FRANKFURT UNIVERSIY OF APPLIED SCIENCES, MARCH 2019, AI/ML SOFTWARE ENGINEERING SUPERVISED BY DAMIR DOBRIC

Decision Forest Regression

Sehrish Fahim
Sehrish.fahim91@gmail.com|1277599

Abstract
This document contain the information about the decision forest regression and approach has taken for implementation for machine learning using random forest regression, Random forest algorithm can use both for classification and the regression kind of problems. Decision Trees combined with Bootstrap aggregation make Random Forests a remarkably robust yet simple learning model less prone to overfitting vs single Decision Trees for supervised learning problems.

Keywords — Decision Forest Regression, RSS. CART.

INTRODUCTION

In decision trees we have concept of Classification and regression which you can say CART in short refer to Decision Tree calculations that can utilized for classification or regression problems which cover almost 90% of problems.
The go for each stage is to relate explicit targets (i.e, wanted yield esteems) with explicit estimations of a specific variable. The outcome is a choice tree in which every way distinguishes a combination of values associated with a particular predictions.
Each non-leaf node in this trees is essentially a chief. These node are called decision nodes. Every nodes does a particular test to figure out where to go straightaway. Contingent upon the result, you either go to one side branch or the correct part of this nodes. We continue doing this until we achieve a leaf nodes.
The calculation to create a decision tree is recursive. Each emphasis finds the most ideal approach to part a present preparing subset into two sections and get a partitioning condition.
Which division is the best? There are a few criteria. Gini method mostly use for classification.

Example for regression: identify blood pressure of a person on The basis of height, age and weight etc.
Example for classification: predict survival of a person Y/N On the basis of age, height and weight.

METHODS

For regression the RSS (residual sum of squares) criteria will be connected, yet the Gini is appropriate for classification. The following steps describe splitting algorithms in general. For each component (aside from resolution feature).

1-Calculate RSS for current subset;
2-Sort things by feature value;
3-For every conceivable division into two subsets,
4-figure RSS for left (L) and right (R) subset;
5-Find the division with min(RSS(L)+RSS(R)) and return include name and value. Criteria to stop recursion RSS for current subset is equivalent to 0;

Things include in current subset is not exactly or equivalent to the acceptable maximum number of things in a terminal node. Decision tree will be created and preparing set will be separated into various categories (Terminal nodes) accordingly.

Why Decision Forest Model for Regression ?

A forest is various trees. What's more, what is an "random" Forest? It is various choice trees created dependent on an arbitrary subset of the underlying preparing set. Interestingly, any random subset to produce a choice tree can contain copy preparing things. Each tree of a forest makes its expectation; a forecast of a prediction is a arithmetic mean of every one of trees' prediction.

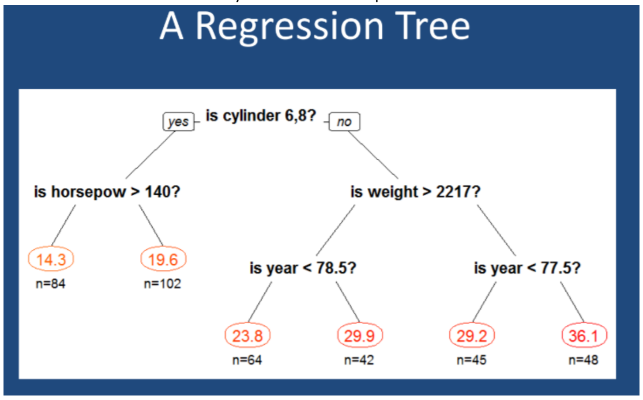


Fig no.1 : Regression tree

One of the essential advantages of utilizing a CART is that, by development, it produces if or then else decision rulesets, which are similar to graphical charts to represent trees. Their principle detriment lies in the way that they are frequently uncompetitive with other directed systems, for example, neural networks in terms of prediction accuracy. They can turn out to be very competitive when utilized in an outfit strategy, for example, with bootstrap aggregation / bagging, Random Forests or boosting.

In insights, the residual sum of squares (RSS), is the sum of the squares of residuals (deviations predicted from given observational estimations of given data). It is a proportion of the inconsistency between the information of given data and an estimation model. A little RSS shows a tight fit of the model to the information. It is utilized as an optimality criterion in parameter choice and model determination.

The RSS can be describe as: RSS=∑m=1M∑i∈Rm(yi−y^Rm)2

PROJECT

A) Git Hub reference,
https://github.com/UniversityOfAppliedSciencesFrankfurt/se-dystsys-2018-2019-softwareengineering/tree/SehrishFahim

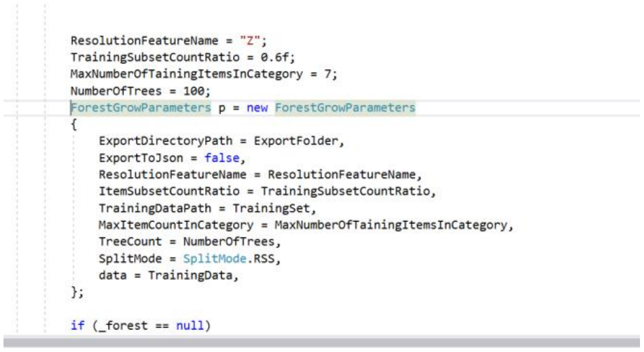
Random Forest is an effective algorithm to prepare from model improvement process, to distinguish how it performs and it's difficult to implement a bad Random Forest, in view of its effortlessness. This algorithm is additionally an extraordinary decision maker, in the event that you have to build up a model in a brief timeframe. In addition, it split your features smartly. Random Forests are likewise exceptionally difficult to beat regarding execution. Obviously you can probably dependably locate a model that can perform better, similar to a neural system, yet these typically take knowingly more time in the advancement. Also, in addition, they can deal with a variety of feature types, similar binary, categorical and numerical. Depending on the number of trees you plug it gets better and better in accuracy.

Code structure or Working:

Cater both scenario of Splitting mechanism

1. Gini index (suitable for classification)
2. RSS (suitable for regression)

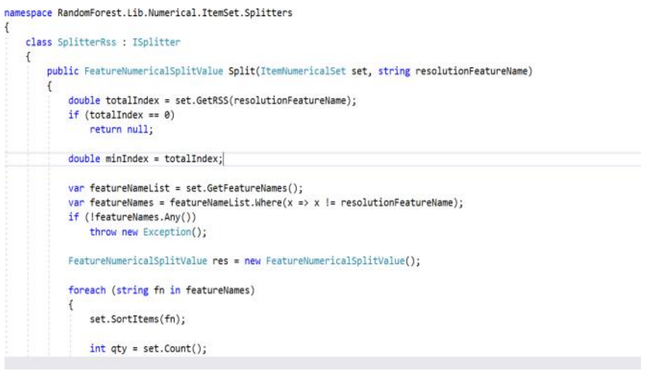
Training has done by taking 80% of the given data, number of trees and other parameters are configurable as follows:



```
ResolutionFeatureName = "Z";
TrainingSubsetCountRatio = 0.6f;
MaxNumberOfTainingItemsInCategory = 7;
NumberOfTrees = 100;
ForestGrowParameters p = new ForestGrowParameters
{
    ExportDirectoryPath = ExportFolder,
    ExportToJson = false,
    ResolutionFeatureName = ResolutionFeatureName,
    ItemSubsetCountRatio = TrainingSubsetCountRatio,
    TrainingDataPath = TrainingSet,
    MaxItemCountInCategory = MaxNumberOfTainingItemsInCategory,
    TreeCount = NumberOfTrees,
    SplitMode = SplitMode.RSS,
    data = TrainingData,
};

if (_forest == null)
```

Fig. no. 2: parameter setting

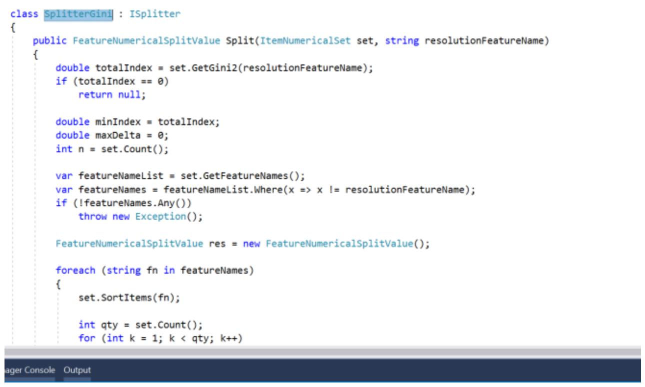
Split mode is selectable by giving configuration.



```
namespace RandomForest.Lib.Numerical.ItemSet.Splitters
{
    class SplitterRss : ISplitter
    {
        public FeatureNumericalSplitValue Split(ItemNumericalSet set, string resolutionFeatureName)
        {
            double totalIndex = set.GetRSS(resolutionFeatureName);
            if (totalIndex == 0)
                return null;

            double minIndex = totalIndex;

            var featureNameList = set.GetFeatureNames();
            var featureNames = featureNameList.Where(x => x != resolutionFeatureName);
            if (!featureNames.Any())
                throw new Exception();

            FeatureNumericalSplitValue res = new FeatureNumericalSplitValue();

            foreach (string fn in featureNames)
            {
                set.SortItems(fn);

                int qty = set.Count();
```

Fig. no. 3:RSS method

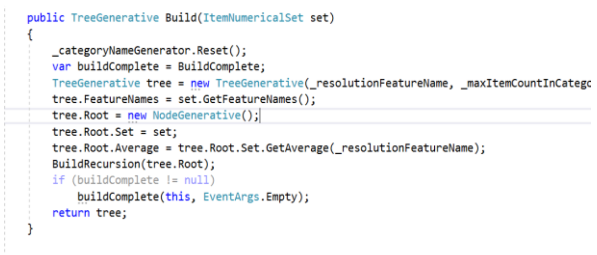
Splitting Rss logic.



```
class SplitterGini : ISplitter
{
    public FeatureNumericalSplitValue Split(ItemNumericalSet set, string resolutionFeatureName)
    {
        double totalIndex = set.GetGini2(resolutionFeatureName);
        if (totalIndex == 0)
            return null;

        double minIndex = totalIndex;
        double maxDelta = 0;
        int n = set.Count();

        var featureNameList = set.GetFeatureNames();
        var featureNames = featureNameList.Where(x => x != resolutionFeatureName);
        if (!featureNames.Any())
            throw new Exception();

        FeatureNumericalSplitValue res = new FeatureNumericalSplitValue();

        foreach (string fn in featureNames)
        {
            set.SortItems(fn);

            int qty = set.Count();
            for (int k = 1; k < qty; k++)
```

Fig. no. 4: Gini method

Gini split logic.



```
public TreeGenerative Build(ItemNumericalSet set)
{
    _categoryNameGenerator.Reset();
    var buildComplete = BuildComplete;
    TreeGenerative tree = new TreeGenerative(_resolutionFeatureName, _maxItemCountInCategc
    tree.FeatureNames = set.GetFeatureNames();
    tree.Root = new NodeGenerative();
    tree.Root.Set = set;
    tree.Root.Average = tree.Root.Set.GetAverage(_resolutionFeatureName);
    BuildRecursion(tree.Root);
    if (buildComplete != null)
        buildComplete(this, EventArgs.Empty);
    return tree;
}
```

Fig. no. 5: Build trees

Building trees recursively.

Working:

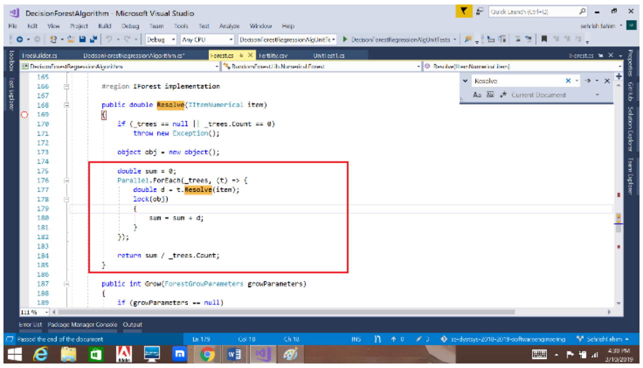
Run method: by providing double [][] data type and setting following parameters:

```
TreeCount = NumberOfTrees,
SplitMode = SplitMode.RSS,
data = TrainingData,
```

There is GenerateTreesTPL method which actually create tress on the bases of split mode selected. Build method as shown in Fig no 5 build the trees and decide which value and feature will be the node and then it call BuildRecusrion method which actually call recursively to create tree until it reach to terminal node and divide data to left and right node. Now at this stage according to the random subset of our given data trees have been build and system it trained now it is ready to predict.

Predict method:

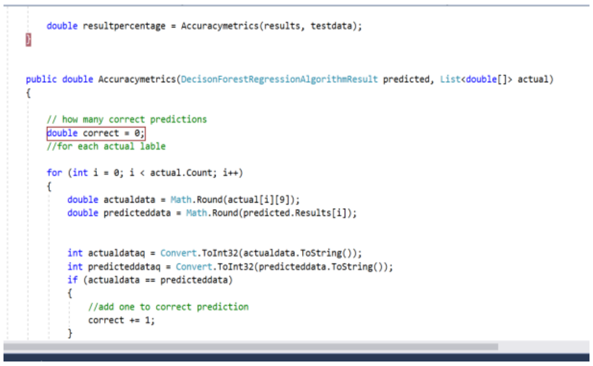
It calls resolve method which work as follows: for each no of tree we open parallel thread and check where the coming data fis in our build trees. Finally after getting result it sum the result of each result from different trees and divide by number of trees and return the result.



```
#region IForest implementation

public double Resolve(IItemNumerical item)
{
    if (_trees == null || _trees.Count == 0)
        throw new Exception();

    object obj = new object();

    double sum = 0;
    Parallel.ForEach(_trees, (t) => {
        double d = t.Resolve(item);
        lock(obj)
        {
            sum = sum + d;
        }
    });

    return sum / _trees.Count;
}

public int Grow(ForestGrowParameters growParameters)
{
    if (growParameters == null)
```

Fig. no.6: Result for each data

Unit Test:



```
double resultpercentage = Accuracymetrics(results, testdata);
}

public double Accuracymetrics(DecisonForestRegressionAlgorithmResult predicted, List<double[]> actual)
{
    // how many correct predictions
    double correct = 0;
    //for each actual lable

    for (int i = 0; i < actual.Count; i++)
    {
        double actualdata = Math.Round(actual[i][9]);
        double predicteddata = Math.Round(predicted.Results[i]);


        int actualdataq = Convert.ToInt32(actualdata.ToString());
        int predicteddataq = Convert.ToInt32(predicteddata.ToString());
        if (actualdata == predicteddata)
        {
            //add one to correct prediction
            correct += 1;
        }
```

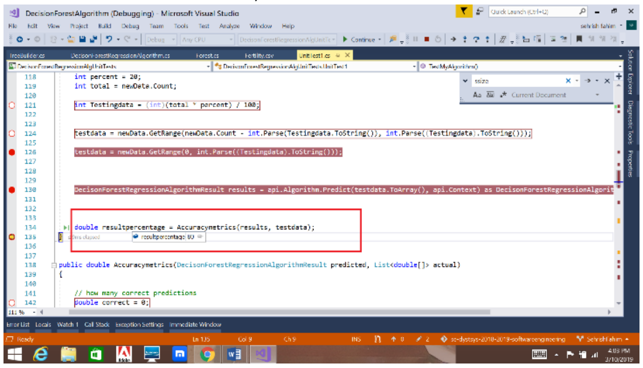
Fig. no.7: accuracy detection unit test

To calculate the accuracy of correct prediction from remaining 20% data.

Result:

There are two test case which works as follows:

Case1: First it takes 20% data which we kept for testing purpose and after trained our system with this data it goes and compare which expected result and gives the percentage of accuracy in this case it is 80%.

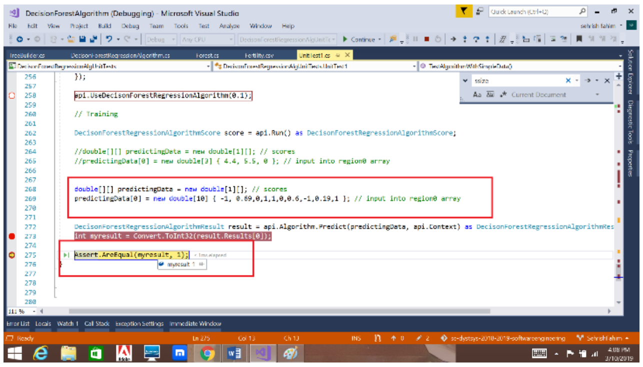


```
int percent = 20;
int total = newData.Count;

int Testingdata = (int)(total * percent) / 100;


testdata = newData.GetRange(newData.Count - int.Parse(Testingdata.ToString()), int.Parse((Testingdata).ToString()));

testdata = newData.GetRange(0, int.Parse((Testingdata).ToString()));



DecisonForestRegressionAlgorithmResult results = api.Algorithm.Predict(testdata.ToArray(), api.Context) as DecisonForestRegressionAlgorit



double resultpercentage = Accuracymetrics(results, testdata);
                                                resultpercentage 80


public double Accuracymetrics(DecisonForestRegressionAlgorithmResult predicted, List<double[]> actual)
{

    // how many correct predictions
    double correct = 0;
```

Fig. no.8: Result for case1

Case2:

By giving one test data from 20% data and check if it gives accurate result.



```
    });

    api.UseDecisonForestRegressionAlgorithm(0.1);

    // Training

    DecisonForestRegressionAlgorithmScore score = api.Run() as DecisonForestRegressionAlgorithmScore;

    //double[][] predictingData = new double[1][]; // scores
    //predictingData[0] = new double[3] { 4.4, 5.5, 0 }; // input into region0 array


    double[][] predictingData = new double[1][]; // scores
    predictingData[0] = new double[10] { -1, 0.69,0,1,1,0,0.6,-1,0.19,1 }; // input into region0 array


    DecisonForestRegressionAlgorithmResult result = api.Algorithm.Predict(predictingData, api.Context) as DecisonForestRegressionAlgorithmRes
    int myresult = Convert.ToInt32(result.Results[0]);

    Assert.AreEqual(myresult, 1);
}
```

DecisionForestAlgorithm (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Continue
UnitTest1.cs
TestAlgorithmWithSimpleData()
myresult 1
Ready
Ln 275 Col 13 Ch 13 INS

Fig. no.9: Result for case2

Data Description Expectation and Result:

This data is for fertility and diagnose as normal or altered depending on certain following parameters:

Season in which the analysis was performed. 1) winter, 2) spring, 3) Summer, 4) fall. (-1, -0.33, 0.33, 1)
Age at the time of analysis. 18-36 (0, 1)
Childish diseases (ie , chicken pox, measles, mumps, polio) 1) yes, 2) no. (0, 1)
Accident or serious trauma 1) yes, 2) no. (0, 1)
Surgical intervention 1) yes, 2) no. (0, 1)
High fevers in the last year 1) less than three months ago, 2) more than three months ago, 3) no. (-1, 0, 1)
Frequency of alcohol consumption 1) several times a day, 2) every day, 3) several times a week, 4) once a week, 5) hardly ever or never (0, 1)
Smoking habit 1) never, 2) occasional 3) daily. (-1, 0, 1)
Number of hours spent sitting per day ene-16 (0, 1)
Output: Diagnosis normal (1), altered (2)

Expectation for the correct data is vary on the trees we have as we predict the 20% remaining data it gives 95% accuracy

Test Methods:

There are two test method, one **TestAlgorithmWithSimpleData** it after get trained the fresh data has been provided and called assert method to check if predicted result is right or not.

Second method **TestMyAlgorithm** is taking whole 20% remaining data and get all the predictions into the array and then it compares with the actual data result we already have and the **Accuracy metrics** function tells us how many prediction was correct so in this second method we have percentage of correct out of 100.

Discussion and conclusion:

As described above, an advantage of random forest is that it is suitable for both regression and classification. Random Forest is likewise considered as a helpful and simple to utilize calculation, due to tree base decision it gives decent forecast outcome. The quantity of parameters is additionally not excessively high and they are direct to understand. One of the major issues in AI is overfitting, yet more often than not this won't occur that simple to a random forest classifier. That is in such a case that there are sufficient trees in the forest, the classifier won't over fits the model. The fundamental constraint of Random Forest is that an expansive number of trees can influence the calculation to ease back to and insufficient for ongoing forecasts. As a rule, these calculations are quick to prepare, however very moderate to make expectations once they are prepared. A progressively exact forecast requires more trees, which results in a slower demonstrate.

To summarized, the aim is to construct the tree by recursively splitting the tree and choosing random forest over decision tree, the main idea here is as follows:

Decision forest covers almost every CART problems.

As we do not want to over fit the data and that is what happen in one decision tree as you already know where to put what, in order to prevent that we create a lot more trees on random subsamples of given data and for each subsample we create trees when we have bunch of trees that we trained, then we take result by majority vote (classification) or by taking mean (regression) also that's given us higher accuracy over just using one decision tree alone.

Random forest is most suitable machine learning technique right now the reason being they can be used for both classification and regression as more than 90% problems are based on these two.

REFERENCES

- DecisionForestsintheTask of Semi-Supervised Learning
- Random decision forests - IEEE Conference Publication - IEEE Xplore
- https://github.com/pavl0v/RandomForest/
- web.stanford.edu/class/stats202/content/lec19.pdf
- *https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi.*
- https://archive.ics.uci.edu/ml/datasets/Fertility#