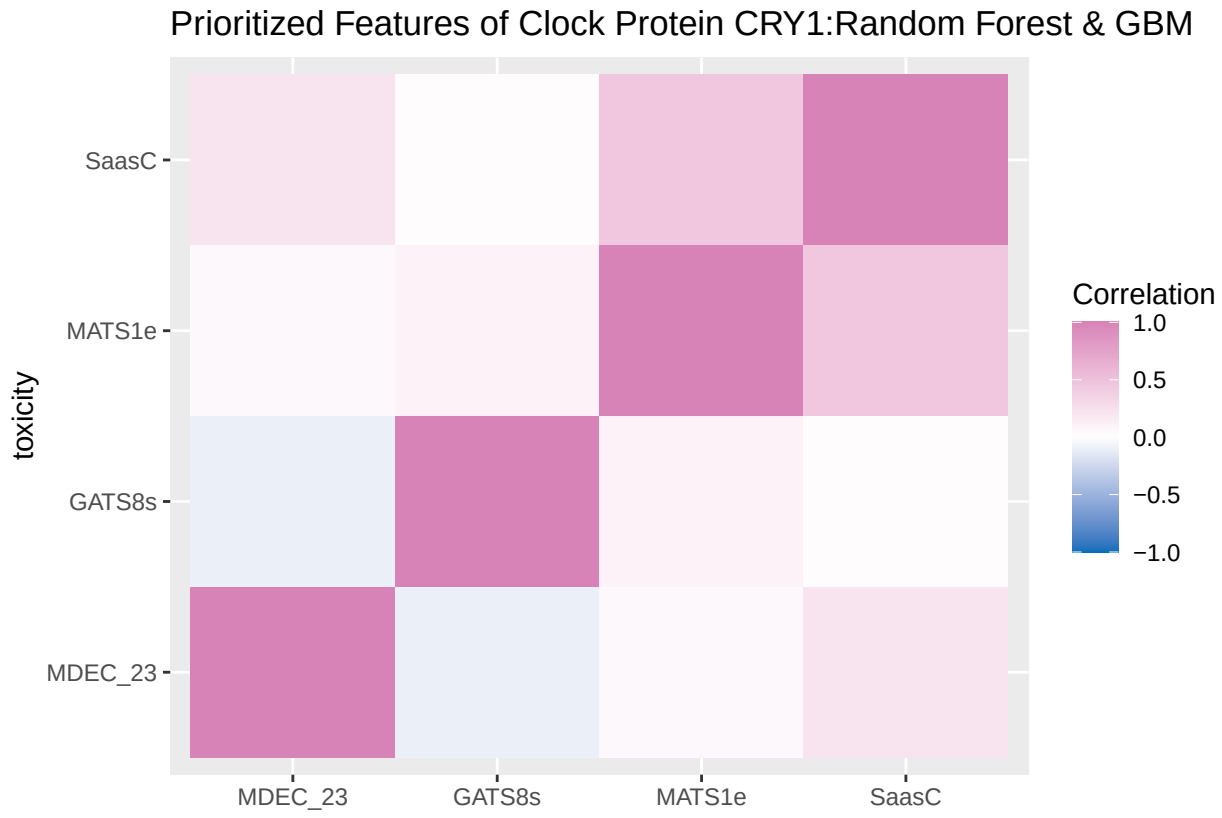
The **5-Fold Cross-Validation** Technique is utilized during the Hyper-Parameter Tuning process. This involves Splitting the Molecular Descriptors into several Folds and training the Model multiple times, with each Fold serving as the validation set in turn. The goal is to find the **Hyper-Parameter** values that yield Best **Performance** across all Folds. Tuning the Hyper-Parameters "mtry" and "ntree," in combination with employing **Cross-Validation** Techniques, aids in building a **Robust Model.**

The Model achieved F-Score of 0.840 and Overall Accuracy of 0.771 by confusionMatrix.

The Best Random Forest Model is ntree=500 and mtry=2 .

R Codes employ Prioritized Descriptors for Heat Map, Random Forest and GBM Model

```
#################################################################
# Apply Prioritized Molecular Descriptors on Random Forest  #
# and Gradient Boosted Machine Model (GBM) Training         #
#################################################################

# Generate df_ccpc_data dataset comprising Prioritized Molecular Descriptors (Random Forest & GBM Model)
df_ccpc_data <- ccpc_data  %>% select(MDEC_23, GATS8s, MATS1e, SaasC, -Class)

# Compute Correlation coefficient of df_ccpc_data
cor_ccpc_data <- cor(na.omit(df_ccpc_data[, unlist(lapply(df_ccpc_data, is.numeric))]))

#########################################################################################################
# Heat Map -  Prioritized Molecular Descriptors Correlation coefficient of Random Forest & GBM Model #
#########################################################################################################
df_cor_ccpc_data <- melt(cor_ccpc_data)
colnames(df_cor_ccpc_data) <- c("molecular_descriptors","toxicity","value")
ggplot(df_cor_ccpc_data , aes(x=molecular_descriptors,y=toxicity, fill=value)) +
  geom_tile() +
  scale_fill_gradient2(low = "#006EBB", high = "#D883B7" , mid = "white",
                       midpoint = 0, limit = c(-1, 1), space = "Lab", name="Correlation") +
  labs(x="", title="Prioritized Features of Clock Protein CRY1:Random Forest & GBM")
```



R Codes create core_protein_cry incorporating descriptors matrix and toxicity vector (For both Random Forest and GBM Model)

```
# Generate molecular_descriptors Matrix and toxicity Vector
molecular_descriptors <- df_ccpc_data %>% as.matrix()
toxicity <- factor(ccpc_data$Class)

# Generate core_clock_protein_cry for Normalization
core_clock_protein_cry <- list(molecular_descriptors=molecular_descriptors,
                               toxicity=toxicity)
```

R Codes that List 6 Rows of df_ccpc_data of Random Forest and GBM Model

```
# Display column names of df_ccpc_data
# colnames(df_ccpc_data)

# Display last 6 rows of df_ccpc_data
tail(df_ccpc_data)
```

```
## # A tibble: 6 x 4
##   MDEC_23 GATS8s  MATS1e  SaasC
##     <dbl>  <dbl>   <dbl>  <dbl>
## 1   31.6   1.36  -0.0444  3.57
## 2   13.6   0.795 -0.0232  0.929
## 3   33.2   0.985 -0.0362  2.70
## 4   28.4   0.910  0.120   6.64
## 5   26.4   1.27  -0.0306  4.84
## 6    9.88  0.272 -0.0603 -0.339
```

R Codes for Scaling and Normalization of Random Forest and GBM Model

```
#####################################################################
#  Scaling and Normalizing data for Random Forest and GBM Model  #
#####################################################################

# Compute Mean of each Column
col_means <- colMeans(core_clock_protein_cry$molecular_descriptors, na.rm=TRUE)

# Compute Standard Deviations of each Column
col_sds <- colSds(core_clock_protein_cry$molecular_descriptors, na.rm=TRUE)

# Subtract Column Means
molecular_descriptors_centered <- sweep(core_clock_protein_cry$molecular_descriptors,
                                        2, col_means, FUN="-")

# Divide by Column Standard Deviations
molecular_descriptors_scaled <- sweep(molecular_descriptors_centered,
                                      2 , col_sds, FUN="/")
```

R Codes List 6 Rows of Normalized molecular_descriptors_scaled (Random Forest and GBM)

```
# Display first 6 rows of molecular_descriptors_scaled
head(molecular_descriptors_scaled)
```

```
##      MDEC_23  GATS8s  MATS1e   SaasC
## [1,]  2.0861 -0.1490 -0.5318 -1.7377
## [2,]  0.5322  1.1421  0.3064  0.1987
## [3,] -0.1573 -0.7000 -1.6525 -0.5686
## [4,]  0.1445 -0.2046 -1.8810 -0.7597
## [5,]  1.3483 -0.7804  2.6592 -0.1125
## [6,]  1.3133 -0.4970 -0.6696 -0.7899
```

R Codes create 80% training set & 20% train set for Random Forest and GBM Model

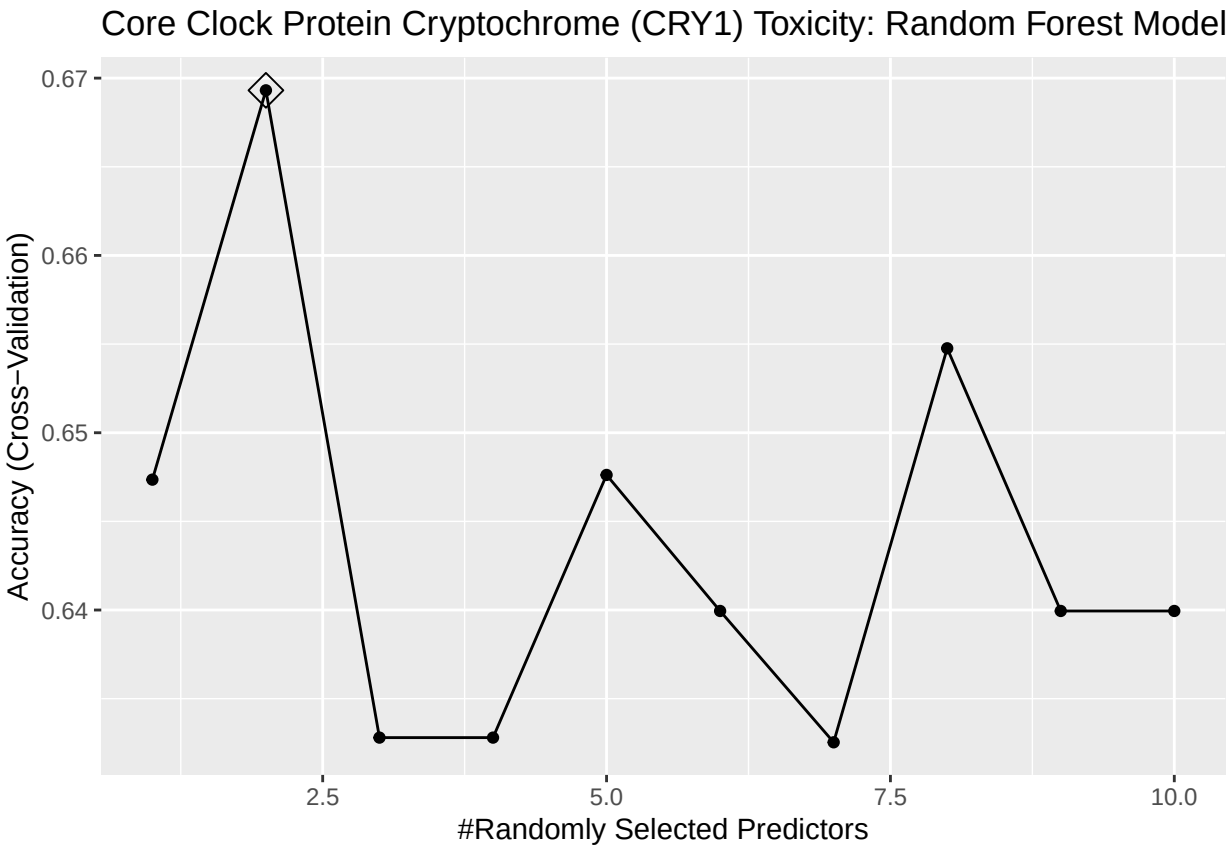
```
#############################################################################
#  Generate 80% training set and 20% test set for Random Forest and GBM  Model #
#############################################################################
set.seed(1)
test_index <- createDataPartition(core_clock_protein_cry$toxicity, times = 1, p = 0.2, list = FALSE)

test_set_x   <-  molecular_descriptors_scaled[test_index,]
test_set_y   <-  core_clock_protein_cry$toxicity[test_index]

train_set_x  <- molecular_descriptors_scaled[-test_index,]
train_set_y  <- core_clock_protein_cry$toxicity[-test_index]
```

