Objective:

To create a Random Forest decision tree classifier model to detect the type of cancer from the Brest cancer dataset.

About Dataset:

This data set will have the following information collected from number of patients.

1) ID number

2) Diagnosis (M = malignant, B = benign)

3-32)

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)

Here in this data set we have to find the type of cancer it is weather it a M = malignant, B = benign by taking all the other attributes.

Algorithm we are applying:

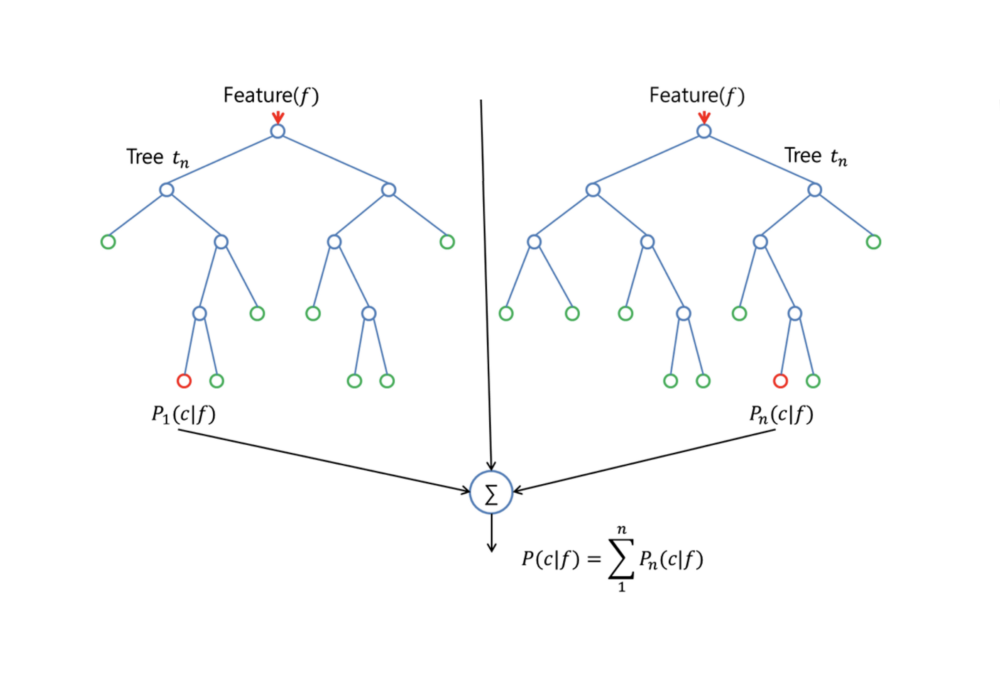
Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because its simplicity and the fact that it can be used for both classification and regression tasks. In this post, you are going to learn, how the random forest algorithm works and several other important things about it.

How it works:

Random Forest is a supervised learning algorithm. Like you can already see from it’s name, it creates a forest and makes it somehow random. The „forest“ it builds, is an ensemble of Decision Trees, most of the time trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. I will talk about random forest in classification, since classification is sometimes considered the building block of machine learning. Below you can see how a random forest would look like with two trees:



With a few exceptions a random-forest classifier has all the hyperparameters of a decision-tree classifier and also all the hyperparameters of a bagging classifier, to control the ensemble itself. Instead of building a bagging-classifier and passing it into a decision-tree-classifier, you can just use the random-forest classifier class, which is more convenient and optimized for decision trees. Note that there is also a random-forest regressor for regression tasks.

The random-forest algorithm brings extra randomness into the model, when it is growing the trees. Instead of searching for the best feature while splitting a node, it searches for the best feature among a random subset of features. This process creates a wide diversity, which generally results in a better model.

Therefore when you are growing a tree in random forest, only a random subset of the features is considered for splitting a node. You can even make trees more random, by using random thresholds on top of it, for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

Real Life Analogy:

Imagine a guy named Andrew, that want’s to decide, to which places he should travel during a one-year vacation trip. He asks people who know him for advice. First, he goes to a friend, tha asks Andrew where he traveled to in the past and if he liked it or not. Based on the answers, he will give Andrew some advice.

This is a typical decision tree algorithm approach. Andrews’s friend created rules to guide his decision about what he should recommend, by using the answers of Andrew.

Afterwards, Andrew starts asking more and more of his friends to advise him and they again ask him different questions, where they can derive some recommendations from. Then he chooses the places that where recommend the most to him, which is the typical Random Forest algorithm approach.

