Identifying Breast Cancer

CSCI-3351

Bytecode Boys

University of New Haven

TAGLIATELA COLLEGE OF ENGINEERING, West Haven, CT

Submitted To:

Prof Reza Sadeghi

Fall 2020

Table of Contents

Project Report: Phase – 6

Team Information

Team Name

Bytecode Boys

Team Members

1. Jake Intravaia : jintr1@unh.newhaven.edu (Team Head)
2. Bobby Chenkus : rchen5@unh.newhaven.edu (Team Member)
3. Trey Gary : tgary1@unh.newhaven.edu (Team Member)
4. Alexander Russello : aruss10@unh.newhaven.edu (Team Member)
5. Nathaniel Small : nsmal1@unh.newhaven.edu (Team Member)

Research Question

Can we use machine learning and patient data to accurately identify malignant breast cancer in patients by analyzing previously diagnosed patients?

Research Objective

The main objective of the project is to create a Neural Network capable of accurately predicting the presence of malignant breast cancer in patients using patient data.

Literature Review

It is known that detection, prevention, and treatments for different cancers are being developed. However, breast cancer is one of the most common cancers among women and the second most common cancer overall [1][2].

Breast cancer research opens the door to finding better ways to prevent, detect, and treat breast cancer. This project's analysis aims to observe which attributes are most helpful in predicting a patient's degree of malignance. General trends will also be observed to enable the selection of attributes that will yield the highest percentage of correctly guessed cases. The goal is to classify the degree of a patient's malignance. Machine learning classification methods were used to fit a function that can, to an extent, predict degree of malignancy based upon several different inputs.

About the Dataset

The dataset covers the data from breast cancer cases reported by the Oncology Institute. This set contains a total of 286 instances for analysis which is split into two separate classes. There are a total of ten attributes which includes class, age, menopause, tumor-size, inv-nodes, node-caps, deg-malig, breast, breast-quad, and irradiant.

It's also important to note that this dataset is from 1988. This dataset has been referenced many times, but was last referenced in 2005 [3]. For the purpose of this project this dataset is useful, however, a newer dataset should be used for an thorough and updated analysis of this issue.

Data Collection Methods

The data was collected by Matjaz Zwitter & Milan Soklic from the Oncology Institute. This institute is known for having datasets used for machine learning applications.

Accuracy of the Dataset

The dataset has been provided by the Oncology Institute and is published by the UCI machine learning repository. This repository is supported by the Nation Science Foundation (NSF) which supports the fundamental research and education in all of science and engineering. The process of machine learning can be classified as computer science which the NSF is the major source of federal backing. With this information, the dataset provided can be considered accurate.

Plan for Solving

Initially our dataset was a mixture of strings and numerical values, separated by commas which was not ideal for our neural network. Our team leader created a convert.py file that allowed us to convert our mixed data into all numerical values for use within our neural network models. This convert.py file and the outputted numerical value would be the base data in which we all based our models off of.

Once we had our numerical data, we could begin to create our different Neural Network models, utilizing a variety of modules from scikit learn to try and get our predictions to be as accurate as possible.

Results

Decision Tree (**Bobby Chenkus**)

Scikit-Learn's decision tree module seemed to be a good fit for this dataset due to its ability to look for trends not only within numbers, but also within categorical data. This versatile method of classification sounds like a good method to analyze many different kinds of datasets. The visualization aspect of this classification is also appealing so the decisions through each step of the "tree" can be seen.

All libraries were first imported. The variable *df* was declared and used to read the converted .py file containing the dataset's information. To ensure that the file was read properly df was output to the terminal showing the full dataset.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn import tree