A) Hyper-Parameter Optimization of Random Forest Model

1) Fine-tune the Models to improve the performance, involving Hyper-Parameter tuning and Cross-Validation.
2) Optimize the Models by setting parameters as following:
   - Sequentially increment "mtry" from 1 to 10 with increment by 1
   - Set Number of Tree ("ntree") equal to 500
   - Set Number of "Cross validation" equal to 5

```
#########################################################################
#                   (Random Forest Model Training)                      #
#                                                                       #
# Optimize Hyper-Parameters of Random Forest Model                      #
#                                                                       #
#  * Sequentially increment "mtry" from 1 to 10 with increment by 1     #
#  * Set Number of Tree ("ntree") equal to 500                          #
#  * Set Number of "cross validation" equal to 5                        #
#########################################################################
set.seed(9)
train_ccpc_rf <- caret::train(train_set_x, train_set_y , method="rf",
                              tuneGrid=data.frame(mtry=seq(1:10)),
                              ntree=500,
                              trControl=trainControl(method="cv", number=5),
                              importance= TRUE)
```

```r
# Plot "mtry" value vs Accuracy
ggplot(train_ccpc_rf, highlight = TRUE) +
  ggtitle("Core Clock Protein Cryptochrome (CRY1) Toxicity: Random Forest Model")
```




```r
#Display the best tine of Random Forest Model
train_ccpc_rf$bestTune
```

```
##   mtry
## 2    2
```

```r
# Makes Prediction on test set and Compute Overall Accuracy of Random Forest Model
predict_ccpc_rf <- predict(train_ccpc_rf, test_set_x)
accuracy_ccpc_rf <- confusionMatrix(table(predict_ccpc_rf, test_set_y),mode="everything")
accuracy_ccpc_rf$byClass["F1"]
```

```
##   F1
## 0.84
```

```r
accuracy_ccpc_rf$overall["Accuracy"]
```

```
## Accuracy
##   0.7714
```

4.1.4 Gradient Boosted Machine Model (GBM)

Prioritized Molecular Descriptors (MDEC_23 + GATS8s + MATS1e + SaasC) were applied for training GBM Model.

Gradient Boosted Machine (GBM) build an ensemble of shallow and weak **Molecular Descriptor Trees**, with each **Tree** learning and improving on the previous one. The core concept of Boosting is to sequentially add new Models to the ensemble. During each iteration, a new weak base-learner Model is trained based on errors of the existing ensemble.

One disadvantage of this Approach is that it continually strives to minimize all errors, which can lead to over-emphasizing outliers of **Molecular Descriptors** and ultimately cause Over-fitting. To mitigate this, I will employ a **5-Fold Cross-Validation** Technique. This method helps ensure the **model** generalizes well by neutralizing Over-fitting and increasing **Robustness**, thereby achieving better **Accuracy.**

Hyper-Parameters Optimization:

A) n.trees=seq(100,1000,by=50): Total Number of Molecular Descriptors Trees to fit Model.

```
* GBM uses a default Number of Trees of 100, which is rarely sufficient. I try to find an Optimal
  Number of Tree that minimize the loss function with Cross Validation by setting the value
  of "n.trees" from 100 to 1000 with increment by 50.
```

B) interaction.depth=c(1,3,5): Number of Splits ("interaction.depth") in each Molecular Descriptors Tree, which controls the complexity of the Boosted ensemble.

```
* Set up value, Trees of different depths to see which works Best.

    "interaction.depth" 1: This makes each Tree a Simple Tree with only one Split.

    "interaction.depth" 3 and 5: These settings allow for complex Trees with more Splits,
                                 potentially capturing more patterns.

Usually, interaction.depth > 1 are used because they can improve Model Performance by capturing
more details.  However, it's rare to need a interaction.depth > 10, as very deep Molecular
Descriptors Trees can Over-fit.  I chosen interaction.depth=c(1,3,5) provide a Balanced Approach
to Model training, allowing us to evaluate the Performance across a range of Tree complexities
and determine the Optimal depth.
```

C) shrinkage=c(0.01,0.1,0.3): "shrinkage" is also called "learning rate", controls how quickly or slowly the Algorithm proceeds down the gradient descent. (Also Known as "Controls Step Size")

* The value of "shrinkage" determines how much each Molecular Descriptors tree contributes to the overall Model. Smaller values reduce the chance of Over-fitting but also increase the time to find the Optimal fit. The default value is "shrinkage" of 0.001. This is a very small learning rate and requires a large Number of Molecular Descriptors Trees to find the minimum error.

  I set the shrinkage = c(0.01, 0.1, 0.3) mean adjusting different values for the learning rate to observe how each affects the Model's Performance.

  "shrinkage" 0.01: Very small learning rate. This minimizes the risk of Over-fitting but will require a substantial Number of Trees to achieve the Optimal Model.

  "shrinkage" 0.1: A moderate learning rate that Balances the risk of Over-fitting and the computational time required.

  "shrinkage" 0.3: Larger learning rate, which means the Model will converge faster but may risk Over-fitting.

  Fine-Tune value of "shrinkage" impacts both the Accuracy and computational Efficiency to achieve the Best Performance.

D) n.minobsinnode=c(10,20): Minimum number of observations that must exist in a node for it to be considered for Splitting.

* The value of "n.minobsinnode" controls the complexity of the Molecular Descriptors Trees grown. Higher values of "n.minobsinnode" prevent the creation of Splits that would result in very small leaf nodes, which helps to prevent Over-fitting and make the Model more Robust.

  I set n.minobsinnode=c(10,20) for Algorithm to consider different minimum node sizes within the rangefrom 10 to 20. The Model will evaluate its Performance for each of these values.
  The 5-Fold Cross-Validation is used during the Tuning process to find "n.minobsinnode" value that yields the Best Performance across all Folds.

  "n.minobsinnode" 10: Lower values (e.g. 1,5,10) mean that fewer observations are required in a node for it to be Split which can capture more detail from the training data but may also lead to Over-fitting.

  "n.minobsinnode" 20: Higher values (e.g. 20,50,100) mean that more observations are required in a node for it to be Split which can help to prevent Over-fitting.

  Higher value ("n.minobsinnode" 20) tend to produce Model with higher bias but lower variance, while lower value ("n.minobsinnode" 10) produce Model with lower bias but higher variance.
  I employ both values in Balancing between Over-fitting and Under-fitting to Achieve Better Performance of the Model.

The Model achieved F1-Score of 0.816 and an Overall Accuracy of 0.743 by confusionMatrix.

The Best GBM Model is n.trees=150, interaction.depth=5, shrinkage=0.1 and n.minobsinnode=10.

Hyper-Parameter Optimization of Gradient Boosted Machine Model (GBM)

1) Fine-tune the Models to improve the performance, involving Hyper-Parameter tuning and Cross-Validation.

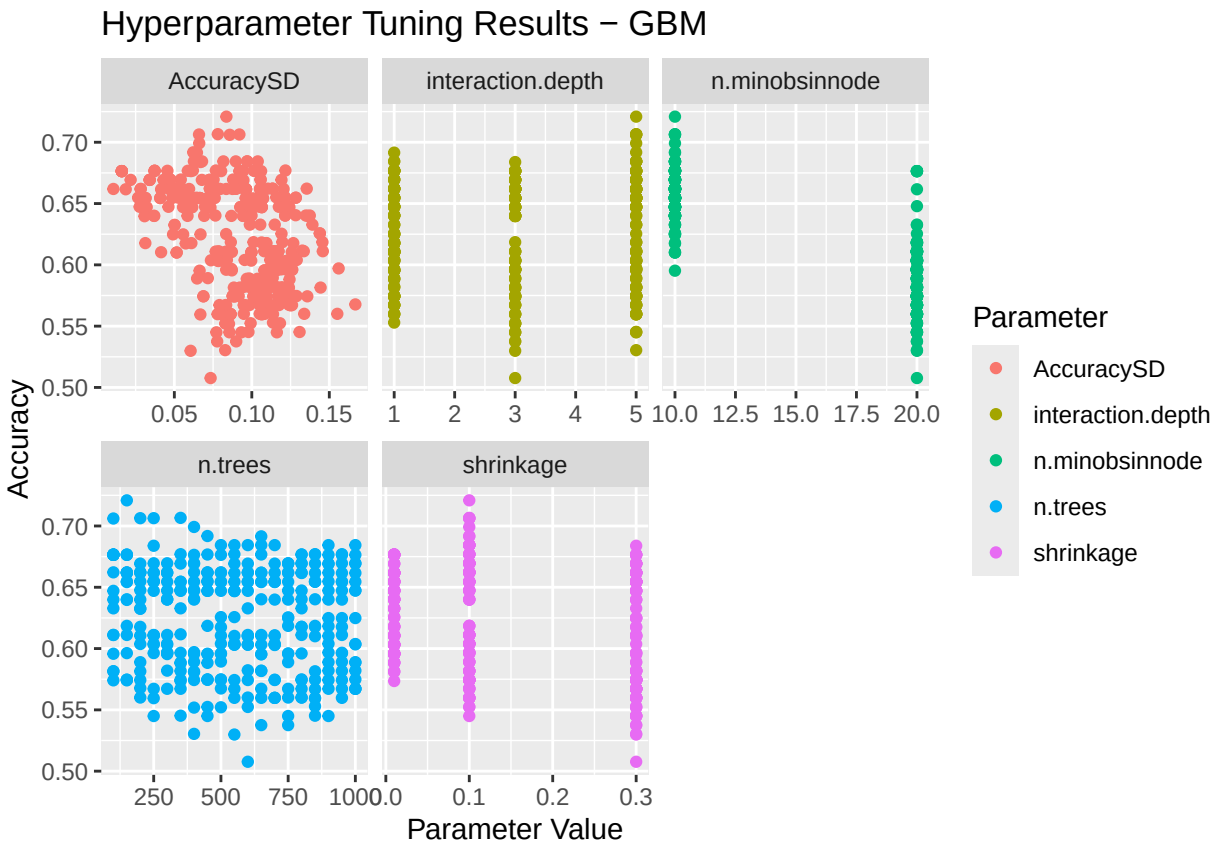
2) Optimize the Models by setting parameters as following:

