Diabetes Progression Manger

By: Alexis Hampton, Bilques Jan, Aysha Mukhtar

Table of contents:

    - Completed Work
    - Incomplete Work
- Planning for Version 2.0 (Phase 2)
- Location for Github Repository

Introduction:

In the United States 11.3% of people have diabetes and many patients have a hard time with their disease. Our system is made to help doctors and nurses maintain their patients' disease. We take the patients data such as their height, weight, sex and sugar levels. With this information we will give them their BMI and predict the progress of the disease. Doctors and nurses have many patients and our system will help them manage each patient's progress with diabetes more efficiently.

Glossary:

Diabetes- Diabetes is a chronic, metabolic disease characterized by elevated levels of blood glucose (or blood sugar), which leads over time to serious damage to the heart, blood vessels, eyes, kidneys and nerves.

Sugar levels- Refers to the concentration of glucose (a type of sugar) in the blood. This measurement is commonly known as blood sugar level or blood glucose level

BMI- Body Mass Index. It is a numerical value of a person's weight in relation to their height. BMI is a widely used screening tool for assessing whether a person has a healthy body weight for a given height.

Random Forest- Random Forest builds multiple decision trees and merges their predictions.

Decision Tree- a flowchart-like structure in which each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label or a numerical value.

Requirements:

User Requirements:

1. The machine learning model will accurately predict whether or not a patient’s diabetes progression is bad or good. This is important because it allows the doctor(user) to give more accurate medical advice to their patients. [nonfunctional and functional]
2. We will train a machine learning model on previous diabetes data that will determine whether or not their diabetes progression is bad or good. This is important because it will provide the technology needed to predict a patient’s disease progression. [functional]
3. If the diabetes prediction is bad, doctors(Users) will get a list of the patient's information that is causing the patient's diabetes to get worse. This will allow the doctors to know what is causing the patient's diabetes to get worse. [functional]

System Requirements:

Functional:

1. Uses user-submitted data to predict a patients disease progression level based on random forest [<150 is good, >=150 is bad]
    a. Input: age, sex, blood sugar levels,average blood pressure, BMI, total serum cholesterol, low-density lipoproteins, high-density lipoproteins, total cholesterol / HDL, possibly log of serum triglycerides level
    b. Output: predicted disease progression level [<150 is good, >=150 is bad]
    c. Source: from our database
    d. Precondition: have the user data and the user requests the disease progression level
    e. Action:

   i. Input the new patient’s data, for each user into the random forest to get the disease progression level.
   f. Post action: releases projected user disease progression to the user using a GUI
   g. Destination: A button on the patient-information GUI
   h. Missing Data: replaces missing data with the median value of the entire dataset
2. Uses in house data to train a bunch of decision trees that will later fill a random forest
   a. Input: age, sex, blood sugar levels,average blood pressure, BMI, total serum cholesterol, low-density lipoproteins, high-density lipoproteins, total cholesterol / HDL, possibly log of serum triglycerides level
   b. Output: a random forest that will predict a patients’ disease progression
   c. Source: from the database
   d. Precondition: have the data
   e. Action:
      i. Will only generate decision trees based on database data not on new patient data.
   f. Post action: have a random forest that can predict disease progression
   g. Destination: run right after the application is started
   h. Missing data: the missing data will be replaced by the median of the missing data point
3. Uses user added data as well as scientific data for healthy levels of blood sugar, blood pressure, BMI, and tells the user if their levels are under or over the recommended level.
   a. Input: age, sex, blood sugar levels,average blood pressure, BMI, total serum cholesterol, low-density lipoproteins, high-density

lipoproteins, total cholesterol / HDL, possibly log of serum triglycerides level, predicted disease progression level

b. Output: A GUI warning screen with indicated data that is under or over the recommended amount and if they are under or over

c. Source: from database

d. Precondition: have user data and disease progression level is bad as predicted by random forest

e. Action:

    i. Checks each new patients (user) data point and compares it to the recommended healthy levels as provided by scientific data online, only if the disease progression level is bad (>=150).

f. Post action: release warning data

g. Destination: run after the data is loaded for a specified patient.

h. Missing data: we shouldn’t have any at this stage, it will likely be filled in already, but if it isn’t, then we would use the median value from the dataset for the datapoint

<u>Non-Functional</u>:

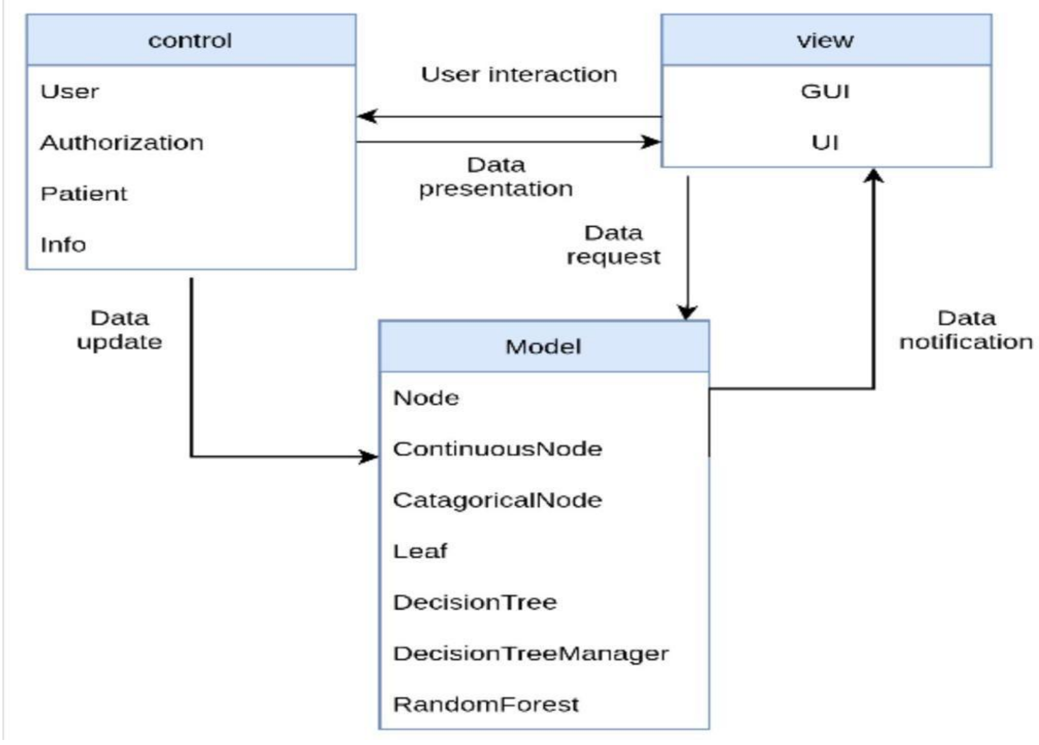
1. User Friendly: We would print to the console, letting the user know that the program is still running. We will let them know when the program is in action, such as when their data is being added to the system.
2. Secure: Ensuring the data does not escape our database. We do this by adding a login(user and password) to make sure no random person can access the data (authorization).
3. Accurate: By ensuring that our trees accuracy rate is at least 90. We can test this when we do validation to see how right the tree is.

Architectural Design

Model-View-Controller:

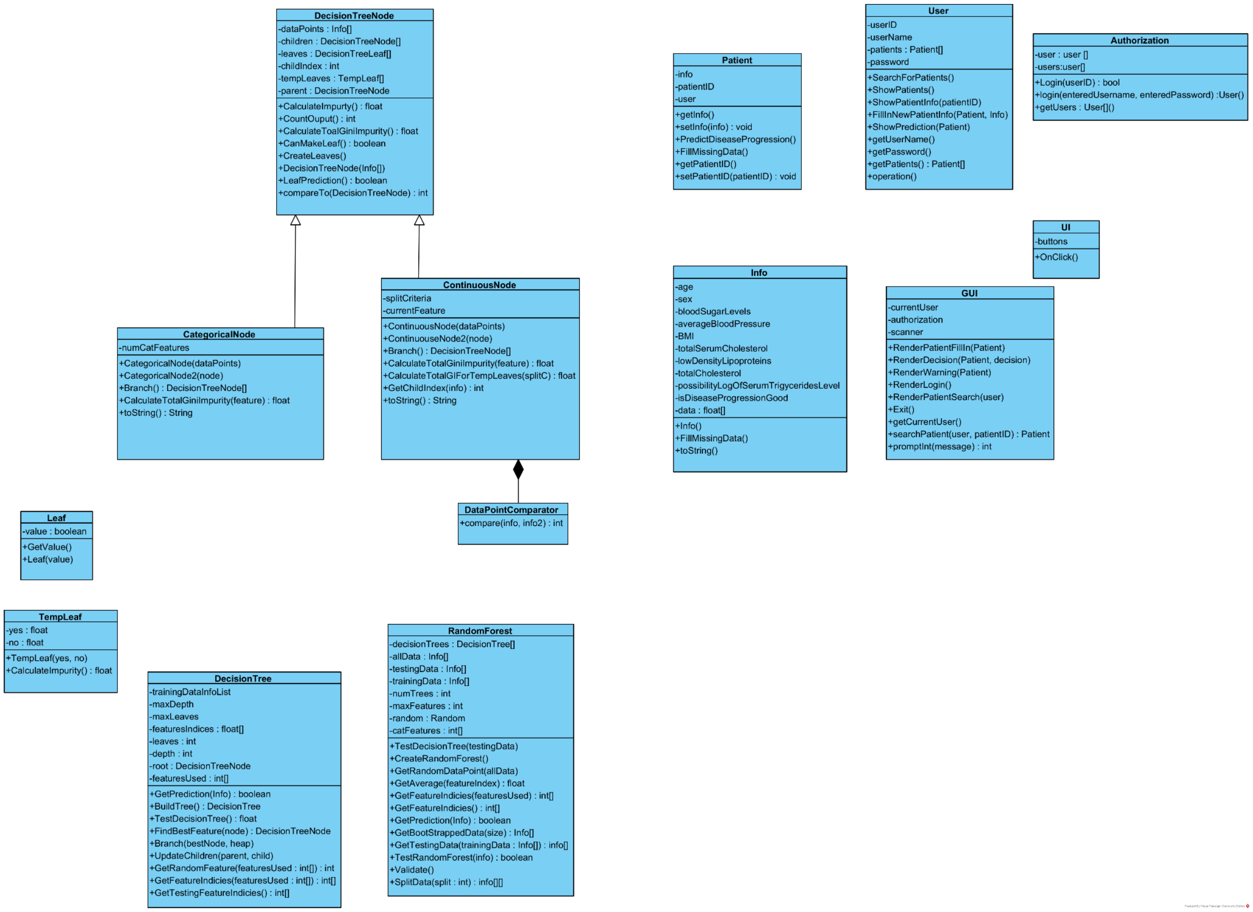
We're using this architecture because we want to separate the logic, the interaction, and the interface. This also allows us to have multiple ways to interact with our data via user warning and random forest predictions. This is the best for our project because it can organize our data more efficiently than the other architecture patterns. Our data needs to be both interactive and still be organized in a way that certain data is not available to every user, for example nurses, doctors and patients will not all have access to the same data, each user will have a different view derived from similar data. In order to update information as we get it from the patient we need to use the controller. The model will make the predictions and then notify the user through the view.