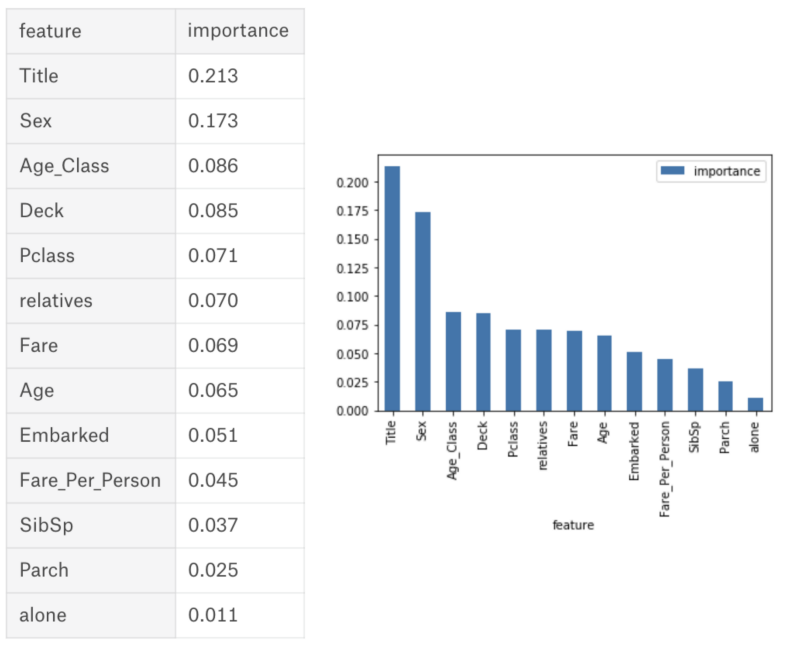
Feature Importance:

Another great quality of the random forest algorithm is that it is very easy to measure the relative importance of each feature on the prediction. Sklearn provides a great tool for this, that measures a features importance by looking at how much the tree nodes, which use that feature, reduce impurity across all trees in the forest. It computes this score automatically for each feature after training and scales the results, so that the sum of all importance is equal to 1.

If you don’t know how a decision tree works and if you don’t know what a leaf or node is, here is a good description from Wikipedia: In a decision tree each internal node represents a “test” on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). A node that has no children is a leaf.

Through looking at the feature importance, you can decide which features you may want to drop, because they don’t contribute enough or nothing to the prediction process. This is important, because a general rule in machine learning is that the more features you have, the more likely your model will suffer from over fitting and vice versa.

Below you can see a table and a visualization that show the importance of 13 features, which I used during a supervised classification project with the famous Titanic dataset on kaggle. You can find the whole project here.



Difference between Decision Trees and Random Forests:

Like I already mentioned, Random Forest is a collection of Decision Trees, but there are some differences.

If you input a training dataset with features and labels into a decision tree, it will formulate some set of rules, which will be used to make the predictions.

For example, if you want to predict whether a person will click on an online advertisement, you could collect the ad’s the person clicked in the past and some features that describe his decision. If you put the features and labels into a decision tree, it will generate some rules. Then you can predict whether the advertisement will be clicked or not. In comparison, the Random Forest algorithm randomly selects observations and features to build several decision trees and then averages the results.

Another difference is that „deep “decision trees might suffer from over fitting. Random Forest prevents over fitting most of the time, by creating random subsets of the features and building smaller trees using these subsets. Afterwards, it combines the sub trees. Note that this doesn’t work every time and that it also makes the computation slower, depending on how many trees your random forest builds.

## Here is our implementation for our dataset:

//importing required packages / modules.  
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  
import coremltools  
  
  
//Loading breast cancer dataset  
dataset = ds.load\_breast\_cancer()  
  
//creating a pandas dataframe with data  
cancerdata = pd.DataFrame(dataset.data)  
  
//getting all the column names.  
cancerdata.columns = dataset.feature\_names  
  
  
//Deleting all other data except the below mentioned columns to make the dataset consistent  
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)  
  
  
# print(cancerdata)  
//Exporting the resulted data into excel  
cancerdata.to\_csv("myfile.csv")  
  
//assigning types array to cancer\_types  
cancer\_types = dataset.target\_names  
  
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]])  
  
# print(cancer\_names)  
  
//spliting as test & train data  
x\_train, x\_test, y\_train, y\_test = sklearn.model\_selection.train\_test\_split(cancerdata,cancer\_names,test\_size=0.3,  
 random\_state=5)  
  
//Initilizing thge classifier  
classifier = RandomForestClassifier()  
//feeding the training data and fitting it.  
classifier.fit(x\_train, y\_train)  
  
//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")  
  
//testing the model with test data  
print(classifier.predict(x\_test))  
cancerdata['type'] = cancer\_names  
cancerdata.to\_csv("myfile2.csv")  
# print(dataset.feature\_names)

Once you run the above program you will see the test results and you can match this with the myfile.csv which was generated.

Now the pending task is to use this model in the IOS app.

To do this you need some IOS /Swift programming experience. And a MAC system having XCode 9+ installed.

You can download the app from the provided CD.

ANs find the breast Cancer folder and open Breast Cancer.xcdoeproj file in the xcode. There you can go through the view controller file to understand the code.

//

// ViewController.swift

// sample

//

// Created by BossmediaNT on 24/01/18.

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 method will be called whenever any textbox in the UI-Form updated this will pass the form data to the model and get the result displayed in the cancertype textbox\*/

@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;

}

}