   * Sequentially increment "n.trees" from 100 to 1000 with increment by 50
   * Set Number of splits ("interaction.depth") in each tree equal 1,3 and 5
   * Set Learning Rate ("shrinkage") equal to 0.01, 0.1 and 0.3
   * Set n.minobsinnode equal to 10 and 20
     (Minimum number of observations must exist in a node for it to be considered for splitting)
   * Set Number of "Cross validation" equal to 5

```
#########################################################
#                                                       #
#    Gradient Boosted Machine Model (GBM) Training      #
#                                                       #
#########################################################


#########################################################################################################################
# Optimize Parameters of Gradient Boosted Machine (GBM) Model                                                           #
#                                                                                                                       #
#  * Sequentially increment "n.trees" from 100 to 1000 with increment by 50                                             #
#  * Set Number of splits ("interaction.depth") in each tree equal 1,3 and 5                                            #
#  * Set Learning Rate ("shrinkage") equal to 0.01, 0.1 and 0.3                                                         #
#  * Set n.minobsinnode equal to 10 and 20                                                                              #
#    (Minimum number of observations that must exist in a node for it to be considered for splitting)                   #
#  * Set Number of "cross validation" equal to 5                                                                        #
#########################################################################################################################

set.seed(9)

control <- trainControl(method="cv",number=5)

grid <- expand.grid(n.trees = seq(100, 1000, by = 50),
                    interaction.depth=c(1,3,5),
                    shrinkage=c(0.01,0.1,0.3),
                    n.minobsinnode=c(10,20))

train_ccpc_GBM <- train(train_set_x,train_set_y, method="gbm",
                        trControl=control,
                        tuneGrid=grid,
                        verbose=FALSE)
```

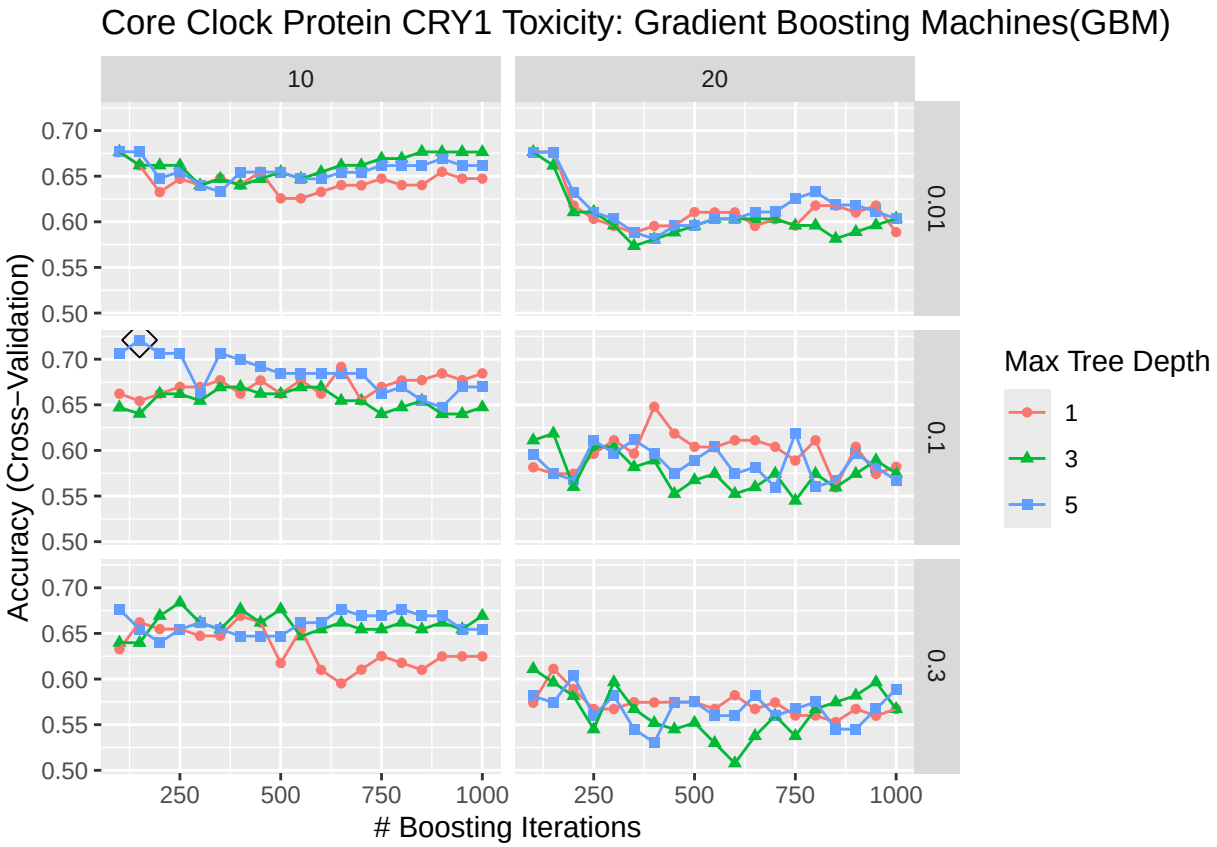
R Codes that Convert results to long format then ggplot Hyper-Parameters of GBM Model

```
# results contain Hyper-Parameters of GBM Model
results <- train_ccpc_GBM$results

# Convert the results to a long format for ggplot -GBM
results_long <- results %>% pivot_longer(cols = -c(Accuracy, Kappa, KappaSD),
                                         names_to = "Parameter",
                                         values_to = "Value")

# ggplot Hyper-Parameters of GBM Model
ggplot(results_long, aes(x = Value, y = Accuracy, color = Parameter)) +
  geom_point() +
  facet_wrap(~ Parameter, scales = "free_x") +
  labs(title = "Hyperparameter Tuning Results - GBM", x = "Parameter Value", y = "Accuracy")
```



R Codes that ggplot Accuracy vs Boosting Iterations then display Best GBM Model

```
# ggplot Accuracy vs Boosting Iterations - GBM Model
ggplot(train_ccpc_GBM, highlight = TRUE) +
  ggtitle("Core Clock Protein CRY1 Toxicity: Gradient Boosting Machines(GBM)")
```

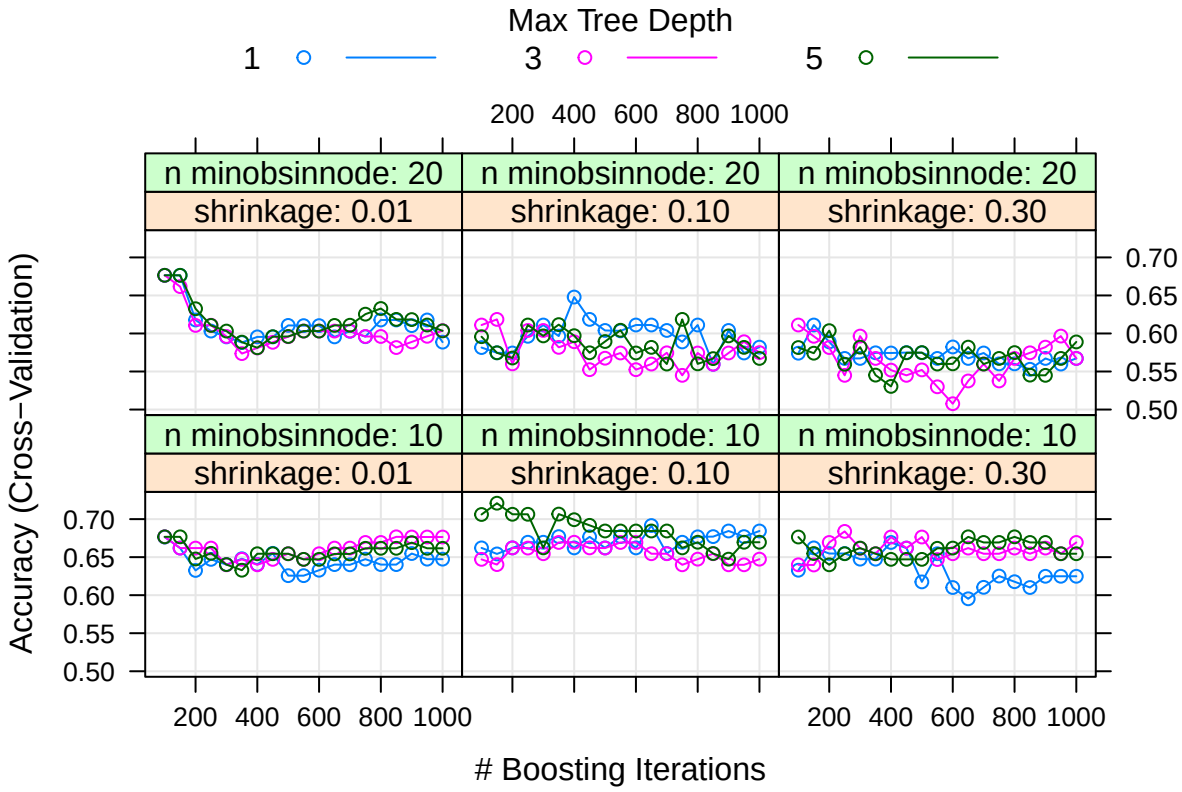



```
# Display the Best GBM Model
train_ccpc_GBM$bestTune
```

```
##     n.trees interaction.depth shrinkage n.minobsinnode
## 192     150                 5       0.1             10
```

