ABSTRACT

Machine learning models can be used to build powerful tools to facilitate decision-making, especially when the outcome is dependent on numerous and varying factors. This paper introduces an R-Shiny based application called 'Project Success Forecast' that can predict the possibility of a project's success in a clinical trial environment, using a random forest machine-learning algorithm. This application serves as a very robust AGILE planning tool to evaluate the project risk on an ongoing basis based on the historical data.

INTRODUCTION

The paper introduces a project success forecast application that allows the user to enter real-time information and predict the rate of success of the project. The rate of success is dependent on the user inputs such as deliverable type, number of resources, number of days, etc. as shown in figure 1 below. The greater the value of success prediction, the higher is the likelihood of project success. This application helps the project team to re-assess the project resources or adjust timelines even before the actual initiation of the project, leading to smoother project execution and success. This paper covers step by step methodology on how this machine-learning application is built using R-Shiny and also provides in detail the usage of the random forest model as a machine learning algorithm.

Figure 1. Project Success Forecast
R-SHINY

Project Success forecast application is built using R-Shiny package.

Shiny is an R package that allows application development and comes as a built-in package available through CRAN that interacts well with RStudio. Shiny applications can be deployed as a webpage or a standalone desktop tool to view interactive data or perform statistical analysis.

A Shiny application typically consists of two components:

- **UI.R**: The function that builds the appearance and assembles the HTML user interface of the app.
- **Server Code**: The function with instructions on how to execute and rebuild the objects displayed in UI. It also contains the algorithms or models that build upon data.

---

**Figure 2. Shiny components**

Input and output functions are used to build the shiny application. Input allows the user to provide values to the app, and output objects are used to render the data, text, plots, and tables after executing the logic when the input controls are modified. Shiny uses a reactive programming model, so the framework can be fast, efficient, and robust. These input and output elements are added as parameters to the `fluidPage()` function in UI.R.

There exist a variety of input functions to create user interface elements that prompt the user for input values or interaction. Some of them are date input, select input, textbox, radio button, checkbox, file upload, slider, etc. The output functions along with the rendering function display the data in different types of output in the application. Some of them are plot, table, HTML, and verbatim text rendered by `renderPlot`, `renderTable`, `renderUI`, and `renderText` functions respectively.

Further information about building reactive web apps using R-shiny can be found in the shiny [cheet sheet](url).
RANDOM FOREST MACHINE LEARNING MODEL

The random forest model in this paper is centered on the idea of 'the wisdom of the crowd'. Random forest builds multiple decision trees and combines them to get accurate and stable predictions compared to single decision tree models. The random forest model used in this application uses training data to train the model, which the users can select with the structure as shown in table 1 below. The random forest model is based on a four-step process as explained below.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ADAM</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>TRUE</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>TFL</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>TRUE</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>SDTM</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>FALSE</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>ADAM</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>TRUE</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>TFL</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>TRUE</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>SDTM</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>FALSE</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>ADAM</td>
<td>6</td>
<td>4</td>
<td>20</td>
<td>TRUE</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>ADAM</td>
<td>5</td>
<td>5</td>
<td>15</td>
<td>TRUE</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>ADAM</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>TRUE</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>ADAM</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>TRUE</td>
<td>Yes</td>
</tr>
<tr>
<td>12</td>
<td>ADAM</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>TRUE</td>
<td>Yes</td>
</tr>
<tr>
<td>13</td>
<td>ADAM</td>
<td>4</td>
<td>7</td>
<td>15</td>
<td>TRUE</td>
<td>No</td>
</tr>
<tr>
<td>14</td>
<td>ADAM</td>
<td>6</td>
<td>5</td>
<td>20</td>
<td>FALSE</td>
<td>No</td>
</tr>
<tr>
<td>15</td>
<td>ADAM</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>FALSE</td>
<td>No</td>
</tr>
<tr>
<td>16</td>
<td>ADAM</td>
<td>3</td>
<td>6</td>
<td>20</td>
<td>TRUE</td>
<td>Yes</td>
</tr>
<tr>
<td>17</td>
<td>ADAM</td>
<td>9</td>
<td>2</td>
<td>30</td>
<td>TRUE</td>
<td>Yes</td>
</tr>
<tr>
<td>18</td>
<td>ADAM</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>FALSE</td>
<td>No</td>
</tr>
<tr>
<td>19</td>
<td>ADAM</td>
<td>7</td>
<td>5</td>
<td>10</td>
<td>TRUE</td>
<td>Yes</td>
</tr>
<tr>
<td>20</td>
<td>ADAM</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>TRUE</td>
<td>No</td>
</tr>
<tr>
<td>21</td>
<td>ADAM</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>TRUE</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 1. Training data structure sample

Step 1: Create a bootstrapped dataset using the training data
Note that the bootstrapped dataset has the same number of records and repetition of records is allowed as shown in the sample bootstrapped data in figure 3 above.

Numerous such bootstrapped datasets are created for the prediction.

**Step 2**: Create a decision tree using the bootstrapped dataset from step 1, but only using a random subset of variables.

In this step, the root node candidates are randomly selected (eg: Project success) and decision trees are created using each of the bootstrapped data as shown in figure 4 below.

![Decision tree using project success as the root node.](image)

**Step 3**: Repeat step 1 and step 2 and create more decision trees and use the new input data to run down on all the trees to identify the cumulative predictive outcome as shown in figure 5 below.

![Multiple decision trees using different samples](image)
Step 4: In this step new sample of data based on which the outcome needs to be predicted is run through the decision trees that have been built using the bootstrapped training data as shown in figure 6 below.