Model Diagram:

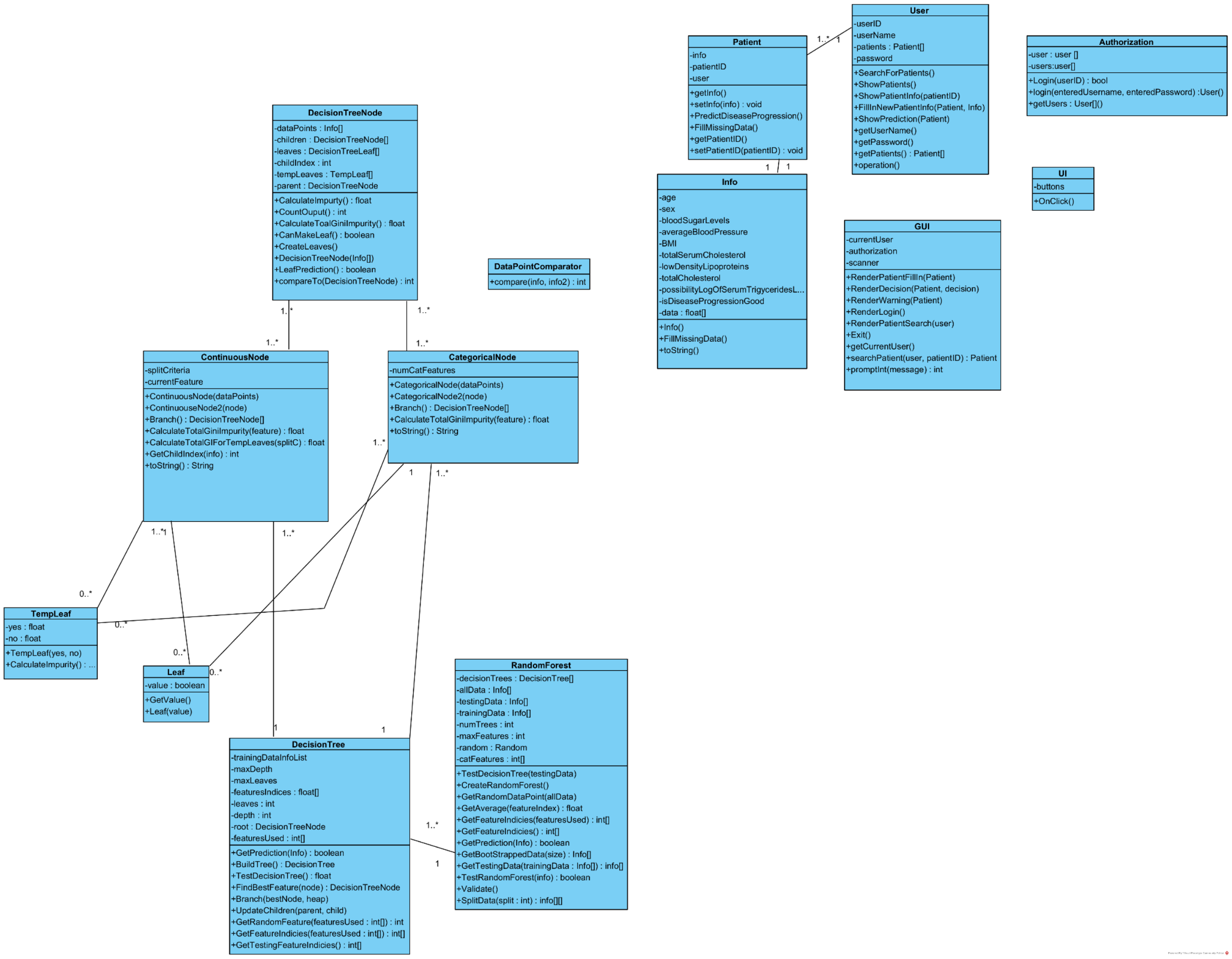


Structural Modeling

Class Diagram:


DecisionTreeNode
-dataPoints : Info[]
-children : DecisionTreeNode[]
-leaves : DecisionTreeLeaf[]
-childIndex : int
-tempLeaves : TempLeaf[]
-parent : DecisionTreeNode
+CalculateImpurty() : float
+CountOuput() : int
+CalculateToalGiniImpurity() : float
+CanMakeLeaf() : boolean
+CreateLeaves()
+DecisionTreeNode(Info[])
+LeafPrediction() : boolean
+compareTo(DecisionTreeNode) : int
Patient
-info
-patientID
-user
+getInfo()
+setInfo(info) : void
+PredictDiseaseProgression()
+FillMissingData()
+getPatientID()
+setPatientID(patientID) : void
User
-userID
-userName
-patients : Patient[]
-password
+SearchForPatients()
+ShowPatients()
+ShowPatientInfo(patientID)
+FillInNewPatientInfo(Patient, Info)
+ShowPrediction(Patient)
+getUserName()
+getPassword()
+getPatients() : Patient[]
+operation()
Authorization
-user : user []
-users:user[]
+Login(userID) : bool
+login(enteredUsername, enteredPassword) :User()
+getUsers : User[]()
UI
-buttons
+OnClick()
CategoricalNode
-numCatFeatures
+CategoricalNode(dataPoints)
+CategoricalNode2(node)
+Branch() : DecisionTreeNode[]
+CalculateTotalGiniImpurity(feature) : float
+toString() : String
ContinuousNode
-splitCriteria
-currentFeature
+ContinuousNode(dataPoints)
+ContinuouseNode2(node)
+Branch() : DecisionTreeNode[]
+CalculateTotalGiniImpurity(feature) : float
+CalculateTotalGIForTempLeaves(splitC) : float
+GetChildIndex(info) : int
+toString() : String
Info
-age
-sex
-bloodSugarLevels
-averageBloodPressure
-BMI
-totalSerumCholesterol
-lowDensityLipoproteins
-totalCholesterol
-possibilityLogOfSerumTrigyceridesLevel
-isDiseaseProgressionGood
-data : float[]
+Info()
+FillMissingData()
+toString()
GUI
-currentUser
-authorization
-scanner
+RenderPatientFillIn(Patient)
+RenderDecision(Patient, decision)
+RenderWarning(Patient)
+RenderLogin()
+RenderPatientSearch(user)
+Exit()
+getCurrentUser()
+searchPatient(user, patientID) : Patient
+promptInt(message) : int
Leaf
-value : boolean
+GetValue()
+Leaf(value)
DataPointComparator
+compare(info, info2) : int
TempLeaf
-yes : float
-no : float
+TempLeaf(yes, no)
+CalculateImpurity() : float
DecisionTree
-trainingDataInfoList
-maxDepth
-maxLeaves
-featuresIndices : float[]
-leaves : int
-depth : int
-root : DecisionTreeNode
-featuresUsed : int[]
+GetPrediction(Info) : boolean
+BuildTree() : DecisionTree
+TestDecisionTree() : float
+FindBestFeature(node) : DecisionTreeNode
+Branch(bestNode, heap)
+UpdateChildren(parent, child)
+GetRandomFeature(featuresUsed : int[]) : int
+GetFeatureIndicies(featuresUsed : int[]) : int[]
+GetTestingFeatureIndicies() : int[]
RandomForest
-decisionTrees : DecisionTree[]
-allData : Info[]
-testingData : Info[]
-trainingData : Info[]
-numTrees : int
-maxFeatures : int
-random : Random
-catFeatures : int[]
+TestDecisionTree(testingData)
+CreateRandomForest()
+GetRandomDataPoint(allData)
+GetAverage(featureIndex) : float
+GetFeatureIndicies(featuresUsed) : int[]
+GetFeatureIndicies() : int[]
+GetPrediction(Info) : boolean
+GetBootStrappedData(size) : Info[]
+GetTestingData(trainingData : Info[]) : info[]
+TestRandomForest(info) : boolean
+Validate()
+SplitData(split : int) : info[][]

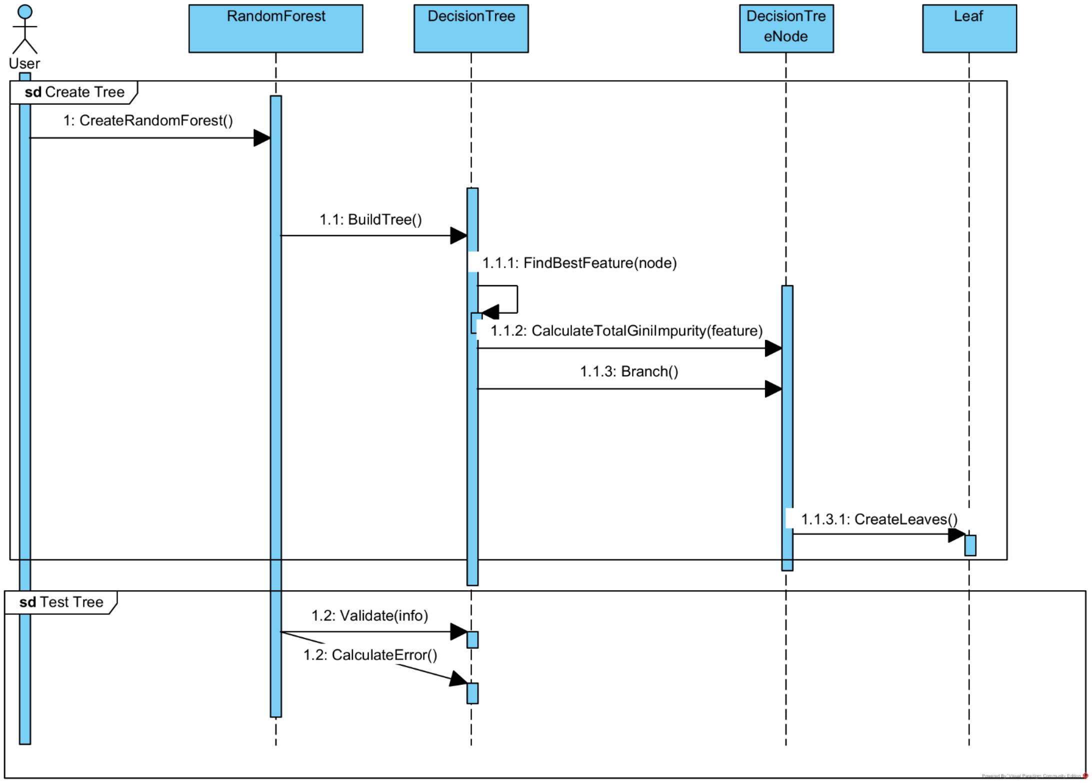
Association Diagram:



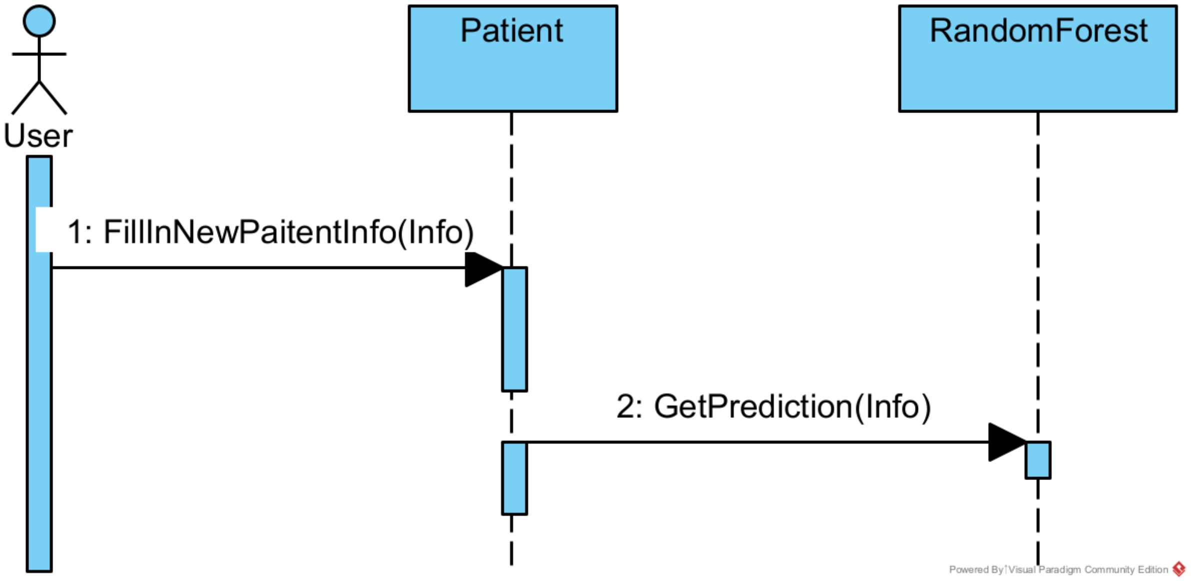
DecisionTreeNode
-dataPoints : Info[]
-children : DecisionTreeNode[]
-leaves : DecisionTreeLeaf[]
-childIndex : int
-tempLeaves : TempLeaf[]
-parent : DecisionTreeNode
+CalculateImpurty() : float
+CountOuput() : int
+CalculateToalGiniImpurity() : float
+CanMakeLeaf() : boolean
+CreateLeaves()
+DecisionTreeNode(Info[])
+LeafPrediction() : boolean
+compareTo(DecisionTreeNode) : int
DataPointComparator
+compare(info, info2) : int
1..*
1..*
1..*
1..*
ContinuousNode
-splitCriteria
-currentFeature
+ContinuousNode(dataPoints)
+ContinuouseNode2(node)
+Branch() : DecisionTreeNode[]
+CalculateTotalGiniImpurity(feature) : float
+CalculateTotalGIForTempLeaves(splitC) : float
+GetChildIndex(info) : int
+toString() : String
CategoricalNode
-numCatFeatures
+CategoricalNode(dataPoints)
+CategoricalNode2(node)
+Branch() : DecisionTreeNode[]
+CalculateTotalGiniImpurity(feature) : float
+toString() : String
1..*
1
1..*
1..*1
1..*
0..*
0..*
0..*
0..*
TempLeaf
-yes : float
-no : float
+TempLeaf(yes, no)
+CalculateImpurity() : ...
Leaf
-value : boolean
+GetValue()
+Leaf(value)
1
1
DecisionTree
-trainingDataInfoList
-maxDepth
-maxLeaves
-featuresIndices : float[]
-leaves : int
-depth : int
-root : DecisionTreeNode
-featuresUsed : int[]
+GetPrediction(Info) : boolean
+BuildTree() : DecisionTree
+TestDecisionTree() : float
+FindBestFeature(node) : DecisionTreeNode
+Branch(bestNode, heap)
+UpdateChildren(parent, child)
+GetRandomFeature(featuresUsed : int[]) : int
+GetFeatureIndicies(featuresUsed : int[]) : int[]
+GetTestingFeatureIndicies() : int[]
1..*
1
RandomForest
-decisionTrees : DecisionTree[]
-allData : Info[]
-testingData : Info[]
-trainingData : Info[]
-numTrees : int
-maxFeatures : int
-random : Random
-catFeatures : int[]
+TestDecisionTree(testingData)
+CreateRandomForest()
+GetRandomDataPoint(allData)
+GetAverage(featureIndex) : float
+GetFeatureIndicies(featuresUsed) : int[]
+GetFeatureIndicies() : int[]
+GetPrediction(Info) : boolean
+GetBootStrappedData(size) : Info[]
+GetTestingData(trainingData : Info[]) : info[]
+TestRandomForest(info) : boolean
+Validate()
+SplitData(split : int) : info[][]
Patient
-info
-patientID
-user
+getInfo()
+setInfo(info) : void
+PredictDiseaseProgression()
+FillMissingData()
+getPatientID()
+setPatientID(patientID) : void
1..*
1
User
-userID
-userName
-patients : Patient[]
-password
+SearchForPatients()
+ShowPatients()
+ShowPatientInfo(patientID)
+FillInNewPatientInfo(Patient, Info)
+ShowPrediction(Patient)
+getUserName()
+getPassword()
+getPatients() : Patient[]
+operation()
Authorization
-user : user []
-users:user[]
+Login(userID) : bool
+login(enteredUsername, enteredPassword) :User()
+getUsers : User[]()
1
1
Info
-age
-sex
-bloodSugarLevels
-averageBloodPressure
-BMI
-totalSerumCholesterol
-lowDensityLipoproteins
-totalCholesterol
-possibilityLogOfSerumTrigyceridesL...
-isDiseaseProgressionGood
-data : float[]
+Info()
+FillMissingData()
+toString()
UI
-buttons
+OnClick()
GUI
-currentUser
-authorization
-scanner
+RenderPatientFillIn(Patient)
+RenderDecision(Patient, decision)
+RenderWarning(Patient)
+RenderLogin()
+RenderPatientSearch(user)
+Exit()
+getCurrentUser()
+searchPatient(user, patientID) : Patient
+promptInt(message) : int