R Codes that plot Boosting Iterations vs Accuracy of GBM Model

```
# Plot Boosting Iterations vs Accuracy
plot(train_ccpc_GBM)
```



R Codes that Compute F1-Score and Overall Accuracy of GBM Model by confusionMatrix.

```
# Makes Prediction on test set and Compute Overall Accuracy of GBM Model
predict_ccpc_GBM <- predict(train_ccpc_GBM, test_set_x)
accuracy_ccpc_GBM <- confusionMatrix(table(predict_ccpc_GBM, test_set_y),mode="everything")

accuracy_ccpc_GBM$byClass["F1"]
```

```
##     F1
## 0.8163
```

```
accuracy_ccpc_GBM$overall["Accuracy"]
```

```
## Accuracy
##   0.7429
```

4.1.5 Extreme Gradient Boosting Model (XGBoost)

XGBoost builds a series of **Molecular Descriptors** Trees, each learning from the errors of the previous ones, and combine them to make Accurate **predictions.** The process begins with a simple prediction, usually the average value of the target variable. Sequentially builds **Molecular Descriptors** Tree to correct errors made by the previous ones. Each new **Tree** focus on the errors from the previous Model. Uses gradient descent to minimize the loss function, which measures the difference between actual and predicted values. Each **Molecular Descriptors** Tree's **prediction** is given a weight, and the combined result is more **Accurate.**

Hyper-Parameters Optimization:

```
A) nrounds=100: Number of boosting iterations (Trees)

    * Set nrounds=100 can Improve Performance because Model has more opportunities to learn from errors

B) max_depth=8: Maximum depth of each Tree.

   * Set max_depth=8 because Deeper Trees capture complex patterns.

C) eta=c(0.01, 0.1, 0.3):  Controls the contribution of each tree. It is also called "learning rate".

   * Set eta=c(0.01, 0.1, 0.3) because smaller values ensure the Model learns more slowly and steadily
     for better generalization and precise.

D) gamma=c(0, 0.2, 0.4): Minimum loss reduction required to Split a node.

   * Set gamma=c(0, 0.2, 0.4) because Higher values make fewer Splits and makes the Model more
     conservative which can prevent over-fitting.

E) colsample_bytree=c(0.5, 0.7, 1): This represents Fraction (e.g., 100%) of Features/Molecular
                                    Descriptors used to training each Tree.

   * Set colsample_bytree=c(0.5, 0.7, 1) Improve Model diversity and prevents Over-fitting by reducing
                          Molecular Descriptors/Features Correlation.

F) min_child_weight=c(1, 3, 5): set minimum sum of instance weight needed in a child.

   * Set min_child_weight=c(1, 3, 5) ensures that any Split happens only if there are at least(e.g., 1)
     samples in the child nodes to Avoid Over-fitting.

G) subsample=c(0.5, 0.7, 1): This represents Fraction (e.g., 70%) of samples used to training each Tree

   * set subsample=c(0.5, 0.7, 1) because it helps Balance between bias and variance for Improving
                 generalization and prevents Over-fitting by introducing Randomness.
```

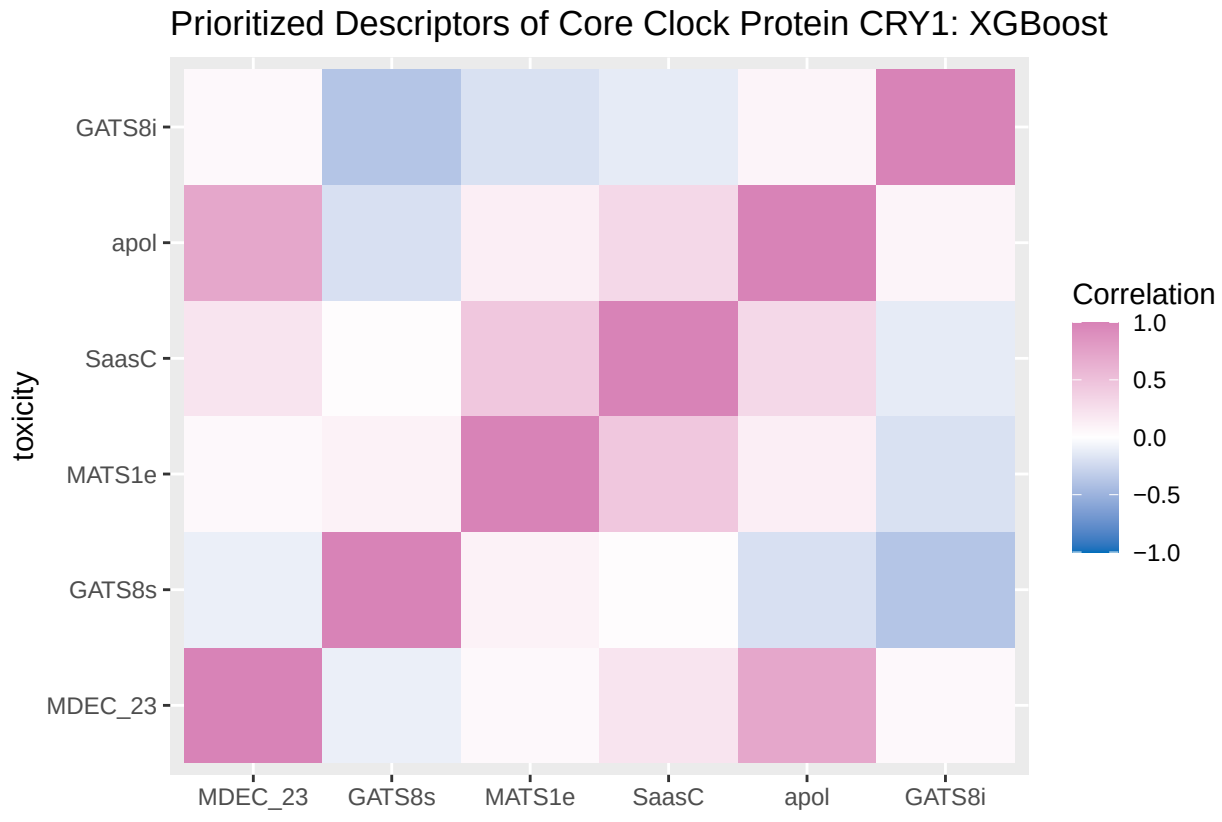
Optimize its **Hyper-Parameters** to give the **Model** a better understanding of the underlying **Molecular Descriptors** data patterns while being cautious about Over-fitting. In addition, applying **5-Fold Cross-Validation** can help identify the Optimal combination of Parameters and Achieve more Robust Performance in a shorter time.

XGBoost Model achieved F1-Score of 0.809 & Overall Accuracy of 0.743 by confusionMatrix.

The Best XGBoost Model as follow: nrounds=100, max_depth=8, eta=0.1, gamma=0, colsample_bytree=0.5, min_child_weight=1, subsample=0.7

R Codes that utilize Prioritized Molecular Descriptors for XGBoost Model and Heat Map

```
#########################################################################################################
#  Apply Prioritized Molecular Descriptors on Extreme Gradient Boosting Model (XGBoost) Training  #
#########################################################################################################

# Generate df_ccpc_data dataset comprising Prioritized Molecular Descriptors of XGBoost Model
df_ccpc_data <- ccpc_data  %>% select(MDEC_23, GATS8s, MATS1e, SaasC, apol, GATS8i, -Class)

# Compute Correlation coefficient of df_ccpc_data
cor_ccpc_data <- cor(na.omit(df_ccpc_data[, unlist(lapply(df_ccpc_data, is.numeric))]))

#########################################################################################################
# Heat Map -  Prioritized Molecular Descriptors Correlation coefficient of XGBoost Model #
#########################################################################################################
df_cor_ccpc_data <- melt(cor_ccpc_data)
colnames(df_cor_ccpc_data) <- c("molecular_descriptors","toxicity","value")

ggplot(df_cor_ccpc_data , aes(x=molecular_descriptors,y=toxicity, fill=value)) +
  geom_tile() +
  scale_fill_gradient2(low = "#006EBB", high = "#D883B7" , mid = "white",
                       midpoint = 0, limit = c(-1, 1), space = "Lab",
                       name="Correlation") +
  labs(x="", title="Prioritized Descriptors of Core Clock Protein CRY1: XGBoost")
```



R Codes create core_clock_protein_cry Matrix and Toxicity Vector for XGBoost Model

```
# Generate molecular_descriptors Matrix and toxicity Vector
molecular_descriptors <- df_ccpc_data %>% as.matrix()
toxicity <- factor(ccpc_data$Class)

# Generate core_clock_protein_cry for Normalization
core_clock_protein_cry <- list(molecular_descriptors=molecular_descriptors,
                               toxicity=toxicity )
```

R Codes Scaling and Normalizing core_clock_protein_cry for XGBoost Model

```
##############################################################################
#  Scaling and Normalizing data for Extreme Gradient Boosting Model (XGBoost)  #
##############################################################################

# Compute Mean of each Column
col_means <- colMeans(core_clock_protein_cry$molecular_descriptors, na.rm=TRUE)

# Compute Standard Deviations of each Column
col_sds <- colSds(core_clock_protein_cry$molecular_descriptors, na.rm=TRUE)

# Subtract Column Means
molecular_descriptors_centered <- sweep(core_clock_protein_cry$molecular_descriptors,
                                        2, col_means, FUN="-")

# Divide by Column Standard Deviations
molecular_descriptors_scaled <- sweep(molecular_descriptors_centered,
                                      2 , col_sds, FUN="/")
```