df = pd.read_csv(r"C:\Users\bobby\Desktop\6_Fall 2020 Semester\Python\Final Project\breast-cancer-num-fixed.data")
df #Output the data that has been read
```

Out[1]:

	Class	Age	Menopause	Tumor-Size	INV-Nodes	Node-Caps	Breast	Breast-Quad	Irradiant	Deg-Malig
0	0	3	0	7	1	0	0	2	0	3
1	0	4	0	5	1	0	1	3	0	2
2	0	4	0	5	1	0	0	2	0	2
3	0	6	2	4	1	0	1	1	0	2
4	0	4	0	1	1	0	1	4	0	2
...	...	...	...	...	...	...	...	...	...	...
281	1	3	0	7	1	0	0	1	0	2
282	1	3	0	5	1	0	0	1	1	3
283	1	6	2	5	1	0	1	1	0	1
284	1	4	2	7	2	0	0	2	0	3
285	1	5	2	7	2	0	0	2	0	3

286 rows × 10 columns

Figure 1. Imported libraries and complete dataset

Variables X and y were declared to separate the dataset into two separate arrays of information. X.describe() was used to show some of the characteristics of the array and allowed for confirmation that Deg-Malig was dropped from the attributes. The values within variable X were output to show how the data in the array was organized.

```
In [8]: X = df.drop(columns=['Deg-Malig']) #Delete the last column of "Deg-Malig"
        y = df['Deg-Malig'] #The variable y is now the array of values in the Deg-Malig column

        X.describe() #Output characteristics of X
```

Out[8]:

	Class	Age	Menopause	Tumor-Size	INV-Nodes	Node-Caps	Breast	Breast-Quad	Irradiant
count	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000	286.000000
mean	0.297203	4.664336	0.926573	5.881119	1.524476	0.195804	0.468531	2.157343	0.237762
std	0.457828	1.011818	0.986680	2.105930	1.150635	0.397514	0.499883	1.202220	0.426459
min	0.000000	2.000000	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	4.000000	0.000000	5.000000	1.000000	0.000000	0.000000	1.000000	0.000000
50%	0.000000	5.000000	0.000000	6.000000	1.000000	0.000000	0.000000	2.000000	0.000000
75%	1.000000	5.000000	2.000000	7.000000	2.000000	0.000000	1.000000	3.000000	0.000000
max	1.000000	7.000000	2.000000	11.000000	9.000000	1.000000	1.000000	5.000000	1.000000

Figure 2. Characteristics of dataset X

```
In [9]: X.values #Showing how the data is organized in X

Out[9]: array([[0, 3, 0, ..., 0, 2, 0],
               [0, 4, 0, ..., 1, 3, 0],
               [0, 4, 0, ..., 0, 2, 0],
               ...,
               [1, 6, 2, ..., 1, 1, 0],
               [1, 4, 2, ..., 0, 2, 0],
               [1, 5, 2, ..., 0, 2, 0]], dtype=int64)
```

Figure 3. Values within the X dataset

The training and testing then began by using train_test_split from sklearn.model. The sizes of the training and testing datasets were printed to ensure that the splitting of data was done properly. Model was declared as a variable for the DecisionTreeClassifier() which would allow the training variables to be fit into the decision tree classification system. However, the training and testing data was first fit to a scalar function to create a number scaling format which would allow for properly scaled results.

```
In [4]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

        print('Data_train size:',X_train.shape) #Showing the sizes of our training and testing data sets
        print('Data_test size:',X_test.shape)

        Data_train size: (228, 9)
        Data_test size: (58, 9)

In [5]: model = DecisionTreeClassifier() #Defining a DecisionTreeClassifier variable
        scaler = StandardScaler() # Defining a StandardScaler variable

        scaler.fit(X_train) # Using scaler fit function to calculate best scale
        X_test = scaler.transform(X_test) # Scaling our test values

        model.fit(X_train, y_train) #Fit the training variables
        predictions= model.predict(X_test) #Create predictions based upon our tested X values
```

Figure 4. Train and Test Data Split & Fitting to Standard Scale and Decision Tree

The decision tree was visualized by using the below functions. The labels variable allowed the created tree to create titles for each box and make decisions based on the values of these labels or attributes. Since the scale has been standardized number comparison has been best fit for the numbers within this dataset.

