**SAMPLE CODE**

**Random forest:**

import java.util.ArrayList;

import java.util.HashMap;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

public class RandomForest {

private static int NUM\_THREADS;//=Runtime.getRuntime().availableProcessors();

public static int C;

public static int M;

public static int Ms;//recommended by Breiman: =(int)Math.round(Math.log(M)/Math.log(2)+1);

private long time\_o;

private int numTrees;

private double update;

private double progress;

private int[] importances;

private HashMap<int[],int[]> estimateOOB;

private double error;

private ExecutorService treePool;

private ArrayList<ArrayList<String>> data;

private ArrayList<ArrayList<String>> testdata;

private ArrayList<ArrayList<String>> Prediction;

public ArrayList<Character> DataAttributes;

public HashMap<ArrayList<String>, String> FinalPredictions;

public int corretsPredictions;

@SuppressWarnings("static-access")

public RandomForest(ArrayList<Character> dataLayout,int numTrees,int numThreads,int M,int Ms,int C, ArrayList<ArrayList<String>> train,ArrayList<ArrayList<String>> test) {

// TODO Auto-generated constructor stub

StartTimer();

this.numTrees=numTrees;

this.NUM\_THREADS=numThreads;

this.data=train;

this.testdata=test;

this.M=M;

this.Ms=Ms;

this.C=C;

this.DataAttributes = dataLayout;

trees2 = new ArrayList<DecisionTree>(numTrees);

update=100/((double)numTrees);

progress=0;

corretsPredictions =0;

System.out.println("creating "+numTrees+" trees in a random Forest. . . ");

System.out.println("total data size is "+train.size());

System.out.println("number of attributes "+M);

System.out.println("number of selected attributes "+Ms);

estimateOOB=new HashMap<int[],int[]>(data.size());

Prediction = new ArrayList<ArrayList<String>>();

FinalPredictions = new HashMap<ArrayList<String>, String>();

}

@SuppressWarnings("unchecked")

public void Start(boolean forAccuracy,boolean withThreads) {

// TODO Auto-generated method stub

System.out.println("Number of threads started : "+NUM\_THREADS);

System.out.print("Starting trees");

treePool=Executors.newFixedThreadPool(NUM\_THREADS);

for (int t=0;t<numTrees;t++){

treePool.execute(new CreateTree(data,this,t+1));

}treePool.shutdown();

try {

treePool.awaitTermination(10,TimeUnit.SECONDS); //effectively infinity

} catch (InterruptedException ignored){

System.out.println("interrupted exception in Random Forests");

}

System.out.println("Trees Production completed in "+TimeElapsed(time\_o));

if(forAccuracy){

if(withThreads){

System.out.println("Testing Forest for Accuracy with threads");

ArrayList<DecisionTree> Tree1 = (ArrayList<DecisionTree>) trees2.clone();

TestforAccuracy(Tree1,testdata,data);

}else{

System.out.println("Testing Forest for Accuracy without threads");

ArrayList<DecisionTree> Tree2 = (ArrayList<DecisionTree>) trees2.clone();

TestForestForAccuracy(Tree2, data, testdata);

}

}else{

if(withThreads){

System.out.println("Testing Forest for Labels with threads");

ArrayList<DecisionTree> Tree3 = (ArrayList<DecisionTree>) trees2.clone();

TestForestForLabelWT(Tree3, data, testdata);

}else{

System.out.println("Testing Forest for Labels without threads");

ArrayList<DecisionTree> Tree4 = (ArrayList<DecisionTree>) trees2.clone();

TestForestForLabel(Tree4, data, testdata);

}

}

}

private void TestforAccuracy(ArrayList<DecisionTree> trees,ArrayList<ArrayList<String>> Testdata,ArrayList<ArrayList<String>> TrainData) {

long time2 = System.currentTimeMillis();

ExecutorService TestthreadPool = Executors.newFixedThreadPool(NUM\_THREADS);

for(ArrayList<String> TP:Testdata){

TestthreadPool.execute(new TestTree(TP,trees,TrainData));

}TestthreadPool.shutdown();

try{

TestthreadPool.awaitTermination(10, TimeUnit.SECONDS);

}catch(InterruptedException ignored){

System.out.print("Interuption in testing");

}System.out.println("Testing Complete");

System.out.println("Results are ...");

System.out.println("Forest Accuracy is "+((corretsPredictions\*100)/Testdata.size())+"%");

System.out.println("this test was done in "+TimeElapsed(time2));

System.out.println("");System.out.println("");

}

private void TestForestForLabel(ArrayList<DecisionTree> trees,ArrayList<ArrayList<String>> traindata,ArrayList<ArrayList<String>> testdata) {

// TODO Auto-generated method stub

long time = System.currentTimeMillis();

int treee=1;

System.out.println("Predicting Labels now");

for(DecisionTree DTC : trees){

DTC.CalculateClasses(traindata, testdata, treee);treee++;

if(DTC.predictions!=null)

Prediction.add(DTC.predictions);

}

for(int i = 0;i<testdata.size();i++){

ArrayList<String> Val = new ArrayList<String>();

for(int j=0;j<trees.size();j++){

Val.add(Prediction.get(j).get(i));

}

String pred = ModeofList(Val);

System.out.println("["+pred+"]: Class predicted for data point: "+i+1);

}

System.out.println("this test was done in "+TimeElapsed(time));

}

private void TestForestForLabelWT(ArrayList<DecisionTree> tree,ArrayList<ArrayList<String>> traindata,ArrayList<ArrayList<String>> testdata) {

long time = System.currentTimeMillis();

ExecutorService TestthreadPool = Executors.newFixedThreadPool(NUM\_THREADS);int i=1;

for(ArrayList<String> TP:testdata){

TestthreadPool.execute(new TestTreeforLabel(TP,tree,traindata,i));i++;

}TestthreadPool.shutdown();

try{

TestthreadPool.awaitTermination(10, TimeUnit.SECONDS);

}catch(InterruptedException ignored){

System.out.print("Interuption in testing");

}

System.out.println("Testing Complete");

System.out.println("this test was done in "+TimeElapsed(time));

}

public void TestForestForAccuracy(ArrayList<DecisionTree> trees,ArrayList<ArrayList<String>> train,ArrayList<ArrayList<String>> test){

long time = System.currentTimeMillis();

int correctness=0;ArrayList<String> ActualValues = new ArrayList<String>();

for(ArrayList<String> s:test){

ActualValues.add(s.get(s.size()-1));

}int treee=1;

System.out.println("Testing forest now ");

for(DecisionTree DTC : trees){

DTC.CalculateClasses(train, test, treee);treee++;

if(DTC.predictions!=null)

Prediction.add(DTC.predictions);

}

for(int i = 0;i<test.size();i++){

ArrayList<String> Val = new ArrayList<String>();

for(int j=0;j<trees.size();j++){

Val.add(Prediction.get(j).get(i));

}

String pred = ModeofList(Val);

if(pred.equalsIgnoreCase(ActualValues.get(i))){

correctness = correctness +1;

}

}

System.out.println("The Result of Predictions :-");

System.out.println("Total Cases : "+test.size());

System.out.println("Total CorrectPredicitions : "+correctness);

System.out.println("Forest Accuracy :"+(correctness\*100/test.size())+"%");

System.out.println("this test was done in "+TimeElapsed(time));

}

public String ModeofList(ArrayList<String> predictions) {

// TODO Auto-generated method stub

String MaxValue = null; int MaxCount = 0;

for(int i=0;i<predictions.size();i++){

int count=0;

for(int j=0;j<predictions.size();j++){

if(predictions.get(j).trim().equalsIgnoreCase(predictions.get(i).trim()))

count++;

if(count>MaxCount){

MaxValue=predictions.get(i);

MaxCount=count;

}

}

}return MaxValue;

}

private class CreateTree implements Runnable{

private ArrayList<ArrayList<String>> data;

private RandomForest forest;

private int treenum;

public CreateTree(ArrayList<ArrayList<String>> data,RandomForest forest,int num){

this.data=data;

this.forest=forest;

this.treenum=num;

}

public void run() {

//trees.add(new DTreeCateg(data,forest,treenum));

trees2.add(new DecisionTree(data, forest, treenum));

progress+=update;

}

}

public class TestTree implements Runnable{

public ArrayList<String> testrecord;

public ArrayList<DecisionTree> Trees;

public ArrayList<ArrayList<String>> trainData;

public TestTree(ArrayList<String> testpoint, ArrayList<DecisionTree> Dtrees, ArrayList<ArrayList<String>> train){

this.testrecord = testpoint;

this.Trees = Dtrees;

this.trainData = train;

}

@Override

public void run() {

//System.out.print("Testing...");

ArrayList<String> predictions = new ArrayList<String>();

for(DecisionTree DT:Trees){

String Class = DT.Evaluate(testrecord, trainData);

if(Class == null)

predictions.add("n/a");

else

predictions.add(Class);

}

String finalClass = ModeofList(predictions);

if(finalClass.equalsIgnoreCase(testrecord.get(M)))

corretsPredictions++;

//System.out.println(finalClass);

FinalPredictions.put(testrecord,finalClass);

}

}

public class TestTreeforLabel implements Runnable{

public ArrayList<String> testrecord;

public ArrayList<DecisionTree> Trees;

public ArrayList<ArrayList<String>> trainData;

public int point;

public TestTreeforLabel(ArrayList<String> dp, ArrayList<DecisionTree> dtree, ArrayList<ArrayList<String>> data,int i){

this.testrecord = dp;

this.Trees = dtree;

this.trainData = data;

this.point =i;

}

@Override

public void run() {

ArrayList<String> predictions = new ArrayList<String>();

for(DecisionTree DT:Trees){

String Class = DT.Evaluate(testrecord, trainData);

if(Class == null)

predictions.add("n/a");

else

predictions.add(Class);

}

String finalClass = ModeofList(predictions);

System.out.println("["+finalClass+"]: Class predicted for data point: "+point);

}

}

private void StartTimer(){

time\_o=System.currentTimeMillis();

}

private static String TimeElapsed(long timeinms){

double s=(double)(System.currentTimeMillis()-timeinms)/1000;

int h=(int)Math.floor(s/((double)3600));

s-=(h\*3600);

int m=(int)Math.floor(s/((double)60));

s-=(m\*60);

return ""+h+"hr "+m+"m "+s+"sec";

}

}

**Random forest categ:**

package com.rf.real.categ;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

public class RandomForestCateg {

private static final int NUM\_THREADS=Runtime.getRuntime().availableProcessors();

public static int C;

public static int M;

public static int Ms;//recommended by Breiman: =(int)Math.round(Math.log(M)/Math.log(2)+1);

private ArrayList<DTreeCateg> trees;

private ArrayList<DTreeCateg2> trees2;

private long time\_o;

private int numTrees;

private double update;

private double progress;

private int[] importances;

private HashMap<int[],int[]> estimateOOB;

private double error;

private ExecutorService treePool;

private ArrayList<ArrayList<String>> data;

private ArrayList<ArrayList<String>> testdata;

private ArrayList<ArrayList<String>> Prediction;

public ArrayList<Integer> TrainAttributes;

public ArrayList<Integer> TestAttributes;

public RandomForestCateg(int numTrees,int M,int Ms,int C, ArrayList<ArrayList<String>> train,ArrayList<ArrayList<String>> test) {

// TODO Auto-generated constructor stub

StartTimer();

this.numTrees=numTrees;

this.data=train;

this.testdata=test;

this.M=M;

this.Ms=Ms;

this.C=C;

this.TrainAttributes=GetAttributes(train);

this.TestAttributes=GetAttributes(test);

trees = new ArrayList<DTreeCateg>(numTrees);

trees2 = new ArrayList<DTreeCateg2>(numTrees);

update=100/((double)numTrees);

progress=0;

System.out.println("creating "+numTrees+" trees in a random Forest. . . ");

System.out.println("total data size is "+train.size());

System.out.println("number of attributes "+M);

System.out.println("number of selected attributes "+Ms);

estimateOOB=new HashMap<int[],int[]>(data.size());

Prediction = new ArrayList<ArrayList<String>>();

}

/\*\*

\* Begins the random forest creation

\*/

public void Start() {

// TODO Auto-generated method stub

System.out.println("Number of threads started : "+NUM\_THREADS);

System.out.println("Starting trees");

treePool=Executors.newFixedThreadPool(NUM\_THREADS);

for (int t=0;t<numTrees;t++){

treePool.execute(new CreateTree(data,this,t+1));

}treePool.shutdown();

try {

treePool.awaitTermination(Long.MAX\_VALUE,TimeUnit.SECONDS); //effectively infinity

} catch (InterruptedException ignored){

System.out.println("interrupted exception in Random Forests");

}

if(data.get(0).size()>testdata.get(0).size()){

//TestForestNoLabel2(trees2, data, testdata);

TestForestNoLabel(trees,data,testdata);

}

else if(data.get(0).size()==testdata.get(0).size()){

TestForest2(trees2, data, testdata);

//TestForest(trees,data,testdata);

}

else

System.out.println("Cannot test this data");

System.out.print("Done in "+TimeElapsed(time\_o));

}

private void TestForestNoLabel(ArrayList<DTreeCateg> trees2,ArrayList<ArrayList<String>> data2,ArrayList<ArrayList<String>> testdata2) {

// TODO Auto-generated method stub

ArrayList<String> TestResult = new ArrayList<String>();

System.out.println("Predicting Labels now");

for(ArrayList<String> DP:testdata2){

ArrayList<String> Predict = new ArrayList<String>();

for(DTreeCateg DT:trees2){

Predict.add(DT.Evaluate(DP, testdata2));

}

TestResult.add(ModeofList(Predict));

}

}

public void TestForest(ArrayList<DTreeCateg> trees,ArrayList<ArrayList<String>> train,ArrayList<ArrayList<String>> test){

int correctness=0;ArrayList<String> ActualValues = new ArrayList<String>();

for(ArrayList<String> s:test){

ActualValues.add(s.get(s.size()-1));

}int treee=1;

System.out.println("Testing forest now ");

for(DTreeCateg DTC : trees){

DTC.CalculateClasses(train, test, treee);treee++;

if(DTC.predictions!=null)

Prediction.add(DTC.predictions);

}

for(int i = 0;i<test.size();i++){

ArrayList<String> Val = new ArrayList<String>();

for(int j=0;j<trees.size();j++){

Val.add(Prediction.get(j).get(i));

}

String pred = ModeofList(Val);

if(pred.equalsIgnoreCase(ActualValues.get(i))){

correctness = correctness +1;

}

}

System.out.println("The Result of Predictions :-");

System.out.println("Total Cases : "+test.size());

System.out.println("Total CorrectPredicitions : "+correctness);

System.out.println("Forest Accuracy :"+(correctness\*100/test.size())+"%");

}

private void TestForestNoLabel2(ArrayList<DTreeCateg2> trees22,ArrayList<ArrayList<String>> data2,ArrayList<ArrayList<String>> testdata2) {

// TODO Auto-generated method stub

ArrayList<String> TestResult = new ArrayList<String>();

System.out.println("Predicting Labels now");

for(ArrayList<String> DP:testdata2){

ArrayList<String> Predict = new ArrayList<String>();

for(DTreeCateg2 DT:trees22){

Predict.add(DT.Evaluate(DP, testdata2));

}

TestResult.add(ModeofList(Predict));

}

}

public void TestForest2(ArrayList<DTreeCateg2> trees,ArrayList<ArrayList<String>> train,ArrayList<ArrayList<String>> test){

int correctness=0;ArrayList<String> ActualValues = new ArrayList<String>();

for(ArrayList<String> s:test){

ActualValues.add(s.get(s.size()-1));

}int treee=1;

System.out.println("Testing forest now ");

for(DTreeCateg2 DTC : trees){

DTC.CalculateClasses(train, test, treee);treee++;

if(DTC.predictions!=null)

Prediction.add(DTC.predictions);

}

for(int i = 0;i<test.size();i++){

ArrayList<String> Val = new ArrayList<String>();

for(int j=0;j<trees.size();j++){

Val.add(Prediction.get(j).get(i));

}

String pred = ModeofList(Val);

if(pred.equalsIgnoreCase(ActualValues.get(i))){

correctness = correctness +1;

}

}

System.out.println("The Result of Predictions :-");

System.out.println("Total Cases : "+test.size());

System.out.println("Total CorrectPredicitions : "+correctness);

System.out.println("Forest Accuracy :"+(correctness\*100/test.size())+"%");

}

public String ModeofList(ArrayList<String> predictions) {

// TODO Auto-generated method stub

String MaxValue = null; int MaxCount = 0;

for(int i=0;i<predictions.size();i++){

int count=0;

for(int j=0;j<predictions.size();j++){

if(predictions.get(j).trim().equalsIgnoreCase(predictions.get(i).trim()))

count++;

if(count>MaxCount){

MaxValue=predictions.get(i);

MaxCount=count;

}

}

}return MaxValue;

}

private class CreateTree implements Runnable{

private ArrayList<ArrayList<String>> data;

private RandomForestCateg forest;

private int treenum;

public CreateTree(ArrayList<ArrayList<String>> data,RandomForestCateg forest,int num){

this.data=data;

this.forest=forest;

this.treenum=num;

}

public void run() {

//trees.add(new DTreeCateg(data,forest,treenum));

trees2.add(new DTreeCateg2(data, forest, treenum));

progress+=update;

}

}

private void StartTimer(){

time\_o=System.currentTimeMillis();

}

private static String TimeElapsed(long timeinms){

double s=(double)(System.currentTimeMillis()-timeinms)/1000;

int h=(int)Math.floor(s/((double)3600));

s-=(h\*3600);

int m=(int)Math.floor(s/((double)60));

s-=(m\*60);

return ""+h+"hr "+m+"m "+s+"sec";

}

private boolean isAlphaNumeric(String s){

char c[]=s.toCharArray();boolean hasalpha=false;

for(int j=0;j<c.length;j++){

hasalpha = Character.isLetter(c[j]);

if(hasalpha)break;

}return hasalpha;

}

private ArrayList<Integer> GetAttributes(List<ArrayList<String>> data){

ArrayList<Integer> Attributes = new ArrayList<Integer>();int iter = 0;

ArrayList<String> DataPoint = data.get(iter);

if(DataPoint.contains("n/a") || DataPoint.contains("N/A")){

iter = iter +1;

DataPoint = data.get(iter);

}

for(int i =0;i<DataPoint.size();i++){

if(isAlphaNumeric(DataPoint.get(i)))

Attributes.add(1);

else

Attributes.add(0);

}

return Attributes;

}

}