R Codes that List 6 Rows of Normalized molecular_descriptors_scaled of XGBoost Model

```
# Display first 6 rows of molecular_descriptors_scaled
head(molecular_descriptors_scaled)
```

```
##      MDEC_23  GATS8s  MATS1e   SaasC   apol  GATS8i
## [1,]  2.0861 -0.1490 -0.5318 -1.7377 0.5888  0.3959
## [2,]  0.5322  1.1421  0.3064  0.1987 0.4472  0.7950
## [3,] -0.1573 -0.7000 -1.6525 -0.5686 0.5684 -0.5977
## [4,]  0.1445 -0.2046 -1.8810 -0.7597 0.7520 -0.5761
## [5,]  1.3483 -0.7804  2.6592 -0.1125 0.4337 -0.1986
## [6,]  1.3133 -0.4970 -0.6696 -0.7899 0.5788  0.1333
```

R Codes create core_clock_protein_cry 80% training set and 20% test set of XGBoost Model

```
#########################################################################################################
#  Generate 80% training set and 20% test set for  Extreme Gradient Boosting Model (XGBoost)  #
#########################################################################################################

set.seed(1)

test_index <- createDataPartition(core_clock_protein_cry$toxicity, times = 1, p = 0.2, list = FALSE)

test_set_x   <-  molecular_descriptors_scaled[test_index,]
test_set_y   <-  core_clock_protein_cry$toxicity[test_index]

train_set_x  <- molecular_descriptors_scaled[-test_index,]
train_set_y  <- core_clock_protein_cry$toxicity[-test_index]

#################################################################
#                                                               #
#     Extreme Gradient Boosting Model (XGBoost) Training        #
#                                                               #
#################################################################
```

A) Hyper-Parameter Optimization of Extreme Gradient Boosting Model (XGBoost)

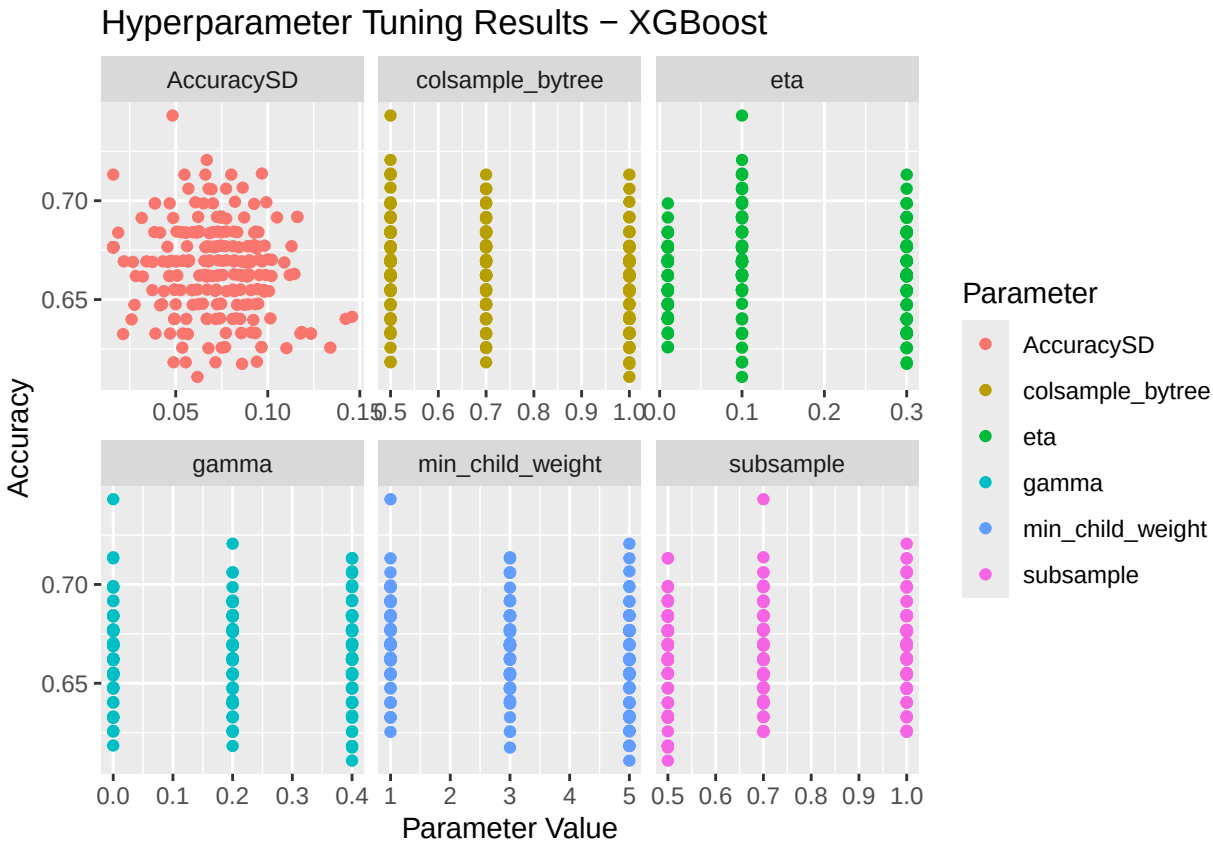
1) Fine-tune the Models to improve the performance, involving Hyper-Parameter tuning and Cross-Validation.
2) Optimize the Models by setting parameters as following:
   - Set Number of boosting iterations ("nrounds") equal to 100
   - Set Maximum depth of each tree ("max_depth") equal to 8
   - Set Learning rate ("eta") equal to 0.01, 0.1 and 0.3
   - Set "gamma" equal to 0, 0.2 and 0.4 (Minimum loss reduction required to split a node)
   - Set "colsample_bytree" equal to 0.5, 0.7 and 1 (Fraction of Molecular Descriptors/Features used for training each tree)
   - Set "min_child_weight" equal to 1, 3, and 5 (Minimum sum of instance weight needed in a child)
   - Set "subsample" equal to 0.5, 0.7 and 1 (Fraction of samples used for training each Molecular Descriptors tree)
   - Set Number of "Cross validation" equal to 5

R Codes that Optimize Hyper-Parameters of XGBoost Model for training

```
##############################################################################################
# Optimize Parameters of Extreme Gradient Boosting (XGBoost) Model                           #
#                                                                                            #
#       * Set Number of boosting iterations ("nrounds") equal to 100                         #
#       * Set Maximum depth of each tree ("max_depth") equal to 8                            #
#       * Set Learning rate ("eta") equal to 0.01, 0.1 and 0.3                               #
#       * Set "gamma" equal to 0, 0.2 and 0.4                                                #
#         (Minimum loss reduction required to split a node)                                  #
#       * Set "colsample_bytree" equal to 0.5, 0.7 and 1                                     #
#         (Fraction of Molecular Descriptors/Features used for training each tree)           #
#       * Set "min_child_weight" equal to 1, 3, and 5                                        #
#         (Minimum sum of instance weight needed in a child )                                #
#       * Set "subsample" equal to 0.5, 0.7 and 1                                            #
#         (Fraction of samples used for training each Molecular Descriptors tree)            #
#       * Set Number of "Cross validation" equal to 5                                        #
##############################################################################################
set.seed(123)

control <- trainControl(method="cv",number=5)

grid <- expand.grid( nrounds=100,
                     max_depth=8,
                     eta=c(0.01, 0.1, 0.3),
                     gamma=c(0, 0.2, 0.4),
                     colsample_bytree=c(0.5, 0.7, 1),
                     min_child_weight=c(1, 3, 5),
                     subsample=c(0.5, 0.7, 1))

train_ccpc_XGB <- train(train_set_x,train_set_y,
                        method="xgbTree",
                        trControl=control,
                        tuneGrid=grid)
```

R Codes that convert the results to long format and ggplot Hyper-Parameters of XGBoost Model

```
# results contain Hyper-Parameters of XGBoost Model
results <- train_ccpc_XGB$results

# Convert the results to a long format for ggplot -XGBoost
results_long <- results %>% pivot_longer(cols = -c(Accuracy, Kappa, KappaSD, nrounds, max_depth),
                                         names_to = "Parameter",
                                         values_to = "Value")

# ggplot Hyper-Parameters of XGBoost Model
ggplot(results_long, aes(x = Value, y = Accuracy, color = Parameter)) +
  geom_point() +
  facet_wrap(~ Parameter, scales = "free_x") +
  labs(title = "Hyperparameter Tuning Results - XGBoost", x = "Parameter Value", y = "Accuracy")
```