Interaction Modeling

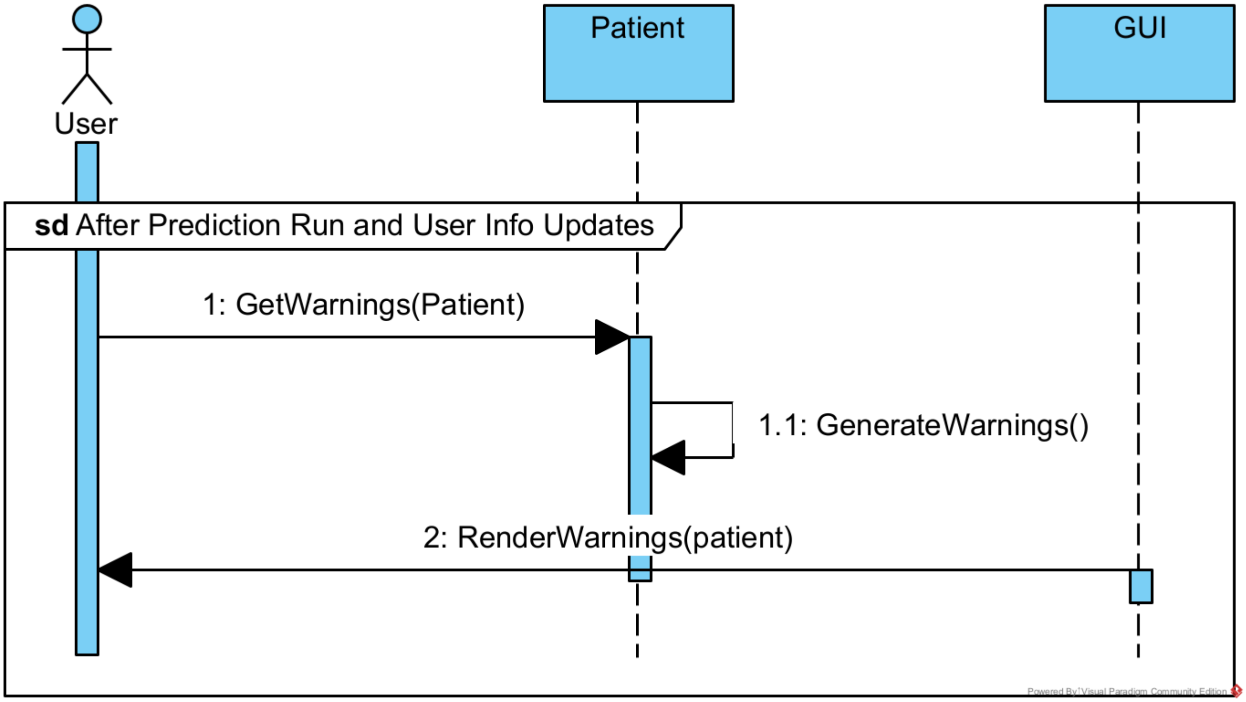
RandomForest Sequence Diagram:



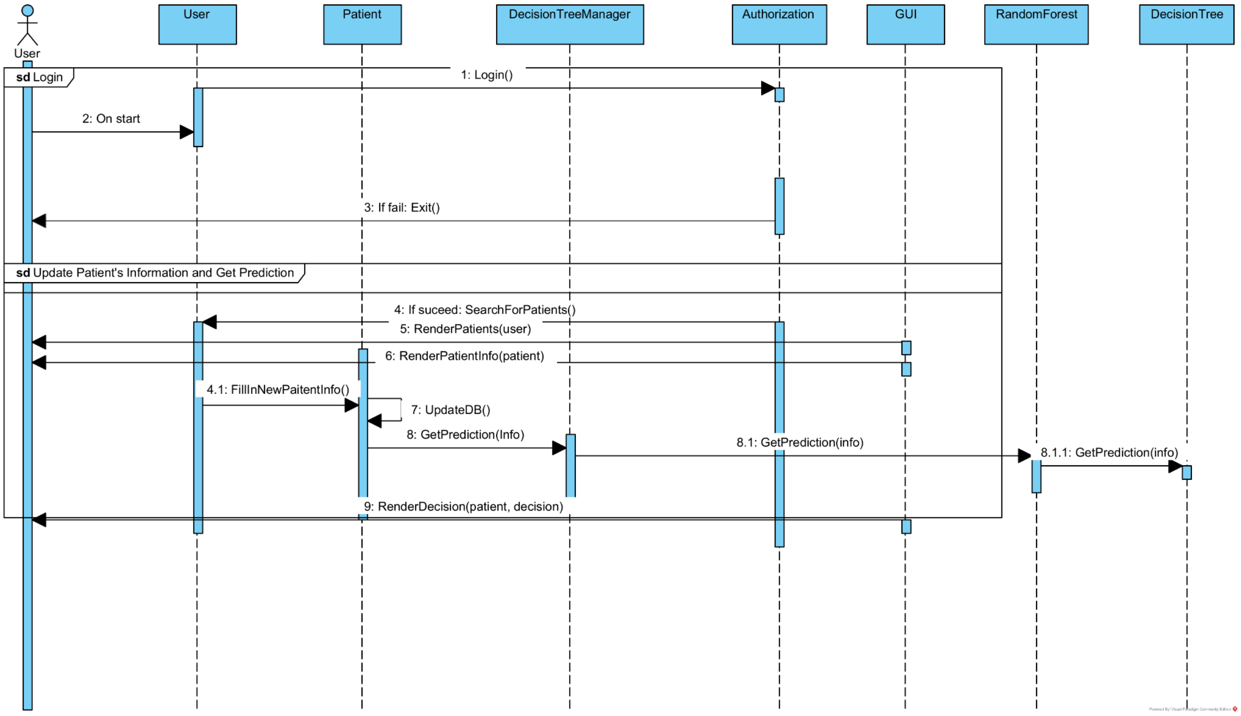
Prediction Sequence Diagram:


User
Patient
RandomForest
1: FillInNewPaitentInfo(Info)
2: GetPrediction(Info)
Powered By Visual Paradigm Community Edition

User Warnings Sequence Diagram:


User
Patient
GUI
sd After Prediction Run and User Info Updates
1: GetWarnings(Patient)
1.1: GenerateWarnings()
2: RenderWarnings(patient)
Powered By Visual Paradigm Community Edition

Main Program Structure Sequence Diagram



Algorithms & data structures

- We mostly used ArrayLists to keep track of all our data, since they are easy to iterate and easy to change.
- For the heap that stores the nodes that will be split, we use a priority queue that compares gini impurity values
- For sorting in continuous nodes, we used the Java array list sort implementation instead of writing our own

Programming environment

-Programming language

Java

-External software:

We used JUnit to run test cases. But for the random forest and user side code, we only used pure Java.

Test Cases and Results:

Full Dataset:

- Total size: 442
- Features in order with index:
    - int age = 0
    - int sex = 1
    - float BMI = 2
    - float bloodPressure = 3
    - float totalSerumCholesterol = 4
    - float lowDensityLipoproteins = 5
    - float highDensityLipoproteins = 6
    - float totalCholesterol = 7
    - float possibilityLogOfSerumTriglyceridesLevel = 8
    - float bloodSugarLevels = 9
    - Float DiseaseProgressionLevel = 10

Continuous Features: 0, 2, 3, 4, 5, 6, 7, 8, 9

Categorical features: 1

Will use 5 fold Cross Validation

Test Case Dataset:

Features to test the decision tree on: 3 for continuous, 1 for categorical nodes

Training data:

a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}
b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}
c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}
i ={ 51, 1, 23.4, 87, 220, 108.8, 93, 2, 4.5109, 82, true }
j = { 50, 2, 29.2, 119, 162, 85.2, 54, 3, 4.7362, 95, false}
k = { 59, 2, 27.2, 107, 158, 102, 39, 4, 4.4427, 93, true}
l = { 52, 1, 27, 78.33, 134, 73, 44, 3.05, 4.4427, 69, false}
m = { 69, 2, 24.5, 108, 243, 136.4, 40, 6, 5.8081, 100, false}
n = { 53, 1, 24.1, 105, 184, 113.4, 46, 4, 4.8122, 95, true}

Testing Data:

e = {42, 2, 30.6, 101, 269, 172.2, 50, 5, 5.4553, 106, false}
f = {25, 2, 22.6, 85, 130, 71, 48, 3, 4.0073, 81, true}
g = {52, 2, 19.7, 81, 152, 53.4, 82, 2, 4.4188, 82, true},
h = {34, 1, 21.2, 84, 254, 113.4, 52, 5, 6.0936, 92, true}

Unit Test Cases:

RandomForest:

- ★ CreateRandomForest()
  - ○ Input:
    a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}
    b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}
    c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
    d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}
    i = {51, 1, 23.4, 87, 220, 108.8, 93, 2, 4.5109, 82, true }
    j = {50, 2, 29.2, 119, 162, 85.2, 54, 3, 4.7362, 95, false}

k = {59, 2, 27.2, 107, 158, 102, 39, 4, 4.4427, 93, true}

l = {52, 1, 27, 78.33, 134, 73, 44, 3.05, 4.4427, 69, false}

m = {69, 2, 24.5, 108, 243, 136.4, 40, 6, 5.8081, 100, false}

n = {53, 1, 24.1, 105, 184, 113.4, 46, 4, 4.8122, 95, true}

- Expected Output:
    - DecisionTree1:

Root =continuousNode{splitCriteria = 107.5, currentFeature = 3, parent = null, children = 2, leaves = 0, dataPoints = {a,b,c,d,i,j,k,l,m,n}}