<table>
<thead>
<tr>
<th>Type</th>
<th>Count_of_outputs</th>
<th>count_of_resources</th>
<th>number_of_days</th>
<th>newstudy</th>
<th>project_success</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAM</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>TRUE</td>
<td>?</td>
</tr>
</tbody>
</table>

Figure 6. Sample-based on which the project success needs to be predicted.

Some of the examples of predictions based on different selection criteria can be seen in figure 7 below.

Figure 7. Sample predictions.

RANDOMFOREST PACKAGE

This application uses a random forest package to predict the successful outcome. The random-forest package is invoked using the syntax as shown in figure 8 below.

```r
# Build model
model <- randomForest(project_success ~ ., data = project_success1, ntree = 500, mtry = 5, importance = TRUE)
```

Figure 8. Randomforest package
FUNCTIONALITY

```r
# Import libraries
library(shiny)
library(shinythemes)
library(data.table)
library(curl)
library(randomForest)
library(flexdashboard)

# Read data
#create data objet
project_success1 <- read.csv(file='C:\Users\a043245\Desktop\myrproject\projectsuccess2.csv', stringsAsFactors=TRUE)

# Build model
model <- randomForest(project_success ~ ., data = project_success1, ntree = 500, mtry = 5, importance = TRUE)
```

The user interface part of the code as shown below.

```r
ui <- fluidPage(theme = shinytheme("superhero"),

# Page header
headerPanel('Project Success forecast ')

# Input values
sidebarPanel(
  fileInput("file", "Upload the Training data"), # fileinput() function is used to get the file upload control option
  # helpText("Please select the training data"),
  HTML("<h5>Input parameters</h5>"),

  selectInput("Type", label = "Project Type:",
              choices = list("ADAM" = "ADAM", "TFL" = "TFL", "SDM" = "SDM"),
              selected = "SDM"),

  sliderInput("number_of_days", "Number of full resources:",
              min = 1, max = 100,
              value = 2),

  sliderInput("count_of_outputs", "Number of days:",
              min = 1, max = 200,
              value = 5),

  selectInput("new_study", label = "New Study?",
              choices = list("Yes" = "TRUE", "No" = "FALSE"),
              selected = "TRUE"),

  sliderInput("count_of_outputs", label = "Count of Outputs",
              min = 1, max = 40,
              value = 5),

  #actionButton("submitbutton", "Submit", class = "btn btn-primary")
  submitButton("Submit")
),

mainPanel(
  tags$label(h3("Status/Output")), # Status/Output Text Box
 verbatimTextOutput("contents"),
  tableOutput("tabledata"), # Prediction results table
  # Output: Histogram ----
  gaugeOutput("Scale")
)
)
The server-side of the application is shown below.

```r
# Server
server <- function(input, output, session) {

# Input Data
datasetInput <- reactive(
{
  # Type, count_of_resources, number_of_days, new_study, project_success
  df <- data.frame(
    Name = c("Type", "count_of_resources", "number_of_days", "new_study", "project_success"),
    value = as.character(c(input$Type, input$count_of_resources, input$number_of_days, input$new_study, input$project_success)),
  stringsAsFactors = FALSE)
  data frame

  project_success <- "project_success"
  data(success)  
  input <- transpose(df, quote = FALSE, row.names = FALSE, col.names = FALSE)
  write.table(input, "input.csv", sep="", quote = FALSE, row.names = FALSE, col.names = FALSE)
  test <- read.csv(paste("input", ".csv", sep=""), header = TRUE)
  test$Type <- factor(test$Type, levels = c("TFL", "SDTM", "ADAM"))
  Output <- data.frame(Prediction = predict(model, test), round(predict(model, test, type = "prob"), 3))
  print(Output)
})

my_plot<-reactive(
{
  gauge(datasetInput()$success, min = 0, max = 100, symbol = "/", label = "Success", gaugeSectors(
    success = c(80, 100), warning = c(40, 79), danger = c(0, 39))
})

# Status/Output Text Box
output$Scale <- renderGauge(
{
  my_plot()
})

# Prediction results table
output$tabledata <- renderTable(
{
  datasetInput()
})

# Create the shiny app
shinyApp(ui = ui, server = server)
```
CONCLUSION

Artificial Intelligence (AI) and Machine Learning (ML) are no longer on the horizon. They are here and are already having a profound impact on how we live, work, and do business. AI/ML can and will transform organizational decision-making, drive efficiencies, build new capabilities and business, and power sustainable, value-driving activities.

Machine Learning using software such as R enables us to build complex analytics pipelines to determine the best model for your data. As the AI/ML market continues to evolve and new best practices are established, there are challenges and unique considerations for the successful technology adoption, and the "Project Success Forecast" application discussed in this paper using R-shiny is one step in that direction.

REFERENCES
Reimagining Statistical Reports with R Shiny

ACKNOWLEDGMENTS
Sincere thanks to Steve Benjamin, Director Statistical Programming, Biostatistics and Global Contracts and Aman Bahl, Associate Director, Statistical Programming, Clinical Division for their vision, great leadership, persistent support, and encouragement throughout and for their valuable assistance in reviewing this paper.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:

Hrideep Antony
Principal Statistical Programmer, Clinical Division, Syneos Health
Work Phone: +1- 984 459 4785
Email: hrideep.anthony@syneoshealth.com
Web: http://www.syneoshealth.com

Aman Bahl
Associate Director, Statistical Programming, Clinical Division, Syneos Health
Phone: +1-905 399 6715
E-mail: Aman.Bahl@syneoshealth.com
Web: http://www.syneoshealth.com