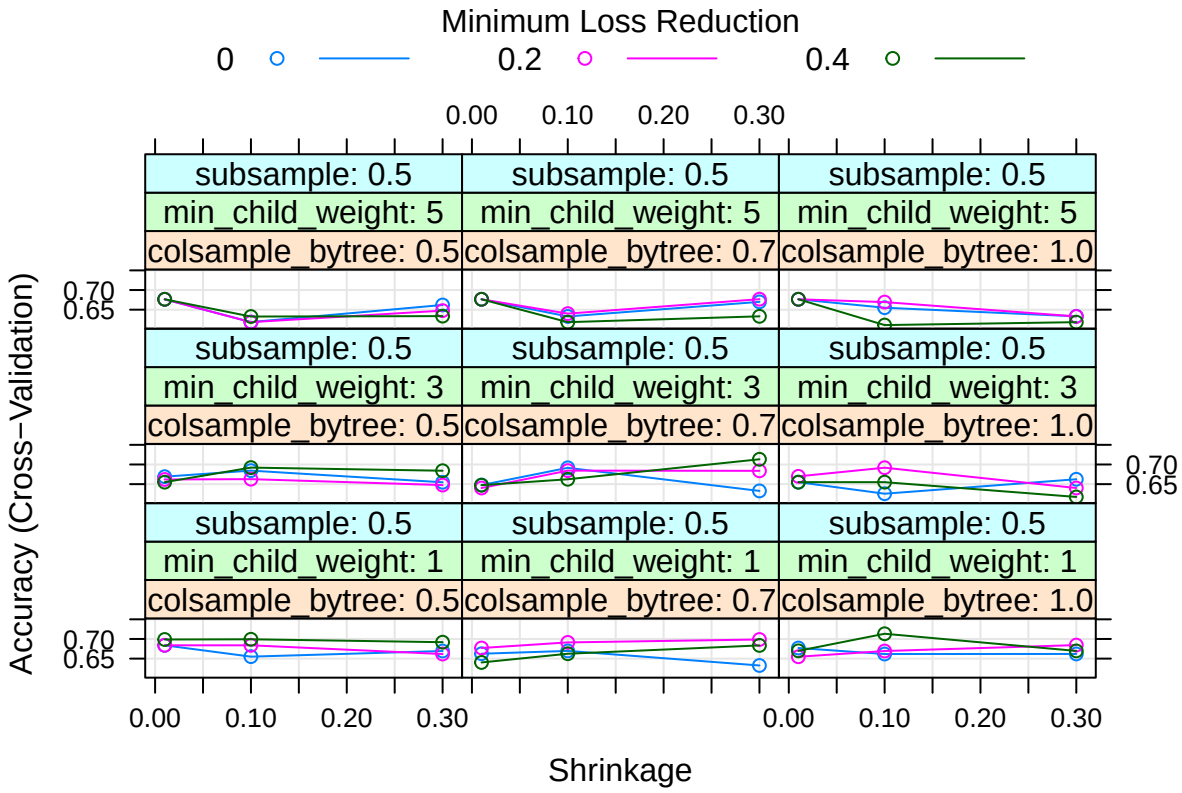


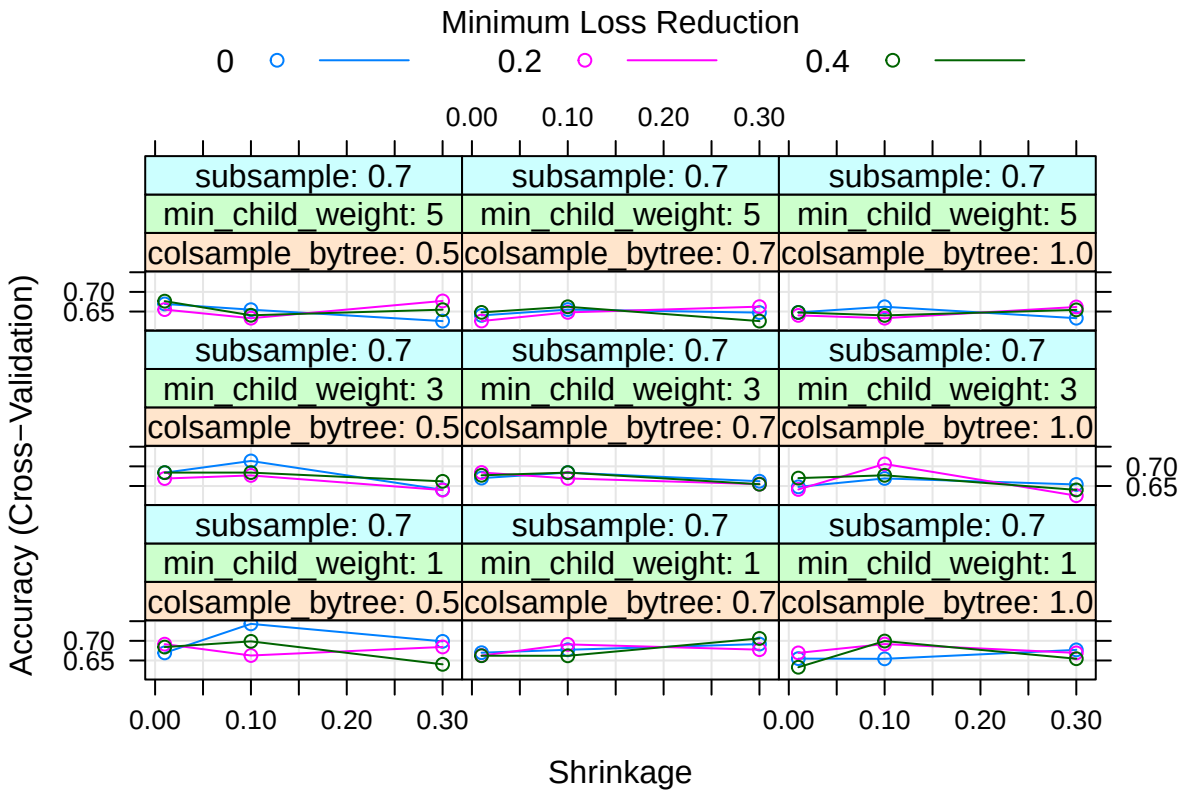

```
# Display the Best Model of XGBoost
train_ccpc_XGB$bestTune
```

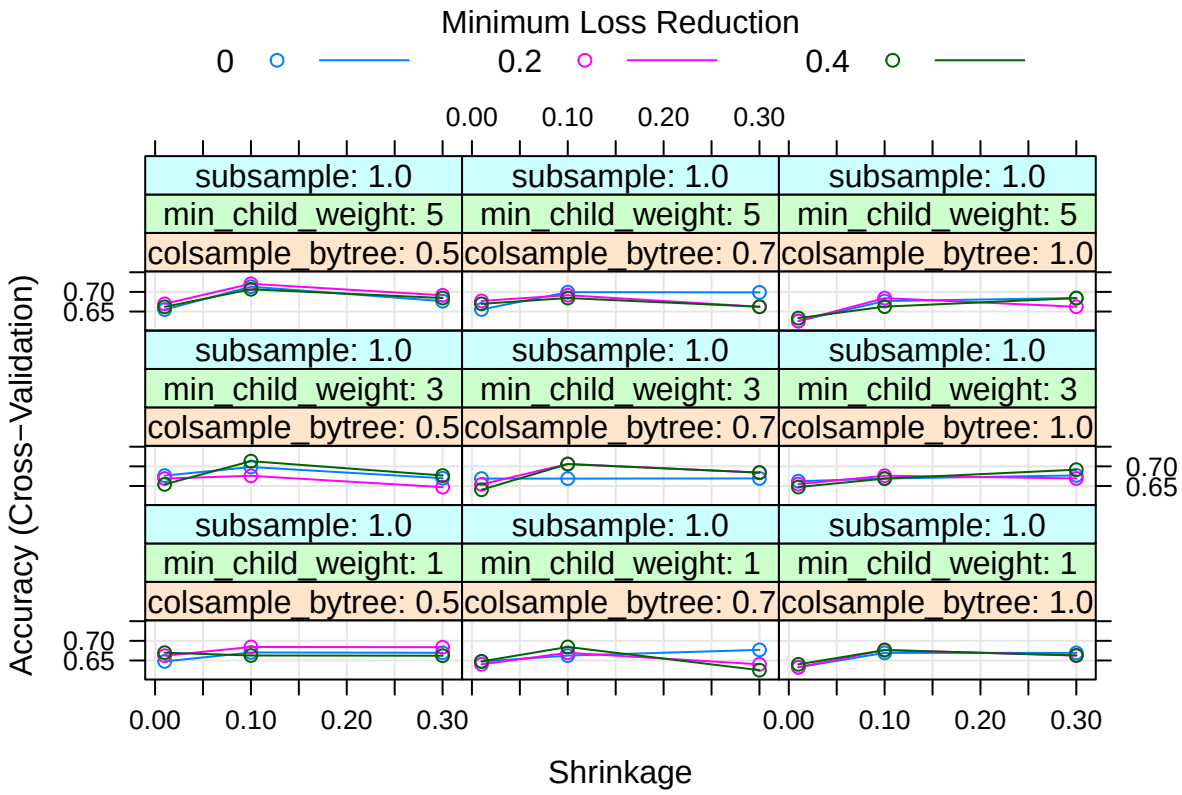
```
##    nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 83     100         8 0.1     0              0.5                1       0.7
```

R Codes plot Hyper-Parameters and Compute F1-Score and Overall Accuracy by confusion-Matrix (For XGBoost Model)

```
# plot Hyper-Parameters - XGBoost
plot(train_ccpc_XGB)
```




Minimum Loss Reduction
0
0.2
0.4
Accuracy (Cross−Validation)
Shrinkage
subsample: 0.7
min_child_weight: 5
colsample_bytree: 0.5
subsample: 0.7
min_child_weight: 5
colsample_bytree: 0.7
subsample: 0.7
min_child_weight: 5
colsample_bytree: 1.0
subsample: 0.7
min_child_weight: 3
colsample_bytree: 0.5
subsample: 0.7
min_child_weight: 3
colsample_bytree: 0.7
subsample: 0.7
min_child_weight: 3
colsample_bytree: 1.0
subsample: 0.7
min_child_weight: 1
colsample_bytree: 0.5
subsample: 0.7
min_child_weight: 1
colsample_bytree: 0.7
subsample: 0.7
min_child_weight: 1
colsample_bytree: 1.0
0.00
0.10
0.20
0.30
0.65
0.70




```
# Makes Prediction on test set and Compute Overall Accuracy of XGBoost Model
predict_ccpc_XGB <- predict(train_ccpc_XGB, test_set_x)

accuracy_ccpc_XGB <- confusionMatrix(table(predict_ccpc_XGB, test_set_y),mode="everything")

accuracy_ccpc_XGB$byClass["F1"]
```

```
##     F1
## 0.8085
```

```
accuracy_ccpc_XGB$overall["Accuracy"]
```

```
## Accuracy
##   0.7429
```

4.1.6 Ensemble Model

Implementing an Ensemble Model of **Core Clock Protein Cryptochrome (CRY1) Toxicity Prediction** using multiple Machine Learning Algorithms of KNN, GBM, Random Forest and XGBoost to Improve Overall Performance. Ensemble Models can **Perform Better** than each individual **Model** because they aggregate diverse perspectives. By combining Models, the Ensemble can Reduce Errors and Improve the Accuracy. It also helps mitigate the risk of **Over-fitting** because if one Model over-fits the **Molecular Descriptors** data, the other Models can help **Balance** this out, leading to a more generalized Model.

My idea is to pool the **Strengths** of each Model to create a more **Robust** and **Accurate** Prediction System. One Approach is to create an Ensemble using the Toxicity Predictions from all of the Models (KNN, GBM, Random Forest and XGBoost). Use this Ensemble to generate a **Majority Prediction of Molecules/Instances Toxicity Class**. The Ensemble Model potentially boost Accuracy by leveraging the Strengths of each Model. This Approach push Accuracy above any single Model, especially if the Models complement each other well.