Root.children[0] = continuousNode{splitCriteria = 25.15, currentFeature = 2, parent = root, children = 0, leaves = 2, dataPoints = {a,b,i,l,n}}

Root.children[0].leaves[0] = true

Root.children[0].leaves[1] = false

Root.children[1] = continuousNode{splitCriteria = 4.589, currentFeature = 8, parent = root, children = 0, leaves = 2, dataPoints = {c,d,j,k,m}}

Root.children[1].leaves[0] = true

Root.children[1].leaves[1] = false

- DecisionTree2:

Root =continuousNode{splitCriteria = 50.5, currentFeature = 0, parent = null, children = 1, leaves = 1, dataPoints = {a,b,c,d,i,j,k,l,m,n}}

Root.leaves[0] = false

Root.children[1] = continuousNode{splitCriteria = 46,
currentFeature = 6, parent = root, children = 1, leaves = 1,
dataPoints = {i,l,n,k,b,m}}

Root.children[1].leaves[1] = true

Root.children[1].children[0] = continuousNode{splitCriteria = 27.1,
currentFeature = 2, parent = root.children[1], children = 0, leaves =
2,
dataPoints = {k,m,l}}

Root.children[1].children[0].leaves[0] = false
Root.children[1].children[0].leaves[1] = true

- DecisionTree3:

Root =continuousNode{splitCriteria = 4.9246, currentFeature = 8,
parent = null, children = 1, leaves = 1,
dataPoints = {a,b,c,d,i,j,k,l,m,n}}

Root.leaves[1] = false

Root.children[0] = continuousNode{splitCriteria = 173,
currentFeature = 4, parent = root, children = 1, leaves = 1,
dataPoints = {b,l,k,i,j,n}}

Root.children[0].leaves[1] = true

Root.children[0].children[0] = continuousNode{splitCriteria =55.5,
currentFeature = 0, parent = root.children[0], children = 0, leaves =
2,

dataPoints = {l,k,j}}

Root.children[0].children[0].leaves[0] = false
Root.children[0].children[0].leaves[1] = true

RandomForest:

- Float GetAverage(int feature)
  - Input:

    Feature = 2,
    a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}
    b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}
    c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
    d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}
    e = {42, 2, 30.6, 101, 269, 172.2, 50, 5, 5.4553, 106, false}
    f = {25, 2, 22.6, 85, 130, 71, 48, 3, 4.0073, 81, true}
    g = {52, 2, 19.7, 81, 152, 53.4, 82, 2, 4.4188, 82, true},
    h = {34, 1, 21.2, 84, 254, 113.4, 52, 5, 6.0936, 92, true}
    i ={ 51, 1, 23.4, 87, 220, 108.8, 93, 2, 4.5109, 82, true }
    j = { 50, 2, 29.2, 119, 162, 85.2, 54, 3, 4.7362, 95, false}
    k = { 59, 2, 27.2, 107, 158, 102, 39, 4, 4.4427, 93, true}
    l = { 52, 1, 27, 78.33, 134, 73, 44, 3.05, 4.4427, 69, false}
    m = { 69, 2, 24.5, 108, 243, 136.4, 40, 6, 5.8081, 100, false}
    n = { 53, 1, 24.1, 105, 184, 113.4, 46, 4, 4.8122, 95, true}
  - Expected Output: 25.33357143
  - Actual Output: 25.33357143
- ArrayList<Info> GetBootStrappedData(int size)
  - Input:

    Size: 10,
    Datapoints:

{
}

  - Expected Output: any 10 datapoints from the dataset, with replacement
  - Actual Output: 10 random datapoints from the dataset
- ArrayList<Info> GetTestingData(ArrayList<Info> trainingData)
  - Input:
    Training Data: {a,b,c,d,i,j,k,l,m,n}
  - Expected Output:{e,f,g,h}
  - Actual Output:{e,f,g,h}
- CreateRandomForest()
  - Input: numTrees = 3,maxFeatures = 5, bsDataSize(bootstraped data size) = 10, dataPoints ={a,b,c,d,e,f,g,h,i,j,k,l,m,n}
  - Expected Output: Any 3 different trees with a max depth of 3, since they tend not to grow longer with such a small dataset, and are subject to randomness
  - Actual Output:
    - Tree1 =
      Root = ContinuousNode {splitCriteria = 24.400002, currentFeature = 2, children = empty, leaves = 2}
      Root.leaves[0] = true
      Root.leaves[1] = false
    - Tree2 =
      Root = ContinuousNode {splitCriteria = 237, currentFeature = 4, children = empty, leaves = 2}
      Root.leaves[0] = true
      Root.leaves[1] = false
    - Tree3 =
      Root = ContinuousNode {splitCriteria = 213.5, currentFeature =4, children = 1, leaves = 1}

Root.leaves[1] = true

Root.children[0] = ContinuousNode {splitCriteria = 51, currentFeature = 0, children = 1, leaves = 1}
Root.children.leaves[0] = false

Root.children[0].children[1] = ContinuousNode {splitCriteria = 79.665, currentFeature = 3, children = 0, leaves =2 }
Root.children[0].children[1].leaves[0] = false
Root.children[0].children[1].leaves[1] = true

★ boolean GetPrediction(Info info)

- ○ Input:
  e = {42, 2, 30.6, 101, 269, 172.2, 50, 5, 5.4553, 106, false}
  - ■ DecisionTree1:

Root =continuousNode{splitCriteria = 107.5, currentFeature = 3, parent = null, children = 2, leaves = 0, dataPoints = {a,b,c,d,i,j,k,l,m,n}}
Root.children[0] = continuousNode{splitCriteria = 25.15, currentFeature = 2, parent = root, children = 0, leaves = 2, dataPoints = {a,b,i,l,n}}

Root.children[0].leaves[0] = true
Root.children[0].leaves[1] = false

Root.children[1] = continuousNode{splitCriteria = 4.589, currentFeature = 8, parent = root, children = 0, leaves = 2, dataPoints = {c,d,j,k,m}}

Root.children[1].leaves[0] = true
Root.children[1].leaves[1] = false

- DecisionTree2:

Root =continuousNode{splitCriteria = 50.5, currentFeature = 0,
parent = null, children = 1, leaves = 1,
dataPoints = {a,b,c,d,i,j,k,l,m,n}}

Root.leaves[0] = false

Root.children[1] = continuousNode{splitCriteria = 46,
currentFeature = 6, parent = root, children = 1, leaves = 1,
dataPoints = {i,l,n,k,b,m}}

Root.children[1].leaves[1] = true

Root.children[1].children[0] = continuousNode{splitCriteria = 27.1,
currentFeature = 2, parent = root.children[1], children = 0, leaves =
2,
dataPoints = {k,m,l}}

Root.children[1].children[0].leaves[0] = false
Root.children[1].children[0].leaves[1] = true

- DecisionTree3:

Root =continuousNode{splitCriteria = 4.9246, currentFeature = 8,
parent = null, children = 1, leaves = 1,
dataPoints = {a,b,c,d,i,j,k,l,m,n}}

Root.leaves[1] = false

Root.children[0] = continuousNode{splitCriteria = 173, currentFeature = 4, parent = root, children = 1, leaves = 1, dataPoints = {b,l,k,i,j,n}}

Root.children[0].leaves[1] = true

Root.children[0].children[0] = continuousNode{splitCriteria =55.5, currentFeature = 0, parent = root.children[0], children = 0, leaves = 2,
dataPoints = {l,k,j}}

Root.children[0].children[0].leaves[0] = false
Root.children[0].children[0].leaves[1] = true

- Expected Output:
  DecisionTree1: true,
  DecisionTree2: false,
  DecisionTree3: false
  Returns False
- Actual Output:
  Root = continuousNode{splitCriteria = 94, currentFeature = 9, children = 2, leaves = 0}

  Root.children[0] = categoricalNode{children = 0, leaves = 2}
  Root.children[0].leaves[0] = true
  Root.children[0].leaves[1] = false

  Root.children[1] =continuousNode{splitCriteria = 149.8, currentFeature = 5, children = 0, leaves = 2}
  Root.children[1].leaves[0] = false
  Root.children[1].leaves[1] = true

Root =continuousNode{splitCriteria = 3.025, currentFeature = 7, children = 0, leaves = 2}
Root.leaves[0] = true
Root.leaves[1] = false

Returned: false, with tree 1= false, tree2 = true, tree3 = false

- ★ ArrayList<ArrayList<Info>> SplitData(int split)
    - ○ Input: split = 5
      allData = {a,b,c,d,e,f,g,h,i,j,k,l,m,n}
    - ○ Expected Output:
      { {a, f, k}, {b, g, l}, {c, h, m}, {d, i, n} ,{e, j} }
    - ○ Actual Output:
      { {a, f, k}, {b, g, l}, {c, h, m}, {d, i, n} ,{e, j} }

- ★ Validate()
    - ○ Input:
      allData = {a,b,c,d,e,f,g,h,i,j,k,l,m,n},
      Split = 5,
      numTrees = 3,
      maxFeatures = 5
    - ○ Expected Output: any 3 decision trees since it is random
    - ○ Actual Output:
        - ■ RandomForest 1:
            - ● Tree1:
              root = continuousNode{ splitCriteria: 24.3, currentFeature:2, children: 2, leaves: 2}
              root.leaves[0] = true
              root.leaves[1] = false

            - ● Tree2:

root = continuousNode{ splitCriteria: 106.5, currentFeature:3, children: 2, leaves: 1}

root.leaves[1] = false

root.children[0] = continuousNode{ splitCriteria: 25.55, currentFeature:2, children: 2, leaves: 2}

root.children[0].leaves[0] = true

root.children[0].leaves[1] = false

- Tree 3:

  root = continuousNode{ splitCriteria: 106.5, currentFeature:3, children: 2, leaves: 1}

  root.leaves[1] = false

  root.children[0] = continuousNode{ splitCriteria: 143.8, currentFeature:5, children: 2, leaves: 1}

  root.children[0].leaves[1] = false

  root.children[0].children[0] = continuousNode{ splitCriteria: 25.55, currentFeature:2, children: 2, leaves: 2}

  root.children[0].children[0].leaves[0] = true

  root.children[0].children[0].leaves[1] = false

  correct/total ratio = ⅔ = 66%

- RandomForest 2:
  - Tree1:

    root = continuousNode{ splitCriteria: 96.5, currentFeature:9, children: 2, leaves: 1}

    root.leaves[1] = false

root.children[0] = continuousNode{ splitCriteria: 24.400002, currentFeature:2, children: 2, leaves: 1}

root.children[0].leaves[0] = true

root.children[0].children[1] = continuousNode{ splitCriteria: 4.58945, currentFeature:8, children: 2, leaves: 2}

root.children[0].children[1].leaves[0] = true

root.children[0].children[1].leaves[1] = false

- Tree2:

  root = continuousNode{ splitCriteria: 96.5, currentFeature:9, children: 2, leaves: 1}

  root.leaves[1] = false

  root.children[0] = continuousNode{ splitCriteria: 50.5, currentFeature:0, children: 2, leaves: 1}

  root.children[0].leaves[1] = true

  root.children[0].children[0] = continuousNode{ splitCriteria: 23.650002, currentFeature:2, children: 2, leaves: 2}

  root.children[0].children[0].leaves[0] = true

  root.children[0].children[0].leaves[1] = false

- Tree3:

  root = continuousNode{ splitCriteria: 107.5, currentFeature:3, children: 2, leaves: 1}

  root.leaves[1] = false

  root.children[0] = continuousNode{ splitCriteria: 4.9374, currentFeature:8, children: 2, leaves: 1}

  root.children[0].leaves[0] = true

root.children[0].children[1] = continuousNode{ splitCriteria: 51.0, currentFeature:6, children: 2, leaves: 2}

root.children[0].children[1].leaves[0] = false

root.children[0].children[1].leaves[1] = true

correct/total ratio = ⅔ = 66%

- RandomForest 3:
    - Tree1:

        root = continuousNode{ splitCriteria: 71.0, currentFeature:6, children: 2, leaves: 1}

        root.leaves[1] = true

        root.children[0] = continuousNode{ splitCriteria: 52.5, currentFeature:0, children: 2, leaves: 1}

        root.children[0].leaves[1] = true

        root.children[0].children[0] = continuousNode{ splitCriteria: 132.0, currentFeature:4, children: 2, leaves: 2}

        root.children[0].children[0].leaves[0] = true

        root.children[0].children[0].leaves[1] = false

    - Tree2:

        root = continuousNode{ splitCriteria: 71.0, currentFeature:6, children: 2, leaves: 1}

        root.leaves[1] = true

        root.children[0] = continuousNode{ splitCriteria: 25.150002, currentFeature:2, children: 2, leaves: 1}

root.children[0].leaves[0] = true

root.children[0].children[1] = continuousNode{ splitCriteria: 55.5, currentFeature:0, children: 2, leaves: 2}

root.children[0].children[1].leaves[0] = false

root.children[0].children[1].leaves[1] = true

- Tree3:

  root = continuousNode{ splitCriteria: 50.5, currentFeature:0, children: 2, leaves: 0}

  root.children[1] = continuousNode{ splitCriteria: 143.0, currentFeature:4, children: 2, leaves: 2}

  root.children[1].leaves[0] = false

  root.children[1].leaves[1] = true

  root.children[0] = continuousNode{ splitCriteria: 84.5, currentFeature:9, children: 2, leaves: 2}

  root.children[0].leaves[0] = true

  root.children[0].leaves[1] = false

correct/total ratio = 3/3 = 100%

- RandomForest 4:
  - Tree1:

    root = continuousNode{ splitCriteria: 4.43075, currentFeature:8, children: 2, leaves: 1}

    root.leaves[0] = true

    root.children[1] = continuousNode{ splitCriteria: 124.899994, currentFeature:5, children: 2, leaves: 1}

    root.children[1].leaves[1] = false

root.children[1].children[0] = continuousNode{ splitCriteria: 23.7, currentFeature:2, children: 2, leaves: 1}
root.children[1].children[0].leaves[0] = true
root.children[1].children[0].children[1] = continuousNode{ splitCriteria: 41.5, currentFeature:6, children: 2, leaves: 2}
root.children[1].children[0].children[1].leaves[0] = true
root.children[1].children[0].children[1].leaves[1] = false