```
In [13]: labels = ['Class', 'Age', 'Menopause', 'Tumor-Size', 'INV-Nodes', 'Node-Caps', 'Breast', 'Breast-Quad', 'Irradiant']
         #Labels is used for the feature names

         fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (12,8), dpi=2540) #This sets up how the figure will be formatted

         tree.plot_tree(model,
                        feature_names = labels,
                        filled = True)
         #tree.plot_tree allows for my data to printed out by using model (Our DecisonTreeClassifer) and labels

         fig.savefig('Decision_Tree.png') #Print out the results as a .png file
```

Figure 5. Code for Decision Tree Visualization

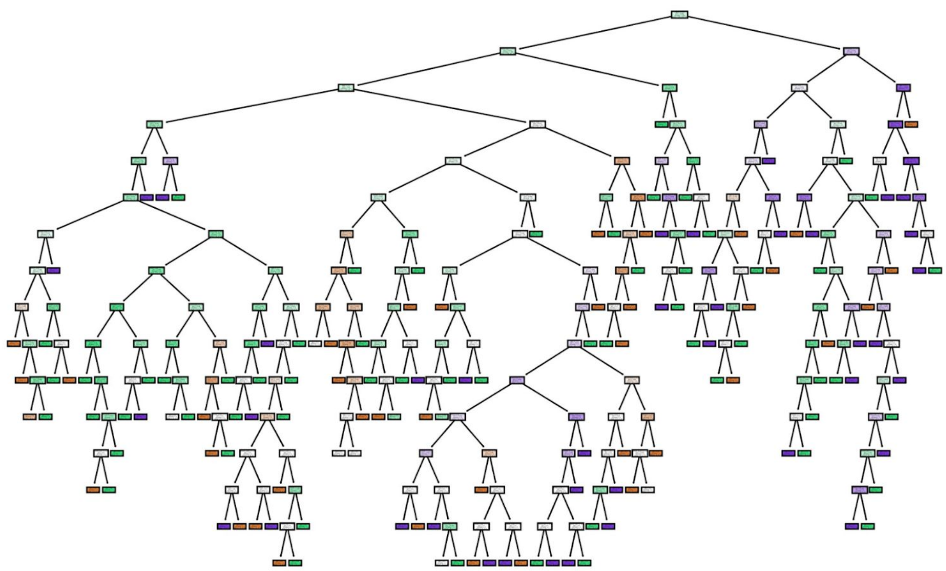

Figure 6. Decision Tree Visualization

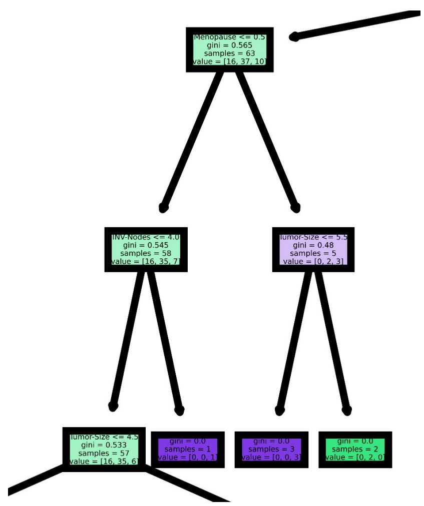

Figure 7. Zoomed in Image of Decisions within Tree

As shown in Figure 8, this method used the accuracy score to show how accurate the predictions made from the training data was when compared to the testing set. The average accuracy score of the decision tree was around 35 - 45%. These numbers seemed to be slightly higher on average when compared to other observed modules. The decision tree worked okay for the analysis of this dataset, however it is not reliable in making consistently accurate predictions. A different analysis should be looked at to determine whether the degree of malignancy can be determined from this dataset's attributes.

```
In [6]: score = accuracy_score(y_test, predictions) #Determine the score that our predictions get
        score #Output the score

Out[6]: 0.41379310344827586
```

Figure 8. Accuracy Score of Decision Tree

MLP Classifier (**Jake Intravaia**)

Scikit-Learn's MLPClassifier module was perfect for the kind of data we had to process. The classifier takes in several attributes and data, and based on that data outputs a certain classification. This works very well for our dataset considering we need to classify the degree of malignance a tumor has based on several attributes of the tumor.

To begin, I imported the data from the text file into python lists that numpy can utilize.

```python
def getData(file):
    with open(file) as f:
        data = f.readlines()
    data = [x.strip() for x in data] # Get rid of newline chars
    data = [x.split(",") for x in data] # Split data by comma
    return data
```

Figure 9. Reading Content of the File

Then, just to ensure the integrity of the data, I created a function that printed our data in a human-readable format.