An **Ensemble** Approach can potentially yield High **Accuracy.** Each Model brings its own Strengths:

A) Random Forest:

* Random Forest builds multiple Decision Trees (Molecular Descriptors Trees) and averages their Predictions, reducing the risk of Over-fitting. It can also capture non-linear relationships of Molecular Descriptors to Improve the Ensemble's Robustness by providing diverse Predictions and reducing variance.

B) Gradient Boosted Machine (GBM):

* GBM builds Models sequentially to correct the errors of previous Models, making it powerful for Classification tasks. It handles unbalanced dataset effectively and capture complex relationships of Molecular Descriptors. It also Improve the Ensemble's Accuracy by focusing on correcting errors and boosting weak learners. (Molecular Descriptors Trees).

C) K-Nearest Neighbors (KNN):

* KNN is simple and effective for Classification tasks. It works well with our small dataset and can capture complex decision boundaries of Molecules Toxicity Class. It can also help Improve the Ensemble's Performance where local similarity of Molecular Descriptors is important.

D) Extreme Gradient Boosting (XGBoost):

* XGBoost is an Optimized version of GBM that is more Efficient. It can also handle imbalanced data well and capturing intricate patterns of Molecular Descriptors to Enhance the Ensemble's Performance.

Ensemble Models combine the Strengths of Four Models (**KNN, GBM, Random Forest and XBoost**) to reduce both variance and bias, yielding a more **Balanced** and Effective Prediction System. Combining their Predictions can lead to a well-rounded, **Accurate** Model, Leveraging the **Strengths** of each **Model.**

Ensemble Model Achieved F1-Score of 0.880 and an Overall Accuracy of 0.8286, as measured by confusionMatrix function of Caret Package in R.

R Codes that generate an Ensemble Model of Core Clock Protein Cryptochrome CRY1 Toxicity Prediction

```
#############################################################################################
#                                                                                           #
#   Generate an Ensemble Model of Core Clock Protein Cryptochrome (CRY1) Toxicity Prediction  #
#                                                                                           #
#############################################################################################
set.seed(1)

# Create Dataframe contains predictions of GBM, KNN, Random Forest and XGBoost Model
combined_predict_ccpc <- data.frame( predict_ccpc_GBM, predict_ccpc_knn, predict_ccpc_rf,
                                     predict_ccpc_XGB)


# Generate a Majority Prediction of Toxicity Class using "combined_predict_ccpc"- My Version
ensemble_ccpc <- apply(combined_predict_ccpc, 1, function(row)
{ ifelse(mean(row =="Toxic") > 0.5, "Toxic","NonToxic") })


# Compute F1-Score and Overall Accuracy of an Ensemble Model by confusionMatrix
accuracy_ccpc_ensemble <- confusionMatrix(factor(ensemble_ccpc), test_set_y, mode="everything")
accuracy_ccpc_ensemble
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction NonToxic Toxic
##   NonToxic       22     5
##   Toxic           1     7
##
##                Accuracy : 0.829
##                  95% CI : (0.664, 0.934)
##     No Information Rate : 0.657
##     P-Value [Acc > NIR] : 0.0209
##
##                   Kappa : 0.587
##
##  Mcnemar's Test P-Value : 0.2207
##
##             Sensitivity : 0.957
##             Specificity : 0.583
##          Pos Pred Value : 0.815
##          Neg Pred Value : 0.875
##               Precision : 0.815
##                  Recall : 0.957
##                      F1 : 0.880
##              Prevalence : 0.657
##          Detection Rate : 0.629
##    Detection Prevalence : 0.771
##       Balanced Accuracy : 0.770
##
##        'Positive' Class : NonToxic
##
```

```r
accuracy_ccpc_ensemble$byClass["F1"]
```

```
##   F1
## 0.88
```

```r
accuracy_ccpc_ensemble$overall["Accuracy"]
```

```
## Accuracy
##   0.8286
```

R Codes that generate Table for Summarizing Accuracy of all Models

```
# Create a table of Accuracy encompassing 4 Models + Ensemble Model (My version- Majority Prediction)
accuracies_table_ccpc <- data.frame(Model    = c( "K-Nearest Neighbors(KNN)",
                                                  "Gradient Boosting Machines(GBM)",
                                                  "Extreme Gradient Boost(XGBoost)",
                                                  "Random Forest",
                                                  "Ensemble"),
                                    Accuracy = c( accuracy_ccpc_knn$overall["Accuracy"],
                                                  accuracy_ccpc_GBM$overall["Accuracy"],
                                                  accuracy_ccpc_XGB$overall["Accuracy"],
                                                  accuracy_ccpc_rf$overall["Accuracy"],
                                                  accuracy_ccpc_ensemble$overall["Accuracy"]))


# Summarize Accuracy of all Models using knitr function
knitr::kable(accuracies_table_ccpc, caption="Accuracy of all Model")
```

Table 12: Accuracy of all Model

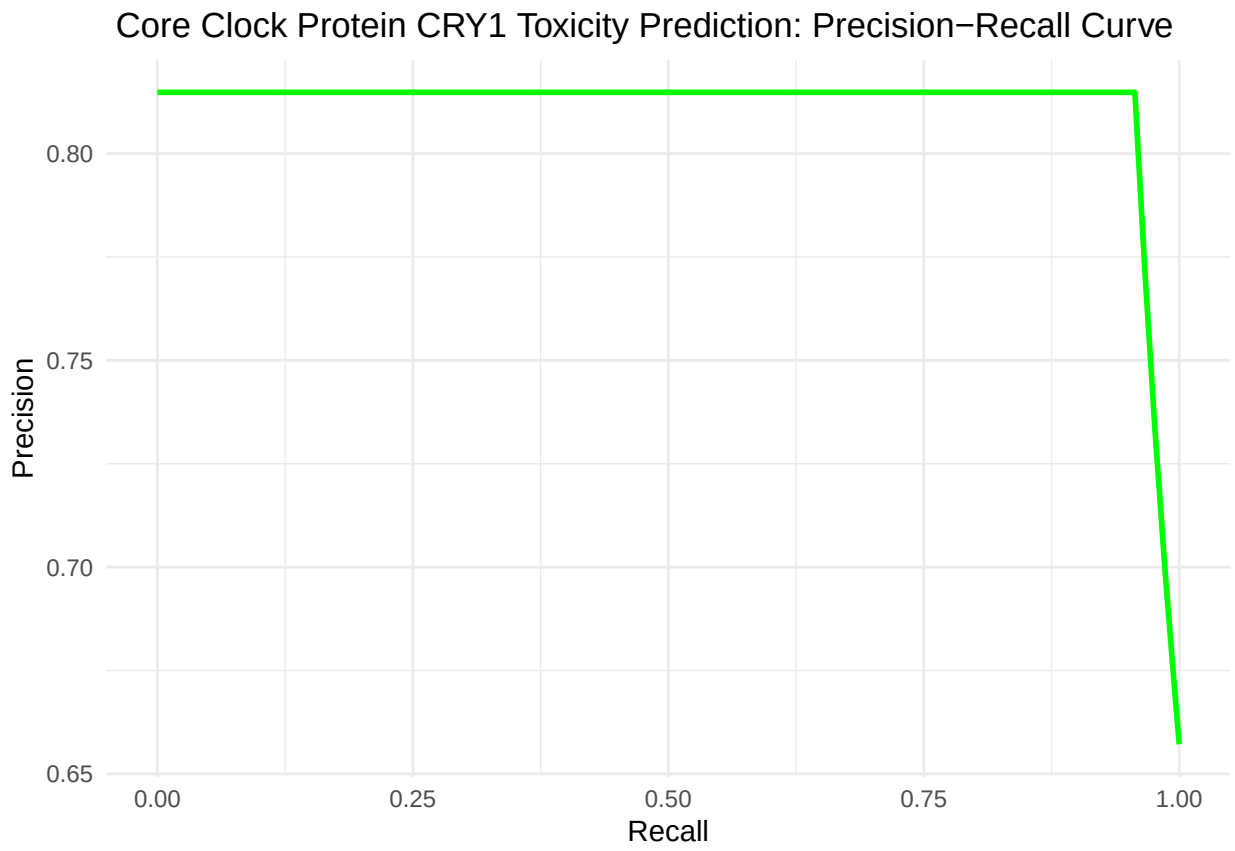
Model	Accuracy
K-Nearest Neighbors(KNN)	0.8000
Gradient Boosting Machines(GBM)	0.7429
Extreme Gradient Boost(XGBoost)	0.7429
Random Forest	0.7714
Ensemble	0.8286

```
# Best Model
best_ensemble_ccpc <- accuracies_table_ccpc$Model[which.max(accuracies_table_ccpc$Accuracy)]
best_ensemble_ccpc
```

```
## [1] "Ensemble"
```

Plot Precision-Recall Curve of Ensemble Model

```
#####################################################
#   Plot Precision-Recall Curve of Ensemble Model   #
#####################################################

# Convert factors to binary (0, 1) for ROC analysis
binary_labels <- ifelse(test_set_y == "NonToxic", 1, 0)
binary_preds <- ifelse(ensemble_ccpc == "NonToxic", 1, 0)

# Create data for precision-recall plot
pr_data <- pr.curve(scores.class0 = binary_preds,
                    weights.class0 = binary_labels == 1,
                    curve = TRUE)

# Extract precision-recall data
pr_df <- data.frame(Recall = pr_data$curve[, 1], Precision = pr_data$curve[, 2])

# Plot Precision-Recall curve
pr_plot <- ggplot(pr_df, aes(x = Recall, y = Precision)) +
  geom_line(color = "green", size = 1) +
  ggtitle(" Core Clock Protein CRY1 Toxicity Prediction: Precision-Recall Curve") +
  xlab("Recall") +
  ylab("Precision") +
  theme_minimal()
print(pr_plot)
```



4.2 Impact of Algorithms on Ensemble Performance

My Design of **Ensemble Model** is based on both Distance Based Learning Algorithm and Decision Tree Based Algorithms that Leverage the capabilities of Core Clock Protein Cryptochrome (CRY1) Toxicity Prediction System. Both Algorithms also have significant impact on **Performance** of our Ensemble Model.

A) Impact of Distance Based Learning Algorithms (KNN) on Performance

* KNN being a non-parametric method. It doesn't make any assumptions about the underlying Molecular Descriptors data distribution. This flexibility allows it to be applied to a variety of Molecules datasets without the need to adjust for different Molecular Descriptors data distributions.