- Tree2:
  root = continuousNode{ splitCriteria: 24.25, currentFeature:2, children: 2, leaves: 1}
  root.leaves[0] = true
  root.children[1] = continuousNode{ splitCriteria: 55.5, currentFeature:0, children: 2, leaves: 1}
  root.children[1].leaves[0] = false
  root.children[1].children[1] = continuousNode{ splitCriteria: 107.5, currentFeature:3, children: 2, leaves: 2}
  root.children[1].children[1].leaves[0] = true
  root.children[1].children[1].leaves[1] = false

- Tree3:
  root = continuousNode{ splitCriteria: 94.0, currentFeature:3, children: 2, leaves: 0}

root.children[1] = continuousNode{ splitCriteria: 103.7, currentFeature:5, children: 2, leaves: 1}
root.children[1].leaves[1] = false
root.children[1].children[0] = continuousNode{ splitCriteria: 94.0, currentFeature:9, children: 2, leaves: 2}
root.children[1].children[0].leaves[0] = true
root.children[1].children[0].leaves[1] = false
root.children[0] = continuousNode{ splitCriteria: 25.5, currentFeature:2, children: 2, leaves: 2}
root.children[0].leaves[0] = true
root.children[0].leaves[1] = false

correct/total ratio =3/3 = 100%

- RandomForest 5:
  - Tree1:
    root = continuousNode{ splitCriteria: 24.3, currentFeature:2, children: 2, leaves: 1}
    root.leaves[0] = true
    root.children[1] = continuousNode{ splitCriteria: 4.73985, currentFeature:8, children: 2, leaves: 1}
    root.children[1].leaves[1] = false
    root.children[1].children[0] = continuousNode{ splitCriteria: 146.0, currentFeature:4, children: 2, leaves: 2}
    root.children[1].children[0].leaves[0] = false
    root.children[1].children[0].leaves[1] = true

  - Tree2:

root = continuousNode{ splitCriteria: 4.9246, currentFeature:8, children: 2, leaves: 0}
root.children[1] = continuousNode{ splitCriteria: 90.5, currentFeature:3, children: 2, leaves: 2}
root.children[1].leaves[0] = true
root.children[1].leaves[1] = false
root.children[0] = continuousNode{ splitCriteria: 71.0, currentFeature:9, children: 2, leaves: 2}
root.children[0].leaves[0] = false
root.children[0].leaves[1] = true

- Tree3:
  root = continuousNode{ splitCriteria: 24.3, currentFeature:2, children: 2, leaves: 1}
  root.leaves[0] = true
  root.children[1] = continuousNode{ splitCriteria: 172.0, currentFeature:4, children: 2, leaves: 1}
  root.children[1].leaves[1] = false
  root.children[1].children[0] = continuousNode{ splitCriteria: 41.5, currentFeature:6, children: 2, leaves: 2}
  root.children[1].children[0].leaves[0] = true
  root.children[1].children[0].leaves[1] = false

correct/total ratio = 2/2 = 100%

DecisionTree:

- ★ DecisionTree BuildTree()
    - ○ Input:
      a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}
      b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}
      c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
      d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}
      i ={ 51, 1, 23.4, 87, 220, 108.8, 93, 2, 4.5109, 82, true }
      j = { 50, 2, 29.2, 119, 162, 85.2, 54, 3, 4.7362, 95, false}
      k = { 59, 2, 27.2, 107, 158, 102, 39, 4, 4.4427, 93, true}
      l = { 52, 1, 27, 78.33, 134, 73, 44, 3.05, 4.4427, 69, false}
      m = { 69, 2, 24.5, 108, 243, 136.4, 40, 6, 5.8081, 100, false}
      n = { 53, 1, 24.1, 105, 184, 113.4, 46, 4, 4.8122, 95, true}
      featureIndicies ={1, 3, 5}
    - ○ Expected Output:
      Root =continuousNode{splitCriteria = 107.5, currentFeature = 3, parent = null, children = 2, leaves = 0, dataPoints = {a,b,c,d,i,j,k,l,m,n}}

      Root.children[0] = continuousNode{splitCriteria = 25.15, currentFeature = 2, parent = root, children = 0, leaves = 2, dataPoints = {a,b,i,k,l,n}}
      Root.children[0].leaves[0] = true
      Root.children[0].children[1] = continuousNode {splitCriteria = 90.5, currentFeature = 9, parent = root.children[0], children = 0, leaves = 2, datapoints ={a,k,l}}

      Root.children[0].children[1].leaves[0] = false
      Root.children[0].children[1].leaves[1] = true

      Root.children[1] = null
      Root.leaves[1] = false

- Actual Output:
  Root =continuousNode{splitCriteria = 107.5, currentFeature = 3, parent = null, children = 2, leaves = 0, dataPoints = {a,b,c,d,i,j,k,l,m,n}}

  Root.children[0] = continuousNode{splitCriteria = 25.15, currentFeature = 2, parent = root, children = 0, leaves = 2, dataPoints = {a,b,i,k,l,n}}
  Root.children[0].leaves[0] = true
  Root.children[0].children[1] = continuousNode {splitCriteria = 90.5, currentFeature = 9, parent = root.children[0], children = 0, leaves = 2, datapoints ={a,k,l}}

  Root.children[0].children[1].leaves[0] = false
  Root.children[0].children[1].leaves[1] = true

  Root.children[1] = null
  Root.leaves[1] = false

  Note: When run with JUnit, the test cases failed, but showed the same output as the expected, and in the "Comparison Failure" window in Intellij, Intellij said the "Contents are identical"

★ DecisionTreeDecisionTreeNode FindBestFeature(DecisionTreeNode node)

- Input:
  {parent = null, childIndex = 0,
  dataPoints = {
  a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}
  b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}
  c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
  d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}

    i ={ 51, 1, 23.4, 87, 220, 108.8, 93, 2, 4.5109, 82, true }
    j = { 50, 2, 29.2, 119, 162, 85.2, 54, 3, 4.7362, 95, false}
    k = { 59, 2, 27.2, 107, 158, 102, 39, 4, 4.4427, 93, true}
    l = { 52, 1, 27, 78.33, 134, 73, 44, 3.05, 4.4427, 69, false}
    m = { 69, 2, 24.5, 108, 243, 136.4, 40, 6, 5.8081, 100, false}
    n = { 53, 1, 24.1, 105, 184, 113.4, 46, 4, 4.8122, 95, true}
    }}
    featureIndicies ={1, 3, 5}

  - Expected Output: ContinuousNode{parent = null, childIndex = 0, currentFeature = 3, splitCriteria = 107.5
    dataPoints = {
    a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}
    b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}
    c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
    d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}
    i ={ 51, 1, 23.4, 87, 220, 108.8, 93, 2, 4.5109, 82, true }
    j = { 50, 2, 29.2, 119, 162, 85.2, 54, 3, 4.7362, 95, false}
    k = { 59, 2, 27.2, 107, 158, 102, 39, 4, 4.4427, 93, true}
    l = { 52, 1, 27, 78.33, 134, 73, 44, 3.05, 4.4427, 69, false}
    m = { 69, 2, 24.5, 108, 243, 136.4, 40, 6, 5.8081, 100, false}
    n = { 53, 1, 24.1, 105, 184, 113.4, 46, 4, 4.8122, 95, true}
    }}

  - Actual Output: ContinuousNode{parent = null, childIndex = 0, currentFeature = 3, splitCriteria = 107.5, datapoints = {a,b,c,d,i,j,k,l,m,n}
    Note: When run with JUnit, the test cases failed, but showed the same output as the expected, and in the "Comparison Failure" window in Intellij, Intellij said the "Contents are identical"

- Branch(DecisionTreeNode bestNode, PriorityQueue<DecisionTreeNode> heap)

- Input:
  ContinuousNode{parent = null, childIndex = 0,
  currentFeature = 3, splitCriteria = 107.5
  dataPoints = {
  a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}
  b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}
  c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
  d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}
  i ={ 51, 1, 23.4, 87, 220, 108.8, 93, 2, 4.5109, 82, true }
  j = { 50, 2, 29.2, 119, 162, 85.2, 54, 3, 4.7362, 95, false}
  k = { 59, 2, 27.2, 107, 158, 102, 39, 4, 4.4427, 93, true}
  l = { 52, 1, 27, 78.33, 134, 73, 44, 3.05, 4.4427, 69, false}
  m = { 69, 2, 24.5, 108, 243, 136.4, 40, 6, 5.8081, 100, false}
  n = { 53, 1, 24.1, 105, 184, 113.4, 46, 4, 4.8122, 95, true}
  }}

  Empty heap because it only had the root node
- Expected Output:
  child 1 = {parent = continuous node, childIndex = 0, leaves = empty,
  children = empty,
  Datapoints{
  a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}
  b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}
  i ={ 51, 1, 23.4, 87, 220, 108.8, 93, 2, 4.5109, 82, true }
  k = { 59, 2, 27.2, 107, 158, 102, 39, 4, 4.4427, 93, true}
  l = { 52, 1, 27, 78.33, 134, 73, 44, 3.05, 4.4427, 69, false}
  n = { 53, 1, 24.1, 105, 184, 113.4, 46, 4, 4.8122, 95, true}
  } }
  Child2 = {parent = continuous node, childIndex = 1, leaves = empty,
  children = empty,
  Datapoints:{

c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}
j = { 50, 2, 29.2, 119, 162, 85.2, 54, 3, 4.7362, 95, false}
m = { 69, 2, 24.5, 108, 243, 136.4, 40, 6, 5.8081, 100, false}
}}

- ○ Actual Output:
  child 1 = {parent = continuous node, childIndex = 0, leaves = empty, children = empty,
  Datapoints{
  a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}
  b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}
  i ={ 51, 1, 23.4, 87, 220, 108.8, 93, 2, 4.5109, 82, true }
  k = { 59, 2, 27.2, 107, 158, 102, 39, 4, 4.4427, 93, true}
  l = { 52, 1, 27, 78.33, 134, 73, 44, 3.05, 4.4427, 69, false}
  n = { 53, 1, 24.1, 105, 184, 113.4, 46, 4, 4.8122, 95, true}
  } }
  Child2 = {parent = continuous node, childIndex = 1, leaves = empty, children = empty,
  Datapoints:{
  c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
  d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}
  j = { 50, 2, 29.2, 119, 162, 85.2, 54, 3, 4.7362, 95, false}
  m = { 69, 2, 24.5, 108, 243, 136.4, 40, 6, 5.8081, 100, false}
  }}
  However the test fails likely because the expected decision tree did not update it’s heap, while the actual code-run decision tree did. But there is no difference in contents besides that.

★ boolean GetPrediction(Info info)

- ○ Input:
  g = {52, 2, 19.7, 81, 152, 53.4, 82, 2, 4.4188, 82, 77}

Root =continuousNode{splitCriteria = 107.5, currentFeature = 3, parent = null, children = 2, leaves = 0,
dataPoints = {a,b,c,d,i,j,k,l,m,n}}

Root.children[0] = continuousNode{splitCriteria = 25.15, currentFeature = 2, parent = root, children = 0, leaves = 2,
dataPoints = {a,b,i,k,l,n}}
Root.children[0].leaves[0] = true

Root.children[0].children[1] = continuousNode {splitCriteria = 90.5, currentFeature = 9, parent = root.children[0], children = 0, leaves = 2, datapoints ={a,k,l}}

Root.children[0].children[1].leaves[0] = false
Root.children[0].children[1].leaves[1] = true

Root.children[1] = null
Root.leaves[1] = false

- Expected Output: true
- Actual Output: true

DecisionTreeNode:

- ★ int CanMakeLeaf()
  - Input:

    Datapoints = {
    {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false},
    {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true},
    {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
    {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}
    }

    - Expected Output: yes = 1, no = 3 so returns 0
    - Actual Output: 0

    - Input:
      Datapoints = {
       {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, true},
       {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true},
       {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, true},
       {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, true}
      }
    - Expected Output: yes = 4, no = 0 so returns 1
    - Actual Output: 1

    - Input:
      Datapoints = {} //empty
    - Expected Output: yes = 0, no = 0 so returns -1
    - Actual Output: -1
- ★ boolean LeafPrediction()
    - Input:
      Datapoints = {
       {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false},
       {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true},
       {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
       {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}
      }
    - Expected Output: false
    - Actual Output: false

CategoricalNode:

- ★ Float CalculateTotalGIForTempLeaves(int feature)
    - Input: feature = 1, numCatFeatures = 2,
       a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}

b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}
c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}

- ○ Expected Output:
  tempLeaf 1 = {yes = 1, no = 1}
  tempLeaf 2 = {yes = 0, no = 2}
  Returns: .25
- ○ Actual Output: .25

★ ArrayList<DecisionTreeNode> Branch()

- ○ Input:
  a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}
  b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}
  c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
  d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}
- ○ Expected Output:
  child1 ={parent = this node, childIndex = 0, dataPoints:
  {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true},
  {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}}
  child2 = null (because it gets turned into leaf)
- ○ Actual Output: *When run with JUnit, the test cases failed, but showed the same output as the expected, and in the "Comparison Failure" window in Intellij, Intellij said the "Contents are identical"*
  child1 ={parent = this node, childIndex = 0, dataPoints:
  {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true},
  {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}}
  child2 = null

ContinuousNode:

★ CalculateTotalGiniImpurity(int feature)

- ○ Input: feature = 3,
  a = {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false}
  b = {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}

c = {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false},
d = {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false}

- Expected Output: averageDataPoints = {94, 103, 113.5}
  totalImpurities = {0, .25, .33333}
  splitCriteria = 94
  returns 0
- Actual Output: 0

- ArrayList<DecisionTreeNode> Branch()
    - Input: splitCriteria = 94, feature = 3
    - Expected Output:
      child1 ={parent = this node, childIndex = 0, dataPoints:
      {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, true}}
      child2 = {parent = this node, childIndex = 1, dataPoints:
      {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, false},
      {47, 1, 30.3, 109, 207, 100.2, 70, 3, 5.2149, 98, false},
      {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false}}
      Child1 = null (became a leaf)
      Child 2 = null (became a leaf)
    - ActualOutput: {null, null}
- Int GetChildIndex(Info info)
    - Input: {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, false}, feature = 3, splitCriteria = 95
    - Expected Output: 1
    - Actual Output: 1

TempLeaf:

- Float Calculate Impurity()
    - Input: yes = 5, no = 6;
    - Expected Output: 0.49586776859
    - Actual Output: 0.49586776859

    - Input: yes = 0, no = 8

- ○ Expected Output: 0
- ○ Actual Output: 0

Info:

- ★ Info(int age, int sex, float BMI, float bloodPressure, float totalSerumCholesterol, float lowDensityLipoproteins, float highDensityLipoproteins, float totalCholesterol, float possibilityLogOfSerumTriglyceridesLevel, float bloodSugarLevels, boolean isDiseaseProgressionGood)
  - ○ *Note: When run with JUnit, the test cases failed, but showed the same output as the expected, and in the "Comparison Failure" window in Intellij, Intellij said the "Contents are identical"*

  - ○ Input: {50, 2, 0, 97, 0, 105.4, 0, 0, 5.0626, 88, 185}
  - ○ Expected Output: {50, 2, 26.375778, 97, 189.14027, 105.4, 49.78846, 4.070249, 5.0626, 88, false}
  - ○ Actual Output: {50, 2, 26.375778, 97, 189.14027, 105.4, 49.78846, 4.070249, 5.0626, 88, false}

  - ○ Input: {50, 2, 26.2,-1, -186, 105.4, 49, 4, 5.0626, 88, 185}
  - ○ Expected Output: {50, 2, 26.2, 94.647026, 189.14027, 105.4, 49, 4, 5.0626, 88, false}
  - ○ Actual Output: {50, 2, 26.2, 94.647026, 189.14027, 105.4, 49, 4, 5.0626, 88, false}

User

- ★ FillInNewPatientInfo(Patient patient, Info info)

  Patient:

  patientID, patientInfo
  patient1, {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, 185}
  patient2, {61, 1, 24, 91, 202, 115.4, 72, 3, 4.2905, 73, 34}
  patient3, {34, 2, 24.7, 118, 254, 184.2, 39, 7, 5.037, 81, 209}

- ○ Input: patient1, {50, 2, 26.2, 97, 186, 105.4, 49, 4, 5.0626, 88, 185}
- ○ Expected Output: patient1 failed to register (already exists)

- ○ Input: patient1, {50, 2, 40.5, 97, 186, 105.4, 49, 4, 5.0626, 88, 185}
- ○ Expected Output: patient1 successfully updated

- ○ Input: patient4, {47, 1, 30.3, 109, 207, 100.2, 70, 3,5.2149,98,167}
- ○ Expected Output: patient4 successfully registered

Authorization

★ boolean Login(String enteredUserName, String enteredPassword)

Users: (userID, userName, password)
1, user1, pass1
2, user2, pass2
3, user3, pass3

- ○ Input: user1, pass1
- ○ Expected Output: login successful
- ○ Actual Output: login successful

- ○ Input:user2, pass1
- ○ Expected Output: login failed
- ○ Actual Output: login failed

System Test Cases:

1. The machine learning model will accurately predict whether or not a patient’s diabetes progression is bad or good.
    - ★ A Doctor will sign into the system with their credentials, after the random forest was created. A list of patients will appear on their screen and then they will pick one from the list, entering that patient’s id into the console. And then the patient information will show up. But the doctor upon entering can change the information

depending on if there is new information(i.e. A doctor’s visit). Then after they submit all the new information, the information will be plugged into the database and the random forest will get a disease progression prediction from their information.

- Actual: Users sign into the system, pick a patient from a list of patients, the random forest is created and then the patient information is put in and predicts their disease progression, which is then shown to the screen.

2. We will train a machine learning model on previous diabetes data that will determine whether or not their diabetes progression is bad or good.
    - When the program starts up, the program will create a random forest using the database information. It will use hopefully all of it, to varying degrees when constructing each decision tree. When it is done, the login screen will show up and the user can log in.
    - Actual: When the program starts up, the user logs in, picks a patient, and then the random forest is generated.
3. Doctors(Users) will get a list of the patient's information that is causing the patient's diabetes to get worse.
    - If the random forest comes back with a prediction of false, i.e the patient’s disease progression is not good, then the information that is below scientific recommendation will be committed to a warning screen.
    - Actual: When the random forest comes back false, the patient is shown some recommendations on how to get their levels back down.

Aggregate Test Cases:

- User-friendly:
    - Input: run the program
    - Expected Output: a message for every transaction or action.
    - Actual Output: There is a message for every action

- Secure:
    - Input:

      Db = {

      doctor1, password

      doctor2, password2,

      doctor3 ,password3}

      User types in:

        Username: doctor4

        Password: password
    - Expected Output: rejected
    - Input:

      User types in:

        Username: doctor1

        Password: password
    - Expected Output: accepted
    - Actual Output: see Authorization test cases
- Accurate:
    - Input: the random forest from above
    - Expected Output: randomForest.Validate() >= 90
    - Actual Output:

      When running 5-fold validation:

      correct: 74 total: 89

      correct: 84 total: 89

      correct: 80 total: 88

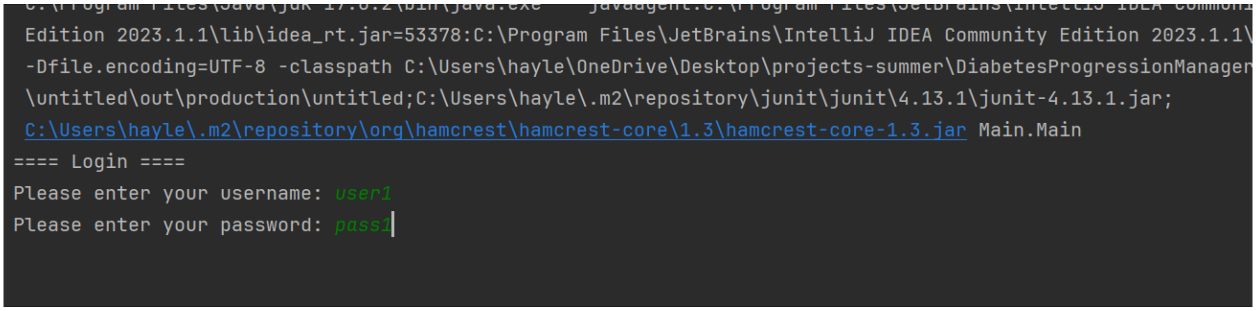
      correct: 81 total: 88

      correct: 82 total: 88

      Which averages out to 92%

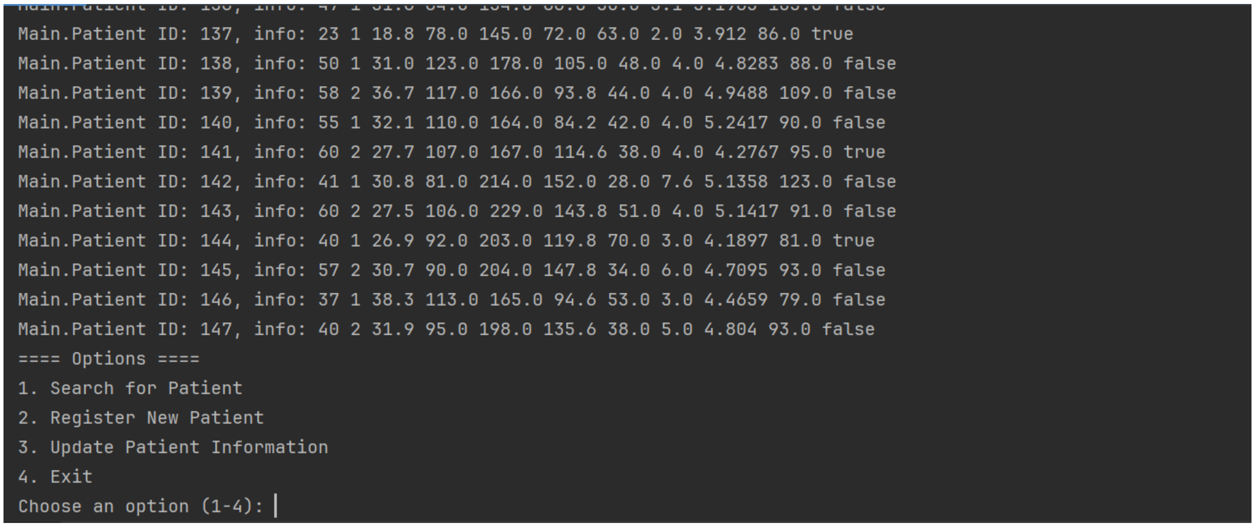
User’s Guide:

When the user first boots up the program, they will be prompted with a login screen in the console:



```
 Edition 2023.1.1\lib\idea_rt.jar=53378:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.1.1\
 -Dfile.encoding=UTF-8 -classpath C:\Users\hayle\OneDrive\Desktop\projects-summer\DiabetesProgressionManager
 \untitled\out\production\untitled;C:\Users\hayle\.m2\repository\junit\junit\4.13.1\junit-4.13.1.jar;
 C:\Users\hayle\.m2\repository\org\hamcrest\hamcrest-core\1.3\hamcrest-core-1.3.jar Main.Main
==== Login ====
Please enter your username: user1
Please enter your password: pass1
```

Please enter your username for username, and password for password.
Sample username/passwords are: user1, pass1; user2, pass2; and user3, pass3



```
Main.Patient ID: 137, info: 23 1 18.8 78.0 145.0 72.0 63.0 2.0 3.912 86.0 true
Main.Patient ID: 138, info: 50 1 31.0 123.0 178.0 105.0 48.0 4.0 4.8283 88.0 false
Main.Patient ID: 139, info: 58 2 36.7 117.0 166.0 93.8 44.0 4.0 4.9488 109.0 false
Main.Patient ID: 140, info: 55 1 32.1 110.0 164.0 84.2 42.0 4.0 5.2417 90.0 false
Main.Patient ID: 141, info: 60 2 27.7 107.0 167.0 114.6 38.0 4.0 4.2767 95.0 true
Main.Patient ID: 142, info: 41 1 30.8 81.0 214.0 152.0 28.0 7.6 5.1358 123.0 false
Main.Patient ID: 143, info: 60 2 27.5 106.0 229.0 143.8 51.0 4.0 5.1417 91.0 false
Main.Patient ID: 144, info: 40 1 26.9 92.0 203.0 119.8 70.0 3.0 4.1897 81.0 true
Main.Patient ID: 145, info: 57 2 30.7 90.0 204.0 147.8 34.0 6.0 4.7095 93.0 false
Main.Patient ID: 146, info: 37 1 38.3 113.0 165.0 94.6 53.0 3.0 4.4659 79.0 false
Main.Patient ID: 147, info: 40 2 31.9 95.0 198.0 135.6 38.0 5.0 4.804 93.0 false
==== Options ====
1. Search for Patient
2. Register New Patient
3. Update Patient Information
4. Exit
Choose an option (1-4):
```

You will then be given a list of patients with all of their information and patient IDs. As well as an options list ranging from 1 to 4.

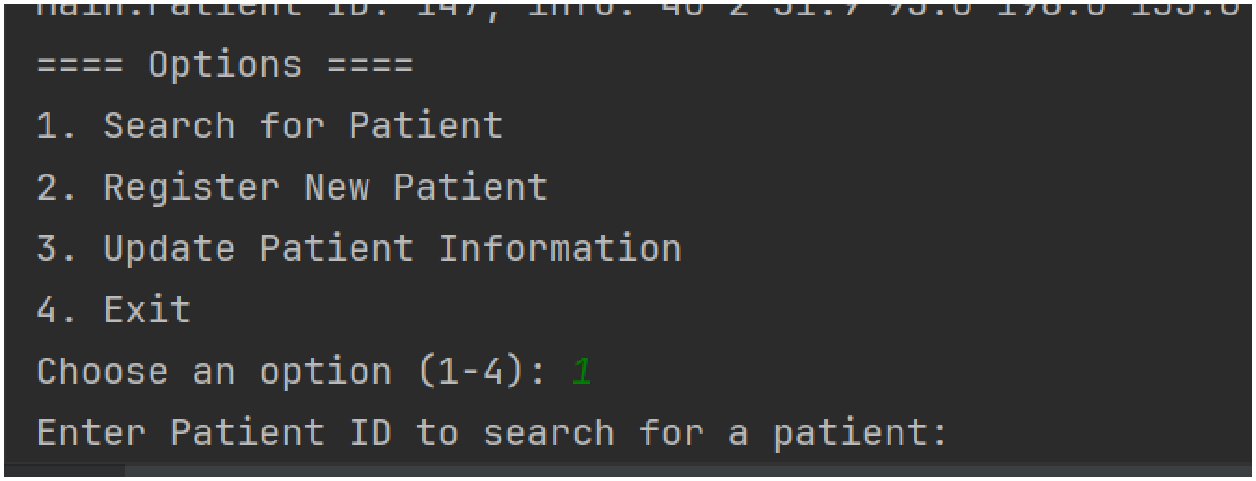
1: is to search, and then run the prediction on a patient.

2: is to register a new patient into the system

3: is to update an existing patients information

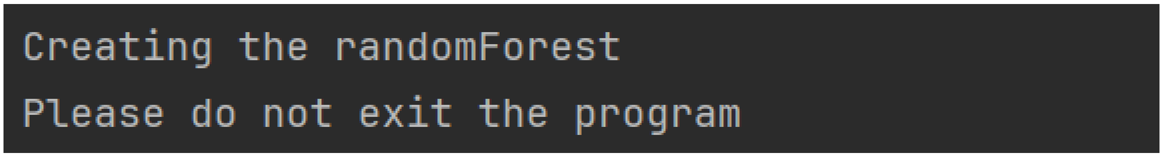
4: is to exit the program

Searching for a Patient:



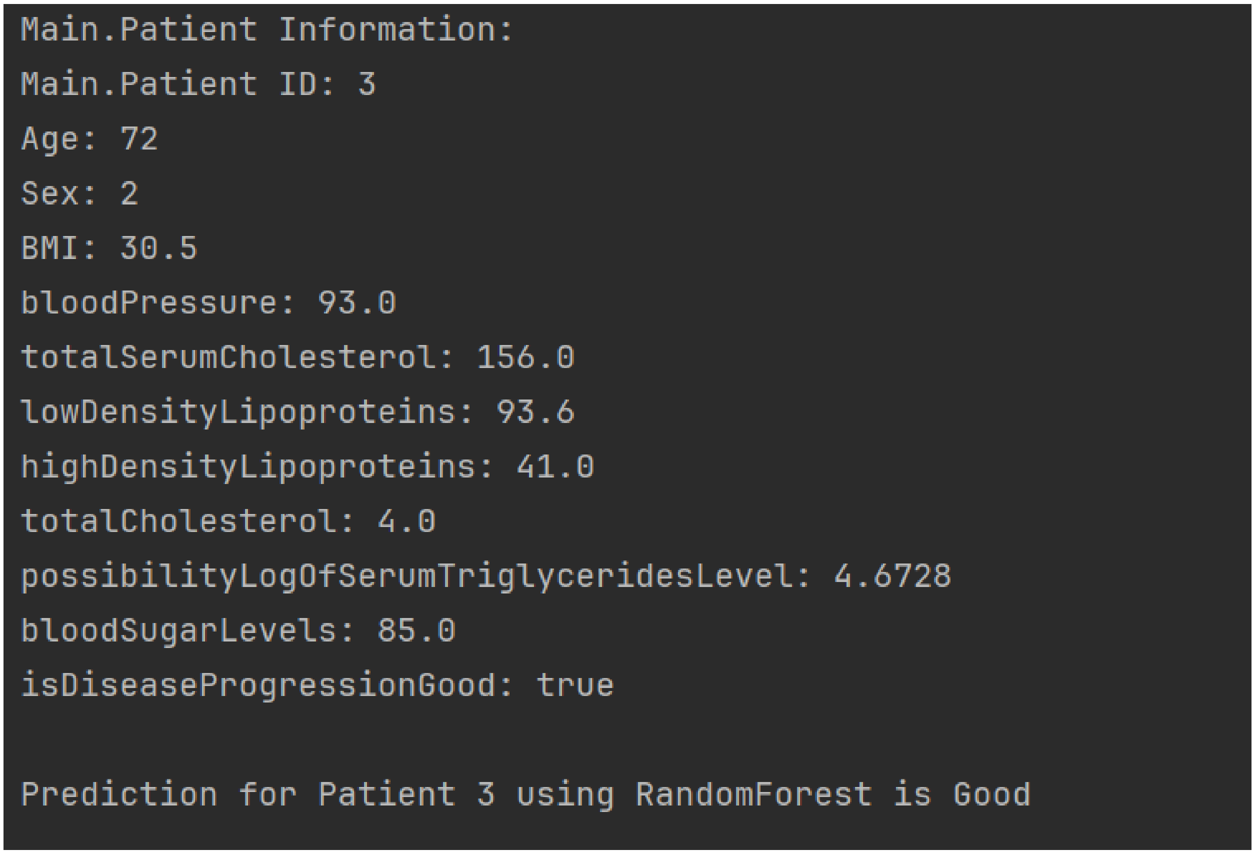
```
main.Patient ID: 147, Info: 48 2 31.7 75.0 198.0 135.0
==== Options ====
1. Search for Patient
2. Register New Patient
3. Update Patient Information
4. Exit
Choose an option (1-4): 1
Enter Patient ID to search for a patient:
```

Upon choosing 1, please input a patient id from the list of options above the =====Options====



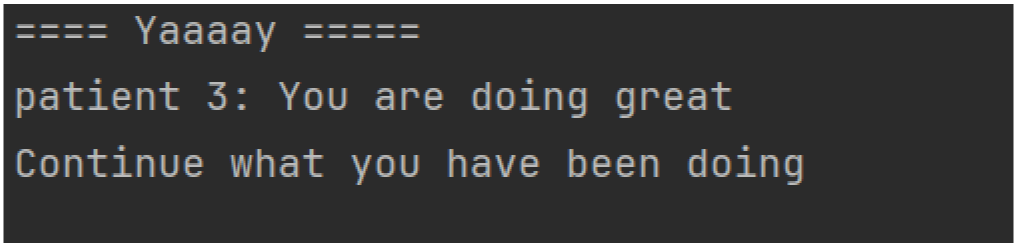
```
Creating the randomForest
Please do not exit the program
```

The random forest will be generated, please do not exit the program while this is happening.



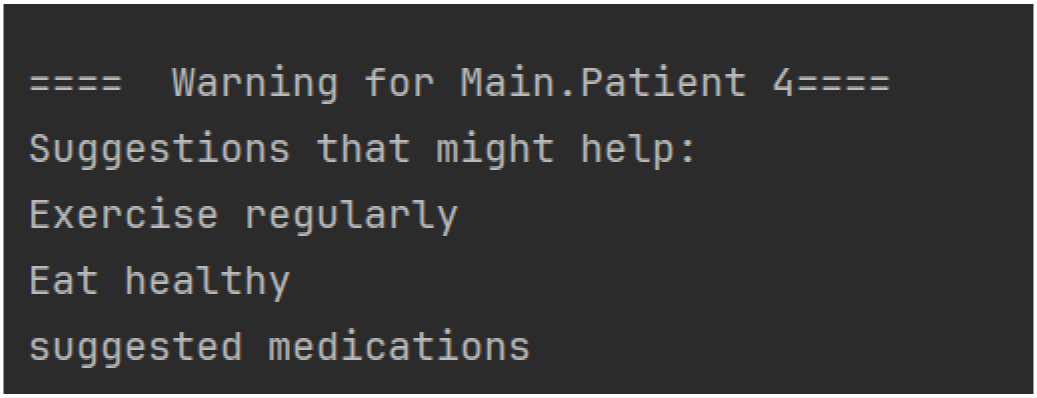
```
Main.Patient Information:
Main.Patient ID: 3
Age: 72
Sex: 2
BMI: 30.5
bloodPressure: 93.0
totalSerumCholesterol: 156.0
lowDensityLipoproteins: 93.6
highDensityLipoproteins: 41.0
totalCholesterol: 4.0
possibilityLogOfSerumTriglyceridesLevel: 4.6728
bloodSugarLevels: 85.0
isDiseaseProgressionGood: true

Prediction for Patient 3 using RandomForest is Good
```

The patient’s information will show up, and so will a prediction from the Random forest using their data



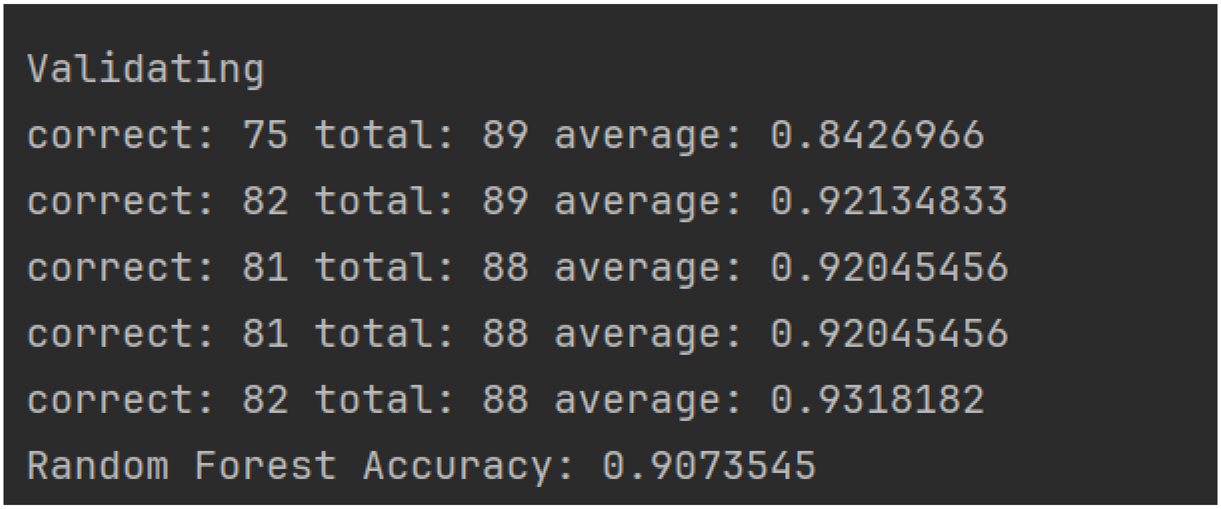
```
==== Yaaaay =====
patient 3: You are doing great
Continue what you have been doing
```

If the prediction is good, you will see a motivating message.



```
====  Warning for Main.Patient 4====
Suggestions that might help:
Exercise regularly
Eat healthy
suggested medications
```

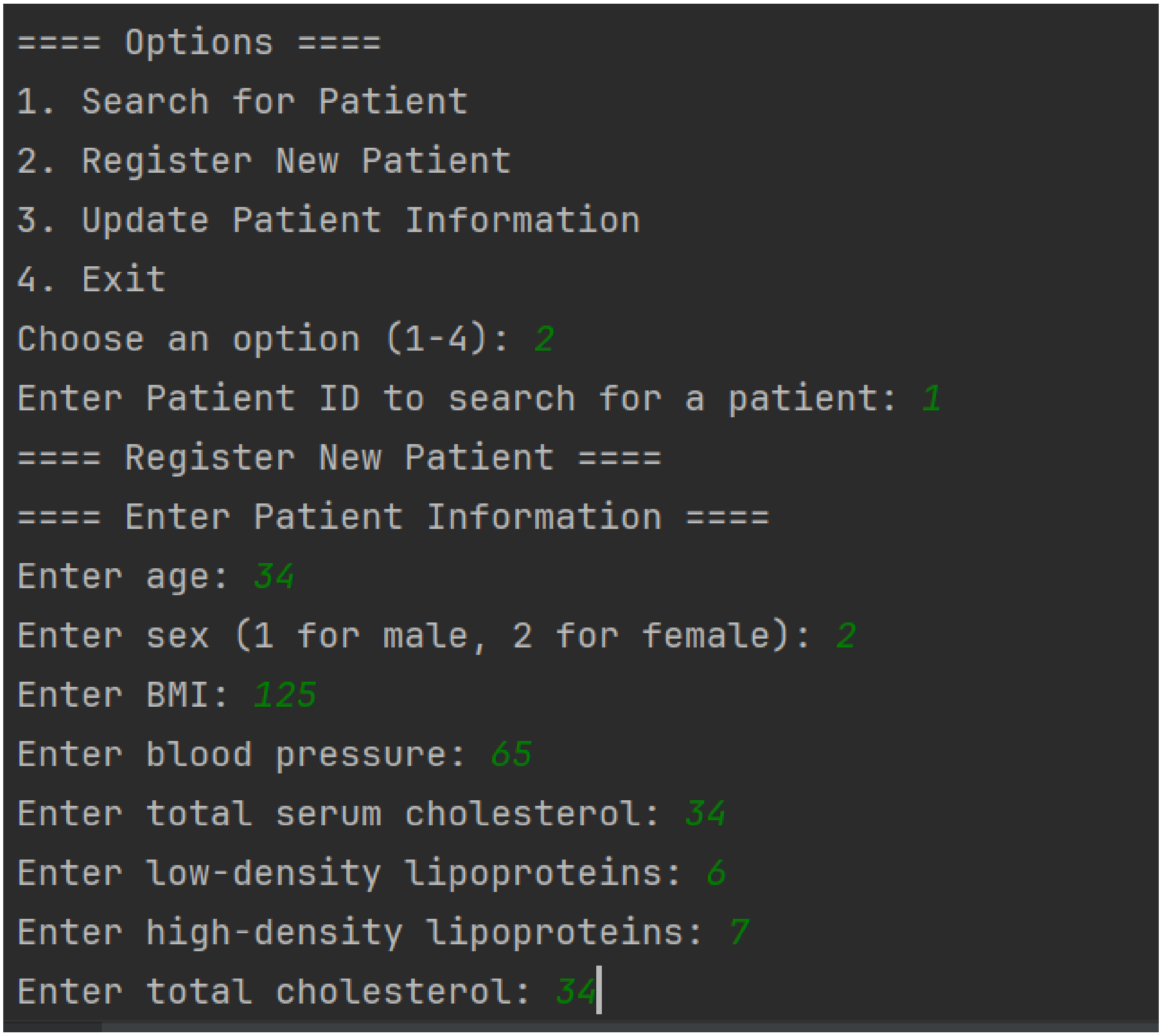
However, if the prediction is bad, you will see some tips on how to get better.



```
Validating
correct: 75 total: 89 average: 0.8426966
correct: 82 total: 89 average: 0.92134833
correct: 81 total: 88 average: 0.92045456
correct: 81 total: 88 average: 0.92045456
correct: 82 total: 88 average: 0.9318182
Random Forest Accuracy: 0.9073545
```

You will then see the random forest being validated, which will show you how accurate the tree is.

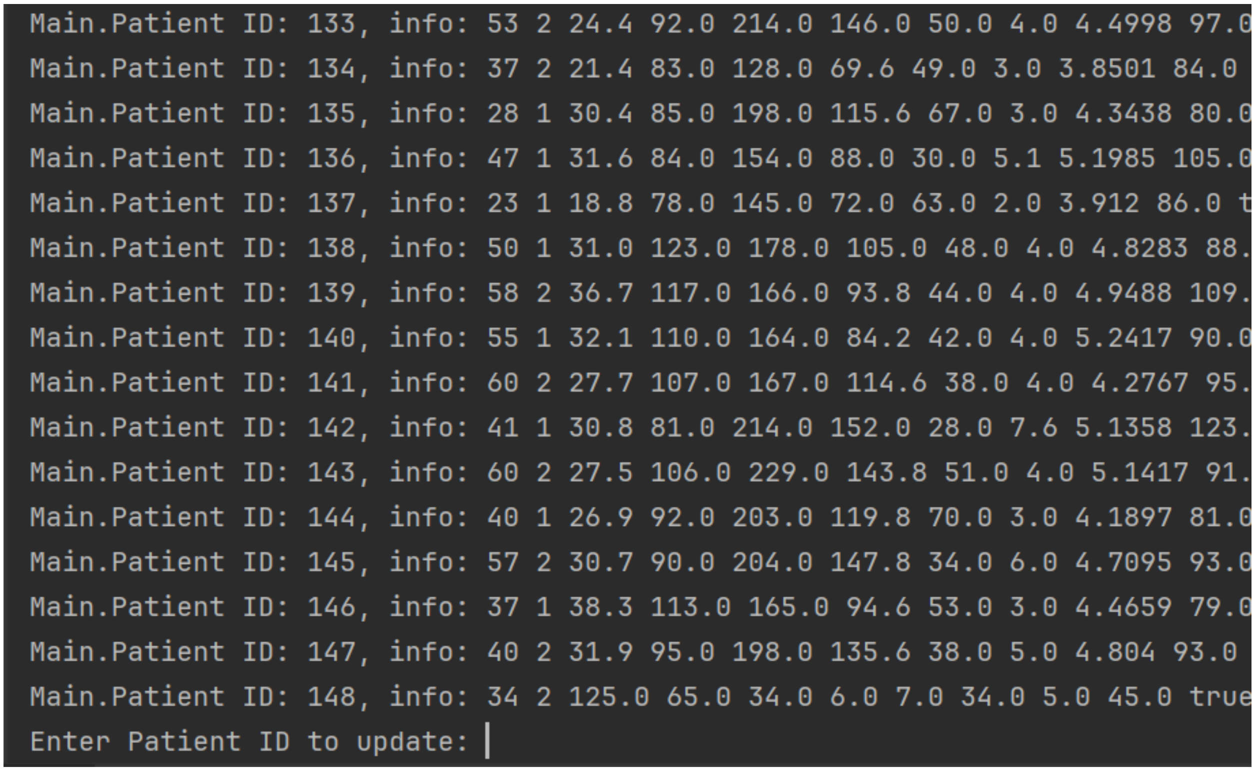
Register a New Patient:



```
==== Options ====
1. Search for Patient
2. Register New Patient
3. Update Patient Information
4. Exit
Choose an option (1-4): 2
Enter Patient ID to search for a patient: 1
==== Register New Patient ====
==== Enter Patient Information ====
Enter age: 34
Enter sex (1 for male, 2 for female): 2
Enter BMI: 125
Enter blood pressure: 65
Enter total serum cholesterol: 34
Enter low-density lipoproteins: 6
Enter high-density lipoproteins: 7
Enter total cholesterol: 34
```

When choosing 2, you will enter a new patient id to add the patient to the system, and then, one, by one, wil enter in all of the patient's health information

Update Patient Information:



```
Main.Patient ID: 133, info: 53 2 24.4 92.0 214.0 146.0 50.0 4.0 4.4998 97.0
Main.Patient ID: 134, info: 37 2 21.4 83.0 128.0 69.6 49.0 3.0 3.8501 84.0
Main.Patient ID: 135, info: 28 1 30.4 85.0 198.0 115.6 67.0 3.0 4.3438 80.0
Main.Patient ID: 136, info: 47 1 31.6 84.0 154.0 88.0 30.0 5.1 5.1985 105.0
Main.Patient ID: 137, info: 23 1 18.8 78.0 145.0 72.0 63.0 2.0 3.912 86.0 t
Main.Patient ID: 138, info: 50 1 31.0 123.0 178.0 105.0 48.0 4.0 4.8283 88.
Main.Patient ID: 139, info: 58 2 36.7 117.0 166.0 93.8 44.0 4.0 4.9488 109.
Main.Patient ID: 140, info: 55 1 32.1 110.0 164.0 84.2 42.0 4.0 5.2417 90.0
Main.Patient ID: 141, info: 60 2 27.7 107.0 167.0 114.6 38.0 4.0 4.2767 95.
Main.Patient ID: 142, info: 41 1 30.8 81.0 214.0 152.0 28.0 7.6 5.1358 123.
Main.Patient ID: 143, info: 60 2 27.5 106.0 229.0 143.8 51.0 4.0 5.1417 91.
Main.Patient ID: 144, info: 40 1 26.9 92.0 203.0 119.8 70.0 3.0 4.1897 81.0
Main.Patient ID: 145, info: 57 2 30.7 90.0 204.0 147.8 34.0 6.0 4.7095 93.0
Main.Patient ID: 146, info: 37 1 38.3 113.0 165.0 94.6 53.0 3.0 4.4659 79.0
Main.Patient ID: 147, info: 40 2 31.9 95.0 198.0 135.6 38.0 5.0 4.804 93.0
Main.Patient ID: 148, info: 34 2 125.0 65.0 34.0 6.0 7.0 34.0 5.0 45.0 true
Enter Patient ID to update: |
```

Pick a patient ID, and then you will be prompted to set new information as is the case above.

Summary of complete work and incomplete work planned for Phase I

Completed Work:

-We have a random forest with a 90-92% accuracy. Our random forest has been implemented correctly if our successful Junit tests are anything to go by.
-Users can pick patients from an existing database and can get a prediction and a warning from the program.
-Our users can log in and users outside of the system are rejected
-Our users can register new patients, update new patient information, and run the random forest on existing patients.

Incomplete Work:

-Currently, our decision tree categorical nodes do not calculate their own number of categorical features, and instead rely on a hard-coded 2. That is mainly because we only have one categorical feature that we train on.

-Our depth is also not calculated correctly. It tallies up the amount of children, rather than the actual depth of the trees.

-We don't give a list of problems that are causing the patient's diabetes progression to get worse, only some general suggestions.

-We don't have a user sign up, so we can't have new users

-We don't have a warnings screen that prints out which data is causing the progression to be bad

Planning for Version 2.0

1. We should calculate the depth correctly to ensure that there is less guesswork on what max_depth should be
2. We would add a GUI interface so the users don't have to touch the console. Many doctors would probably prefer to use a GUI like the ones they already have in their offices.
3. We might also port our work to a real database instead of keeping everything in a file or in the program itself.
4. We would add in a sign-in section so new users could be added to the system.
5. We might also give users 3 chances to get their login's correct and lock them out if they use them up, so that we could keep our data secure.

GitHub repository

https://github.com/AlexisHampton/DiabetesProgressionManager