```python
# Prints our data in a neat format
def printData(data):
    for x in data:
        print("Reccurence events: " + x[0])
        print("Age: " + x[1])
        print("Menopause: " + x[2])
        print("Tumor size: " + x[3])
        print("Inv-nodes: " + x[4])
        print("Node-caps: " + x[5])
        print("Breast: " + x[6])
        print("Breast-quad: " + x[7])
        print("Irradiat: " + x[8])
        print("Deg-malig: " + x[9])
```

Figure 10. Printing Data to Verify Integrity

After ensuring our datas integrity and that it has been imported correctly into lists, I began to initialize our variables by first creating a python list, then converting it to a numpy array.

```python
bd = getData("breast-cancer-num-fixed.data") # Formatting our fixed data into a python list
breastData = np.asarray(bd) # Transforming python list into numpy array
```

Figure 11. Converting Data to a Numpy Array

Then it was time to import our input data (**X**) and the value we are trying to predict (**y**). Once imported, I then split our data into test data and training data. To ensure our values are scaled for use in the MLPClassifier module, I used scikit-learns built in scaler function, that scaled our data into usable numerical data.

```python
X = breastData[:, 0:9] # Selecting input values

y = breastData[:, 9:10] # Selecting our output values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

Figure 12. MLP Classifier Train and Test Data Split

After initializing all our data and creating a training data set and a test data set, it was time to create our MLPClassifier and fit our data.

```python
mlp = MLPClassifier(hidden_layer_sizes=(10,10,10,10), max_iter=50, solver='sgd',

mlp.fit(X_train.astype(float), y_train.astype(float).ravel()) # Fit function tr

predictions = mlp.predict(X_test) # Our prediction output
```

Figure 13. Fitting Data Using MLPClassifier

All that was left to do was to print our confusion matrix and classification report to see how accurate our models predictions were.

```python
print(confusion_matrix(y_test.astype(float), predictions)) # Pri
print(classification_report(y_test.astype(float), predictions))
```

Figure 14. Printing Accuracy of MLP Classifier

Random Forest Tree (**Trey Gary & Nathaniel Small**)

Imported some libraries here as a start.

```python
In [1]: # import libraries we need
import pandas as pd
import numpy as np

# all jupyter to display multiple outputs per cell
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Figure 15. Imported libraries

Set the variable df to read in the data file and organize the specified variables through a categorical means. Output also shown below.

```python
In [2]: # read in the data file, giving names to the columns, and specifying that these variables are categorical
colnames = ['rec_events','age_groups','menopause','tumor_size','inv_nodes','node_caps','deg_malig','breast','breast_quad','irradi
df = pd.read_csv('breast-cancer.data', header=None, names=colnames,
                 dtype={'menopause':'category','breast_quad':'category'} )
df
```

Out[2]:

	rec_events	age_groups	menopause	tumor_size	inv_nodes	node_caps	deg_malig	breast	breast_quad	irradiat
0	no-recurrence-events	30-39	premeno	30-34	0-2	no	3	left	left_low	no
1	no-recurrence-events	40-49	premeno	20-24	0-2	no	2	right	right_up	no
2	no-recurrence-events	40-49	premeno	20-24	0-2	no	2	left	left_low	no
3	no-recurrence-events	60-69	ge40	15-19	0-2	no	2	right	left_up	no
4	no-recurrence-events	40-49	premeno	0-4	0-2	no	2	right	right_low	no
...	...	...	...	...	...	...	...	...	...	...
281	recurrence-events	30-39	premeno	30-34	0-2	no	2	left	left_up	no
282	recurrence-events	30-39	premeno	20-24	0-2	no	3	left	left_up	yes
283	recurrence-events	60-69	ge40	20-24	0-2	no	1	right	left_up	no
284	recurrence-events	40-49	ge40	30-34	3-5	no	3	left	left_low	no
285	recurrence-events	50-59	ge40	30-34	3-5	no	3	left	left_low	no

286 rows × 10 columns

Figure 16. Reading in the data file and labeling w/ complete data set

Here is where the data was mapped together into columns for a better visualization and a simple hot encoding method. Also contains a categorized output.

In [4]:
```python
rec_events = {"recurrence-events" : 1, "no-recurrence-events" : 0}
age_groups = {"10-19" : 1, "20-29" : 2, "30-39" : 3, "40-49" : 4, "50-59" : 5, "60-69" : 6, "70-79" : 7, "80-89" : 8, "90-99" : 9
#menopause = {"premeno" : 0, "lt40" : 1, "ge40" : 2}
tumor_size = {"0-4" : 1, "5-9" : 2, "10-14" : 3, "15-19" : 4, "20-24" : 5, "25-29" : 6, "30-34" : 7, "35-39" : 8, "40-44" : 9, "4
inv_nodes = {"0-2" : 1, "3-5" : 2, "6-8" : 3, "9-11" : 4, "12-14" : 5, "15-17" : 6, "18-20" : 7, "21-23" : 8, "24-26" : 9, "27-29
#deg_malig = {"1" : 1, "2" : 2, "3" : 3}
#breast_quad = {"?" : 0, "left_up" : 1, "left_low" : 2, "right_up" : 3, "right_low" : 4, "central" : 5}
yesno = {'yes':1,'no':0,'?':0}

df_num = df

df['rec_events'] = df['rec_events'].map(rec_events)
df['age_groups'] = df['age_groups'].map(age_groups)
df['tumor_size'] = df['tumor_size'].map(tumor_size)
df['inv_nodes'] = df['inv_nodes'].map(inv_nodes)
df['node_caps'] = df['node_caps'].map(yesno) # yes is a 1, others 0
df['deg_malig'] = pd.to_numeric(df['deg_malig'])
df['breast'] = df['breast'].map({'right':1,'left':0}) # right is a 1, others 0
df['irradiat'] = df['irradiat'].map(yesno)

df.info()
df = pd.get_dummies(df) # one-hot encoding of the categorical variables.

df
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 286 entries, 0 to 285
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   rec_events   286 non-null    int64
 1   age_groups   286 non-null    int64
 2   menopause    286 non-null    category
 3   tumor_size   286 non-null    int64
 4   inv_nodes    286 non-null    int64
 5   node_caps    286 non-null    int64
 6   deg_malig    286 non-null    int64
 7   breast       286 non-null    int64
 8   breast_quad  286 non-null    category
 9   irradiat     286 non-null    int64
dtypes: category(2), int64(8)
memory usage: 18.9 KB
```

Out[4]:

	rec_events	age_groups	tumor_size	inv_nodes	node_caps	deg_malig	breast	irradiat	menopause_ge40	menopause_lt40	menopause_premeno	breast_
0	0	3	7	1	0	3	0	0	0	0	1	
1	0	4	5	1	0	2	1	0	0	0	1	
2	0	4	5	1	0	2	0	0	0	0	1	
3	0	6	4	1	0	2	1	0	1	0	0	
4	0	4	1	1	0	2	1	0	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	
281	1	3	7	1	0	2	0	0	0	0	1	
282	1	3	5	1	0	3	0	1	0	0	1	
283	1	6	5	1	0	1	1	0	1	0	0	
284	1	4	7	2	0	3	0	0	1	0	0	
285	1	5	7	2	0	3	0	0	1	0	0	

286 rows × 17 columns

Figure 17. Categorizing the data w/output

Imported more libraries such as sklearn and train. Similarly to above, rhe training and testing then began by using train_test_split from sklearn.model. The sizes of the training and testing datasets were printed to ensure that the splitting of data was done properly.

```
In [5]: # https://towardsdatascience.com/understanding-random-forest-58381e0602d2
        # https://towardsdatascience.com/random-forest-in-python-24d0893d51c0

        # Labels are the values we want to predict
        labels = np.array(df['rec_events'])
        # Remove the labels from the features
        features = df.drop(columns='rec_events')
        # Saving feature names for later use
        feature_list = list(features.columns)
        # Convert to numpy array
        features = np.array(features)
```

```
In [6]: # Using Skicit-learn to split data into training and testing sets
        from sklearn.model_selection import train_test_split
        # Split the data into training and testing sets. random_state is the seed for the rng
        train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.1, random_state = 42)

        print('Training Features Shape:', train_features.shape)
        print('Training Labels Shape:', train_labels.shape)
        print('Testing Features Shape:', test_features.shape)
        print('Testing Labels Shape:', test_labels.shape)

        Training Features Shape: (257, 16)
        Training Labels Shape: (257,)
        Testing Features Shape: (29, 16)
        Testing Labels Shape: (29,)
```

Figure 18. Train and Test Data Split with labels

Mean Absolute Error for Random Forest Model: 0.356
Accuracy computed using geometric mean on test data: 55.312%

```
In [7]: from scipy.stats.mstats import gmean

        # The baseline prediction is the historical average for recurrence.
        p = np.mean(train_labels)
        print(f'Historically, {p*100:0.1f}% of women in the training set with breast cancer have recurrence events')
        baseline_preds = np.full(test_labels.shape, p)

        # Baseline errors, and display average baseline error
        baseline_errors = abs(baseline_preds - test_labels)
        print(f'Average baseline error in predicting test data set: {np.mean(baseline_errors):0.3f}')
        print(f'Accuracy computed using geometric mean on test data: {gmean(1-baseline_errors)*100:0.3f}%')

        Historically, 28.4% of women in the training set with breast cancer have recurrence events
        Average baseline error in predicting test data set: 0.463
        Accuracy computed using geometric mean on test data: 48.837%
```

```
In [8]: # Import the model we are using
        from sklearn.ensemble import RandomForestRegressor
        # Instantiate model with 1000 decision trees
        rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
        # Train the model on training data
        rf.fit(train_features, train_labels);
```

```
In [9]: # Use the forest's predict method on the test data
        predictions = rf.predict(test_features)
        # these represent the model's predicted likelihood of the cancer recurring.
        predictions

        # Calculate the absolute errors
        errors = abs(predictions - test_labels)
        # Print out the mean absolute error (mae)
        print(f'Mean Absolute Error for Random Forest Model: {np.mean(errors):0.3f}')
        print(f'Accuracy computed using geometric mean on test data: {gmean(1-errors)*100:0.3f}%')
```

Figure 19. Accuracy scores

Random forest was configured and displayed using more imported libraries, and functions that pulled one tree from the forest.

```
In [11]: # Import tools needed for visualization
         from sklearn.tree import plot_tree
         import matplotlib.pyplot as plt

         # Pull out one tree from the forest
         tree = rf.estimators_[5]

         # Plot image
         plt.figure(figsize=[100,100])
         plot_tree(tree)
         plt.savefig('tree5.png')
```

Figure 20. Code for display Random Forest

Almost similar representation of tree diagram

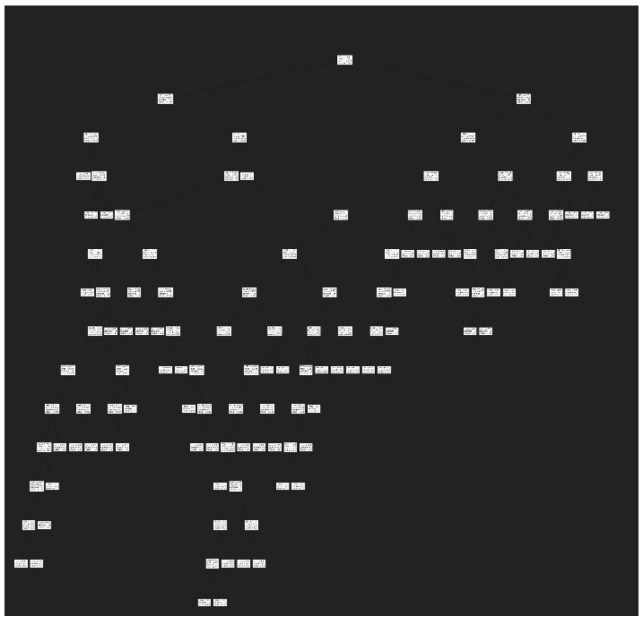

Figure 21. Random Forest Image

References

1. https://www.wcrf.org/dietandcancer/cancer-trends/breast-cancer-statistics
2. https://www.bcrf.org/breast-cancer-statistics-and-resources
3. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer
4. https://towardsdatascience.com/understanding-random-forest-58381e0602d2
5. https://towardsdatascience.com/random-forest-in-python-24d0893d51c0

GitHub

1. https://github.com/tgary1/Random-Forest
2. https://github.com/BobbyChenkus
3. https://github.com/jakeintravaia/breast-cancer-classifier
4. https://github.com/aruss77/Breast_Cancer