* In an Ensemble, KNN can provide a different perspective compared to more complex Models. Its reliance on local Molecular Descriptors data points can complement the global patterns captured by Tree-Based Methods, thus Improving Overall Model Performance.

B) Impact of Decision Tree Based Algorithms (GBM, Random Forest, XGBoost) on Performance

* Decision Trees can capture non-linear relationships between Molecular Descriptors/Features and the target variable, which can be crucial for complex Molecules/Instances datasets. Decision Trees can handle Molecular Descriptors outliers Effectively, making them Robust in a variety of scenarios.

* Random Forest and GBM add strength to an Ensemble through their ability to Model complex Molecular Descriptors relationships. They reduce Over-fitting by combining multiple Molecular Descriptors Trees, and when used in an Ensemble, they can significantly Improve predictive Accuracy and Robustness.

* GBM and XGBoost can dramatically enhance the Performance of an Ensemble Model due to their ability to sequentially correct errors of previous Models. This iterative refinement allows the Ensemble to converge to a very Accurate and generalizable Model.

By combining these diverse **Algorithms** into an Ensemble, you **Leverage** their complementary Strengths:

- KNN: Adds a local and Distance Based perspective.
- Random Forest: Brings in Robust, non-linear Modeling and Reduces Over-fitting.
- GBM/XGBoost: Provides iterative Error Correction.

Together, these Algorithms can create a **Balanced** and **Robust** Ensemble Model that excels in various scenarios, ensuring that you capture a wide range of patterns and relationships in the **Molecular Descriptors** data.

5 Result

5.1 Metrics Evaluation

By carefully selecting the appropriate Evaluation Metrics, you can better understand Model's **Strengths** and Weaknesses and make more informed decisions about its **Performance.** Others than Accuracy, I also use Metrics such as F1-Score, Recall, Precision, Sensitivity and Specificity to Evaluate Model Performance on **Molecular Descriptors** dataset. Choosing the right Evaluation Metric is crucial to Accurately assess its Performance, especially when dealing with imbalanced dataset.

A) Metrics

- Accuracy represents the Overall correctness of the model. The Proportion of true results among the total number of cases.
- F1-Score is the harmonic mean of Precision and Recall. Useful when you need a Balance between Precision and Recall. It is particularly helpful for imbalanced datasets.
- Recall measures the Proportion of Actual Positives that are correctly identified. It also known as Sensitivity.
- Precision measures the Proportion of Positive Identifications that were actually Correct.
- Sensitivity is another term for Recall.
- Specificity measures the Proportion of Actual Negatives that are correctly identified (True Negatives).
- Confusion Matrix is a table that Summarizes the Performance of a Classification Model by showing the True Positive, False Positives, True Negatives and False Negatives. Provides a Comprehensive view of the Model's Performance and helps in calculating other Metrics like Precision , Recall and F1-score.

5.2 Performance

A) performance Evaluation

- Positive Class is "NonToxic", means Metrics are focused on the Correct identification of "NonToxic" content.
- Overall Accuracy of 82.86% seems to Perform well in identifying "NonToxic" content and suggests our Model is quite Reliable.
- Precision of 81.5% of the Molecules predicted as "NonToxic" by the Model are indeed "NonToxic". While this is a Quite Good, there is still some room for improvement.
- High Recall/Sensitivity of 95.7% means that 95.7% of the Actual "NonToxic" Molecules are Correctly identified by the Model. The Model is very Effective at identifying "NonToxic" Molecules.
- Lower specificity leading to some False Negatives.
- Strong F1-Score of 88% indicate Model's Overall Performance, showing Good Balance between Precision and Recall.
- P-value of 0.0209 suggests that the Model's Accuracy is Significantly Better, with a High Level of Statistical Confidence.

Ensemble Model's **Performance** is Quite Impressive indicates our Model has Achieved Significant Success in Molecules Toxicity Prediction. It also **Outperformed** other 4 models. High Recall of 95.7% suggests the Model Rarely Misses **NonToxic** Molecules. Precision of 81.5 % indicates the Model is Reliable. F1-Score of 88% demonstrates the Model's Overall Effectiveness in Balancing both Precision and Recall. These Results indicate a **Robust** Model with **Excellent Performance.** The Ensemble Model Perform well in identifying **NonToxic** content, with High Recall and a Strong F1-score.

6 Conclusion

An **Ensemble Model** Approach is employed to create an ensemble using Toxicity Predictions for the **Core Clock Protein Cryptochrome (CRY1)** from Models(KNN, GBM, Random Forest, and XGBoost). This ensemble of Models generates a Majority Prediction for the **Toxicity Class of Molecules**, potentially boosting Accuracy by **Leveraging** each Model's Strengths. Handling 1,203 Molecular Descriptors of Molecules may lead to Over-fitting as some Molecular Descriptors might be Redundant. Therefore, additional strategies are adopted to Improve **Classification** Performance and **Avoid Over-fitting.**

One such Approach combines **Principal Component Analysis (PCA)** with Molecular Descriptors Prioritization and Supervised Learning Algorithms to Optimize the Machine Learning Algorithm configuration. Molecular Descriptors Prioritization employs Feature Selection techniques, such as Redundant Feature Elimination and Redundant Features Reducing to identify and Reduce Redundant Molecular Descriptors, thereby reducing noise and Model complexity. The **vars importance** function of Random Forest provides **Insights** into **Feature Selection**, crucial for Molecular Descriptors Prioritization. Eventually, only the Most Relevant Molecular Descriptors are chosen for training and prediction. Model Performance is further Improved through **Hyper-Parameter Optimization** and **Cross-Validation** by analyzing the **interactions** of these **Molecular Descriptors.**

Both Distance-Based Learning and Decision Tree-Based Algorithms Leverage the capabilities of the Ensemble Model, significantly impacting **Performance.** Evaluation Metrics such as F1-score, Accuracy, Precision, and Recall are used to assess the Model's Performance. The Ultimate **Ensemble Model** demonstrates **Robust Performance** in predicting **Molecules Toxicity Class**, as evidenced by the outstanding results of these Evaluation Metrics.

6.1 Limitation

However, after incorporating **Molecular Descriptors Prioritization** Method , selection of the Most Relevant Molecular Descriptors is needed for each Model individually. A single set of Descriptors cannot be used Universally. Handling datasets with Very High Dimensions and **Complexity** often necessitates generating a large Number of Trees during **Feature Selection** process which can be both time-consuming and memory-intensive .

The inclusion of a Substantial volume of Molecular Descriptors by the growing interest and development in Molecules Toxicity Prediction Model will result in exponential growth in dimesionallity. Higher dimensionality presents **significant challenges** and complexity will also rises in the future.

6.2 Ensemble Model of Core Clock Protein CRY1 Toxicity Prediction

F1-Score of Ensemble Model(0.880) Outperforms K-Nearest Neighbors Model(0.851), as well as Random Forest Model(0.84), Gradient Boosted Machine Model(0.816) and Extreme Gradient Boosting Model(0.809).

Accuracy of Ensemble Model(0.823) Outperforms K-Nearest Neighbors Model(0.800),as well as Random Forest Model(0.771), Gradient Boosted Machine Model(0.743) and Extreme Gradient Boosting Model(0.743).

Ensemble Model of Core Clock Protein Cryptochrome (CRY1) Toxicity Prediction is the **Optimal Model**, achieving an Highest F1-Score(0.880) and Accuracy(0.823). I am Optimistic about the **Machine Learning** Ensemble Model that we have Selected.

The Performance of Final Ensemble Model is as follow:

F1-Score=0.880, Accuracy=0.823, Recall=0.957, Precision=0.815, Sensitivity=0.957, Specificity = 0.583, P-value=0.021

The Best Model is Ensemble Model, which achieved an Highest F1-Score and Accuracy. I am satisfied with the results, the Reliability and Trustworthiness of the Model are validated by F1-Score and Accuracy Metric Evaluation.

The Final Machine Learning Ensemble Model achieved Highest F1-Score(0.880) and Accuracy(0.823) which indicate a Robust Model with Excellent Performance. As a Result, I am Confident in adopting the ultimate Ensemble Model and Algorithm to build our Core Clock Protein Cryptochrome (CRY1) Toxicity Prediction System.

6.3 Future Work

To further **Enhance** the Performance of my **Machine Learning** Model, I plan to implement Advanced Machine Learning Algorithms for Feature Selection and Molecular Descriptors Prioritization. The integration of **Deep Learning** and **Neural Networks** are the Key Focus Area.

Incorporating Deep Learning Models can help capture Complex Patterns in the data that traditional Machine Learning Models might miss. Additionally, **Leveraging a GPU** for Computation can Significantly reduce training time, making the process more Efficient. These Advancements will contribute to making the Ensemble model more Robust and Accurate, thereby **Improving its Overall Performance.**

Appendix

References

1. Harvard Professor “Rafael A. Irizarry”.(2019). INTRODUCTION TO DATA SCIENCE.
2. https://rafalab.dfci.harvard.edu/dsbook/
3. https://rafalab.github.io/dsbook/
4. https://topepo.github.io/caret/available-models.html
5. https://topepo.github.io/caret/train-models-by-tag.html
6. UCI Machine Learning Repository Toxicity. https://archive.ics.uci.edu/dataset/728/toxicity-2
7. Wikipedia QSAR. https://en.wikipedia.org/wiki/Quantitative_structure%E2%80%93activity_relationship
8. SEMANTIC SCHOLAR. https://semanticscholar.org
9. Encyclopedia of Bioinformatics and Computational Biology.
10. Virtual Computational Chemistry Laboratory. https://vcclab.org
11. Alvascience. https://alvascience.com
12. GeneCards. https://genecards.org
13. the Company of Biologists. journals.biologists.com
14. OCHEM. docs.ochem.eu
15. Comprehensive R Archive Network. https://cran.r-project.org/doc/manuals/r-release/R-lang.html
16. RDocumentation. https://www.rdocumentation.org
17. Microsoft. https://copilot.microsoft.com
18. Kaggle. https://www.kaggle.com
19. UCI Machine Learning Repository. https://archive.ics.uci.edu
20. Stack Overflow. https://stackoverflow.com
21. Statology. https://www.statology.org
22. Steve,L.(2009). Netflix Awards $1Million Prize and Starts a New Contest. Bits. The New York Times.
23. https://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-mew-contest/
24. https://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/
25. https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf