

Random Forest Classification Example

Decision Tree Example

On IOS

Contents

[Introduction 2](#_Toc516844750)

[Learning outcomes 2](#_Toc516844751)

[Random Forest 3](#_Toc516844752)

[Decision Tree: 3](#_Toc516844753)

[From Decision Tree to Random Forest 7](#_Toc516844754)

[Random Forest through Example 11](#_Toc516844755)

[Converting the model to Coreml 14](#_Toc516844756)

[Creating an IOS APP 16](#_Toc516844757)

# Introduction

In this chapter we are about to learn introduction of random forest algorithm and why we use it. And also an example demonstrating the use of Random forest algorithm in prediction of cancer type.

And we are going to show you step by step procedure in creation of cancer prediction model using random forest classification algorithm and also the procedure to port that into an IOS device app.

# Learning outcomes

At the completion of this chapter you will be able to understand what a random forest algorithm is and when to use it. And also how to implement this in Python using Sci-Kit Learn and Pandas libraries. And also you will be able to convert any Sci-Kit Learn model to Core-ML model for the use in IOS devices. And also what is Core-ML library in IOS and how to use it to get the inference from your models.

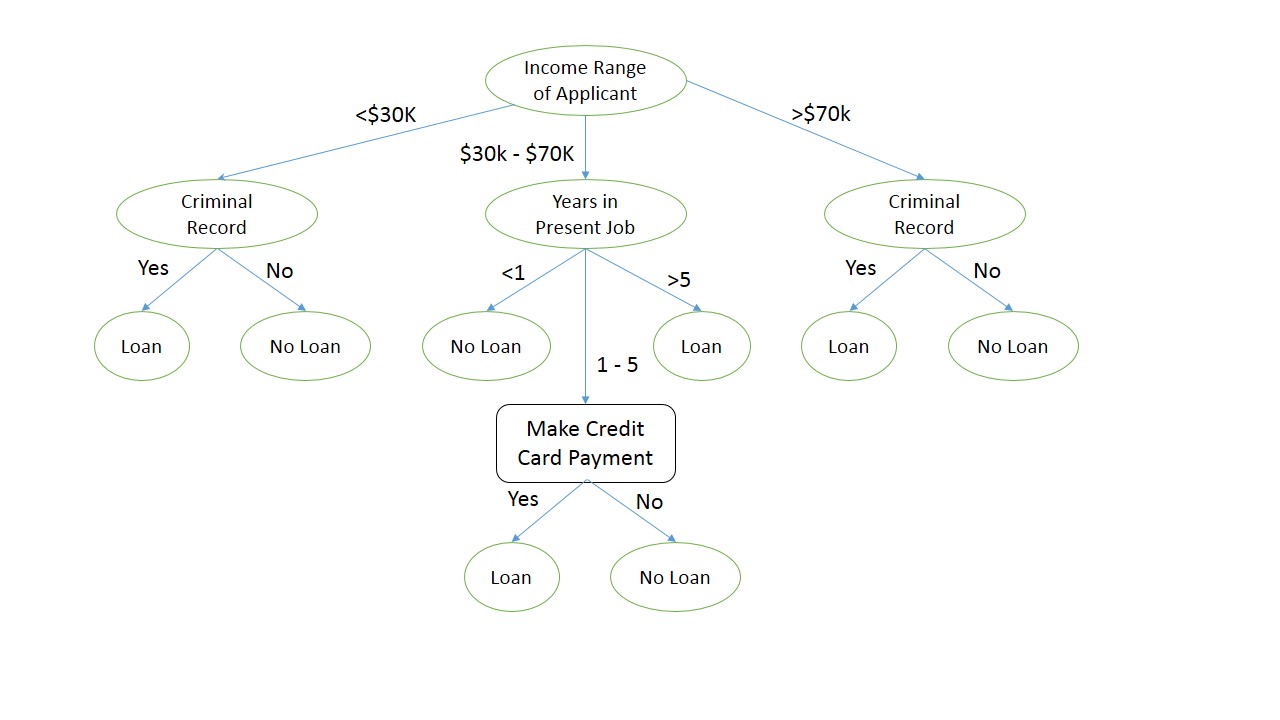
# Random Forest

When learning a technical concept, I find it’s better to start with a high-level overview and work your way down into the details rather than starting at the bottom and getting immediately lost. Along those lines, this post will use an intuitive example to provide a conceptual framework of the random forest, a powerful machine learning algorithm. After getting a basic idea down, I move on to a simple implementation to see how the technique works and if it will be useful to me before finally working out the details by digging deep into the theory.

## Decision Tree:

To understand the random forest model, we must first learn about the decision tree, the basic building block of a random forest. We all use decision trees in our daily life, and even if you don’t know it by that name, I’m sure you’ll recognize the process. To illustrate the concept let me take an example.

I know we all approached bank or a bank website at least once to get a loan. If not, let us go to any banking website and go through the loan eligibility check. There you can come through different types of questions while helps for a bank to consider whether or not to offer you a loan they often go through a sequential list of questions to figure out if it is safe to give said loan to an individual. Those questions can start as simple as what kind of income does the person have? If it is between $30–70k they move on to the next question. How long have they held their current job? If 1–5 years it leads to their next question of do they make their credit card payments? If yes then they offer the Loan and if no they do not. This process at its most basic form is a Decision Tree.



A decision tree is a largely used non-parametric effective machine learning modeling technique for regression and classification problems. To find solutions a decision tree makes sequential, hierarchical decision about the outcomes variable based on the predictor data.

So what does all that mean?

Hierarchical means the model is defined by a series of questions that lead to a class label or a value when applied to any observation. Once set up the model acts like a protocol in a series of “if this occurs then this occurs” conditions that produce a specific result from the input data.

A Non-parametric method means that there are no underlying assumptions about the distribution of the errors or the data. It basically means that the model is constructed based on the observed data.

Decision tree models where the target variable uses a discrete set of values are classified as Classification Trees. In these trees, each node, or leaf, represent class labels while the branches represent conjunctions of features leading to class labels. A decision tree where the target variable takes a continuous value, usually numbers, are called Regression Trees. The two types are commonly referred to together at CART (Classification and Regression Tree).

Each CART model is a case of a Directed Acyclic Graph. These graphs have nodes representing decision points about the main variable given the predictor and edges are the connections between the nodes. In the Loan scenario above the $30-$7ok would be an edge and the “Years Present in Job” are nodes.

As the goal of a decision tree is that it makes the optimal choice at the end of each node it needs an algorithm that is capable of doing just that. That algorithm is known as Hunt’s algorithm, which is both greedy, and recursive. Greedy meaning that at step it makes the most optimal decision and recursive meaning it splits the larger question into smaller questions and resolves them the same way. The decision to split at each node is made according to the metric called purity. A node is 100% impure when a node is split evenly 50/50 and 100% pure when all of its data belongs to a single class.

In order to optimize our model we need to reach maximum purity and avoid impurity. To measure this we use the Gini impurity, which measures how often a randomly chosen element is labeled incorrectly if it was randomly labeled according to distribution. It is calculated by adding the probability, pi, of an item with the label, i, being chosen multiplied by the times the probability (1–pi) of a mistake categorizing the time. Our goal is to have it reach 0 where it will be minimally impure and maximally pure falling into one category.

The other metric used is information gain, which is used to decide what feature to split at each step in the tree. This is calculated in the following way in a nicely laid out equation made by Wikipedia,

Information Gain = Entropy(parent) - Weighted Sum of Entropy(Children).

While this is a great model it does present a large problem by resulting in a model that only stops when all the information is in a single class or attribute. At the expense of bias the variance for this model is massive and will definitely lead to over fitting. “Decision-tree learners can create over-complex trees that do not generalize well from the training data.” So how do web combat this. We can either set a maximum depth of the decision tree (i.e. how many nodes deep it will go (the Loan Tree above has a depth of 3) and/or an alternative is to specify a minimum number of data points needed to make a split each decision.

What are the other disadvantages does a Decision Tree have: It is locally optimized using a greedy algorithm where we cannot guarantee a return to the globally optimal decision tree. It is an incredibly biased model if a single class takes unless a dataset is balanced before putting it in a tree.

While there are disadvantages there are many advantages to Decision Trees.

They are incredibly simple to understand to understand due to their visual representation, they require very little data, can handle qualitative and quantitative data, it can be validated using statistical sets, it can handle large amounts of data and it is quite computationally inexpensive.

That is basically the entire high-level concept of a decision tree: a flowchart of questions leading to a prediction. Now, we take the mighty leap from a single decision tree to a random forest!

## From Decision Tree to Random Forest

Let me get you another example, if I ask you to guess who will be your next city Mayor, will your city sitting mayor will be elected next time. Just give a pause and think before proceeding to the next lines.

Yeah, did you get the answer?

Let you recap how you get in to that answer. Let me explain how I guess. First we will go through different sequence of questions.

* How many parties are there? And what are they?
* Who is the current Mayor?
* How he is performing.
* Which party he is from?
* Which party is in ruling in Country / State?
* Is he belongs to ruling party?
* Which caste people are majority in your city?
* How many nominations are there?

Like this many question will come to our mind and we will give different weightages to them.

My prediction for the above question may be probably wrong. And I hate to break it to you, but so is yours. There are too many factors to take into account, and chances are, each individual guess will be different. Every person comes to the problem with different background knowledge and may interpret the exact same answer to a question entirely differently. In technical terms, the predictions have variance because they will be widely spread around the right answer. Now, what if we take predictions from hundreds or thousands of individuals, some of which are “**yes he may come**” and some of which are “**No, some other person will come**”, and decided to average them together? Well, congratulations, we have created a random forest! The fundamental idea behind a random forest is to combine many decision trees into a single model. Individually, predictions made by decision trees (or humans) may not be accurate, but combined together, the predictions will be closer to the mark on average.

Why exactly is a random forest better than a single decision tree? We can think about it terms of having hundreds of humans make estimates for the above problem: by pooling predictions, we can incorporate much more knowledge than from any one individual. Each individual brings their own background experience and information sources to the problem. Some people may swear by Exit Polls, while others will only look at Social Media. Perhaps some person may be a political analyst / critic.

If we only ask one individual, we would only take advantage of their limited scope of information, but by combining everyone’s predictions together, our net of information is much greater. Furthermore, the more diverse each person’s source of information, the more robust the random forest is because it will not be swayed by a single anomalous data source. If someone goes rogue and starts making predictions bad things and everyone relied on him, then our entire model would be worthless. If instead, individuals in our ‘forest’ use a number of different sources, then our model will not be greatly affected by a single source and we can continue to make reasonable predictions.

Why the name ‘random forest?’ Well, much as people might rely on different sources to make a prediction, each decision tree in the forest considers a random subset of features when forming questions and only has access to a random set of the training data points. This increases diversity in the forest leading to more robust overall predictions and the name ‘random forest.’ When it comes time to make a prediction, the random forest takes an average of all the individual decision tree estimates. (This is the case for a regression task, such as our problem where we are predicting a continuous value of number of votes. The other class of problems is known as classification, where the targets are a discrete class label such as Win or Lose. In that case, the random forest will take a majority vote for the predicted class). With that in mind, we now have down all the conceptual parts of the random forest!

# Random Forest through Example

Now are going to see how to implement the Random forest algorithm in python using Sci-Kit Learn. For the prediction of type of cancer.

For this use case we will use Breast Cancer dataset available in the UCAI repository.

Just think a dataset as an excel sheet having rows and columns with headings.

Our data set is having the below columns / Attributes

**Attribute Information:**

1. ID number   
   2) Diagnosis (M = malignant, B = benign)    
   Ten real-valued features are computed for each cell nucleus:   
   a) radius (mean of distances from center to points on the perimeter)   
   b) texture (standard deviation of gray-scale values)   
   c) perimeter   
   d) area   
   e) smoothness (local variation in radius lengths)   
   f) compactness (perimeter^2 / area - 1.0)   
   g) concavity (severity of concave portions of the contour)   
   h) concave points (number of concave portions of the contour)   
   i) symmetry   
   j) fractal dimension ("coastline approximation" - 1)

If you see the attributes you can find the column name as Diagnosis (M = malignant, B = benign) in this it is mentioned that, the current record / information is belongs to Malignant Cancer Patient or Benign Cancer patient.

So, this serves as the key attribute to predict and all other attributes will serve as input attributes.

For this implementation you require anaconda and python version 3+.

Now through anaconda create an environment named as eg: random forest.

And install all the below packages using pip commands.

* Pandas
* scikit-learn
* numpy

Now write the below lines of code.

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

import sklearn.datasets as ds

import sklearn

The above lines of code will import all the necessary packages for our use.

dataset = ds.load\_breast\_cancer()

This will load the breast cancer dataset in to the dataset variable.

cancerdata = pd.DataFrame(dataset.data)

This will create a data frame from the data present in the dataset. Assume dataset as n excel sheet with rows and columns with column headings.

cancerdata.columns = dataset.feature\_names

***This will put the column headings to the columns.***

for i in range(0,len(dataset.feature\_names)):

if ['mean concave points', 'mean area', 'mean radius', 'mean perimeter', 'mean concavity'].\

\_\_contains\_\_(dataset.feature\_names[i]):

continue

else:

cancerdata = cancerdata.drop(dataset.feature\_names[i], axis=1)

The above lines will delete all the other columns other than

* mean concave points
* mean area
* mean radius
* mean perimeter
* mean concavity

I done this because I don’t want to take this in to account.

cancerdata.to\_csv("myfile.csv")

This line will save the data to a CSV file. You can open it and see in excel. What present in the dataset.

cancer\_types = dataset.target\_names

If you see excel you will know that diagnosis will contain the value as 0 or 1. 0 from malignant and 1 for benign.

To change this to the real names we are writing this code.

cancer\_names = []

//getting all the corresponding cancer types with name [string] format.

for i in range(len(dataset.target)):

cancer\_names.append(cancer\_types[dataset.target[i]])

x\_train, x\_test, y\_train, y\_test = sklearn.model\_selection.train\_test\_split(cancerdata,cancer\_names,test\_size=0.3, random\_state=5)

This line of code will split the dataset in to two as one for training and one for testing and saved in the corresponding variables.

classifier = RandomForestClassifier()

This line will create a classifier.

classifier.fit(x\_train, y\_train)

This will feed the training data and train the model.

//testing the model with test data

print(classifier.predict(x\_test))

The above line will print the predicted cancer type for the testing data to console.

## Converting the model to Coreml

I think for the previous chapter you may get a brief introduction on what is coreml.

To do this you must need a MAC machine.

Let me explain using an example, let us assume you are from France and you know only French / English. Just assume you came to India for some tour on a vacation. And you went to local hotel there the bearer offered you a menu card which was written in some local language.

Now what you will do?

Let me guess, you will ask the bearer to spell out the items in English or any other customer / your tour guide who came with you or you can simple scan the images in Google translate to know what it is.

At last what, my point is you need a translator to understand.

That’s it. That’s all the work of the below code. It will convert the Sci-Kit learn format to Core-Ml format.

//converting the fitted model to a coremlmodel file

model = coremltools.converters.sklearn.convert(classifier, input\_features=list(cancerdata.columns.values), output\_feature\_names='typeofcancer')

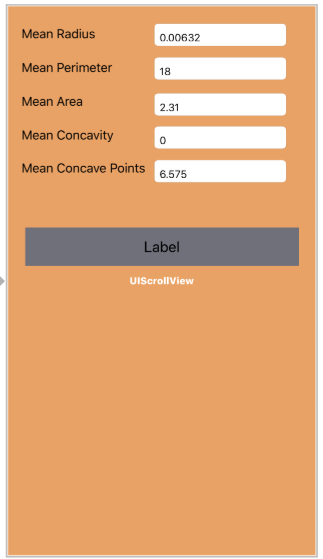
model.save("cancermodel.mlmodel")

For this to work you have to install coreml using your pip. And also write the below code on the top to import it.

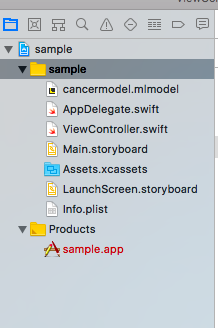
import coremltools

Once you run this program you will get a model file in your disk. Named as cancermodel.mlmodel which you use in your IOS project for inference.

# Creating an IOS APP

Open Xcode. [Note it must be ver 9+]. Then create an empty swift application with story board. In the main story board design the screen like this.

Then add the generated model file to your project. Now trhe project structure will look like this.



Now create outlets to each and every text field. And add on change event listener to each and every text field.

Now your view controller will become like this.

import UIKit

import CoreML

class ViewController: UIViewController {

let model = cancermodel()

@IBOutlet weak var meanradius: UITextField!

@IBOutlet weak var cancertype: UILabel!

@IBOutlet weak var meanperimeter: UITextField!

@IBOutlet weak var meanarea: UITextField!

@IBOutlet weak var meanconcavity: UITextField!

@IBOutlet weak var meanconcavepoints: UITextField!

override func didReceiveMemoryWarning() {

super.didReceiveMemoryWarning()

// Dispose of any resources that can be recreated.

}

override func viewDidLoad() {

super.viewDidLoad();

updated(meanconcavepoints);

***//This line is to fire the initial update of the cancer type.***

}

***/\****

***This method will send the input data to your generated model class and display the returned result to the label.***

***\*/***

@IBAction func updated(\_ sender: Any) {

guard let modeloutput = try? model.prediction(mean\_radius: Double(meanradius.text!)!, mean\_perimeter: Double(meanperimeter.text!)!, mean\_area: Double(meanarea.text!)!, mean\_concavity: Double(meanconcavity.text!)!, mean\_concave\_points: Double(meanconcavepoints.text!)!) else {

fatalError("unexpected runtime error")

}

cancertype.text = modeloutput.typeofcancer;

}

}

You can get the same code in the below github URL.

If you got any issue while building. Like signing or certificate please google it or write to us.

Once you set up the project in Xcode. You can run in the simulator. The result will be like this.

