ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра АСУ

ОТЧЕТ

По лабораторной работе № 1

По дисциплине: Интеллектуальный анализ данных  
на тему: решение задачи классификация и регрессии с помощью деревьев решений.

Вариант № 18

Студента группы ИУС11

Сафонова Михаила  
  
Проверил:  
Васяева Т.А.

Донецк-2014

**Тема:** решение задачи классификация и регрессии с помощью деревьев решений.

**Цель работы:** изучение алгоритмов построения деревьев решений;

создание и исследование классификационных и регрессионных моделей с помощью деревьев решений.

**Задание**

Разработать на языке высокого уровня программу, реализующую алгоритм CART.

**Теоретические сведенья**

Алгоритм CART предназначен для построения бинарного дерева решений, каждый узел дерева при разбиении имеет только двух потомков. На каждом шаге построения дерева правило, формируемое в узле, делит заданное множество примеров на две части — часть, в которой выполняется правило (потомок — right) и часть, в которой правило не выполняется (потомок — left).

Преимуществом алгоритма CART является определенная гарантия того, что, если искомые детерминации существуют в исследуемой совокупности, то они будут выявлены. Кроме того, CART позволяет не «замыкаться» на единственном значении выходного признака, а искать все такие его значения, для которых можно найти соответствующее объясняющее выражение.

Метод CART применяется для **номинальных** (обычно двухуровневых) и **порядковых** предикторных переменных. В этом методе перебираются все возможные варианты ветвления для каждого узла, и выбирается та предикторная переменная, при которой оценочная функция дает наилучший показатель.

Общий вид обучающей выборки:

aardvark,1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,1

antelope,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,1

bass,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,4

bear,1,0,0,1,0,0,1,1,1,1,0,0,4,0,0,1,1

boar,1,0,0,1,0,0,1,1,1,1,0,0,4,1,0,1,1

buffalo,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,1

calf,1,0,0,1,0,0,0,1,1,1,0,0,4,1,1,1,1

carp,0,0,1,0,0,1,0,1,1,0,0,1,0,1,1,0,4

catfish,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,4

cavy,1,0,0,1,0,0,0,1,1,1,0,0,4,0,1,0,1

cheetah,1,0,0,1,0,0,1,1,1,1,0,0,4,1,0,1,1

chicken,0,1,1,0,1,0,0,0,1,1,0,0,2,1,1,0,2

chub,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,0,4

clam,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,7

crab,0,0,1,0,0,1,1,0,0,0,0,0,4,0,0,0,7

crayfish,0,0,1,0,0,1,1,0,0,0,0,0,6,0,0,0,7

crow,0,1,1,0,1,0,1,0,1,1,0,0,2,1,0,0,2

deer,1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,1

dogfish,0,0,1,0,0,1,1,1,1,0,0,1,0,1,0,1,4

dolphin,0,0,0,1,0,1,1,1,1,1,0,1,0,1,0,1,1

dove,0,1,1,0,1,0,0,0,1,1,0,0,2,1,1,0,2

duck,0,1,1,0,1,1,0,0,1,1,0,0,2,1,0,0,2

...

Общий вид тестируемой выборки:

stingray,0,0,1,0,0,1,1,1,1,0,1,1,0,1,0,1,4

swan,0,1,1,0,1,1,0,0,1,1,0,0,2,1,0,1,2

termite,0,0,1,0,0,0,0,0,0,1,0,0,6,0,0,0,6

toad,0,0,1,0,0,1,0,1,1,1,0,0,4,0,0,0,5

seasnake,0,0,0,0,0,1,1,1,1,0,1,0,0,1,0,0,3

...

Описание атрибутов:

1. animal name: Unique for each instance   
2. hair: Boolean   
3. feathers: Boolean   
4. eggs: Boolean   
5. milk: Boolean   
6. airborne: Boolean   
7. aquatic: Boolean   
8. predator: Boolean   
9. toothed: Boolean   
10. backbone: Boolean   
11. breathes: Boolean   
12. venomous: Boolean   
13. fins: Boolean   
14. legs: Numeric (set of values: {0,2,4,5,6,8})   
15. tail: Boolean   
16. domestic: Boolean   
17. catsize: Boolean   
18. type: Numeric (integer values in range [1,7])

Листинг:

public class CART\_Tree {

private Node root;

private double error;

private double testError;

private int countErr;

public CART\_Tree(Node root) {

this.root = root;

training();

error = calculateError();

MainFrame.getInstance().setError(error + "");

}

private void training() {

if (root != null) {

createTree(root);

}

else {

MainFrame.getInstance().showMessage("У дерева нет корня");

}

}

private void createTree(Node node) {

double[] q\_values = new double[19];

for (int i = 0; i < q\_values.length; i++) {

if (i >= 12 && i <= 15) {

q\_values[i] = getQ(node, i, Value.LEGS[i - 11]);

}

else {

q\_values[i] = getQ(node, i, 0);

}

}

double max = q\_values[0];

int max\_index = 0;

for (int i = 1; i < 19; i++) {

if (max <= q\_values[i]) {

max = q\_values[i];

max\_index = i;

}

}

if (max != 0) {

setNodes(node, max\_index);

node.setIndexPartition(max\_index);

node.setMessage(Value.messages[max\_index]);

}

else {

node.setLeaf(true);

node.setClassIndex(node.getValue(0).getType());

node.setMessage(node.getValues()[0].getType() + "");

return;

}

createTree(node.getLeft());

createTree(node.getRight());

}

private double getQ(Node node, int type, int value) {

int count\_l = 0;

int count\_r = 0;

for (int i = 0; i < node.length(); i++) {

if (type >= 12 && type <= 15) {

if ((int) node.getParameter(i, type) > value) {

count\_l++;

}

else {

count\_r++;

}

}

else {

if (!(boolean) node.getParameter(i, type)) {

count\_l++;

}

else {

count\_r++;

}

}

}

Value[] left = new Value[count\_l];

Value[] right = new Value[count\_r];

int r = 0;

int l = 0;

for (int i = 0; i < node.length(); i++) {

if (type >= 12 && type <= 15) {

if ((int) node.getParameter(i, type) > value) {

left[l] = node.getValue(i);

l++;

}

else {

right[r] = node.getValue(i);

r++;

}

}

else {

if (!(boolean) node.getParameter(i, type)) {

left[l] = node.getValue(i);

l++;

}

else {

right[r] = node.getValue(i);

r++;

}

}

}

return calculateQ(left, right);

}

private double calculateQ(Value[] left, Value[] right) {

double Q;

double Pl = (double) left.length / (double) (left.length + right.length);

double Pr = (double) right.length / (double) (left.length + right.length);

double W = 0;

for (int i = 1; i <= 7; i++) {

int Kl = 0;

int Kr = 0;

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

if (left[j].getType() == i) {

Kl++;

}

}

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

if (right[j].getType() == i) {

Kr++;

}

}

double one = 0;

if (left.length > 0) {

one = (double) Kl / (double) left.length;

}

double two = 0;

if (right.length > 0) {

two = (double) Kr / (double) right.length;

}

W += Math.abs(one - two);

}

Q = 2 \* Pl \* Pr \* W;

System.out.println("Лево: " + Pl + " право: " + Pr + "W = " + W + " Q= " + Q);

return Q;

}

private void setNodes(Node node, int number) {

Value[] left = null;

Value[] right = null;

int count\_l = 0;

int count\_r = 0;

int r = 0;

int l = 0;

if (number >= 12 && number <= 15) {

for (int i = 0; i < node.length(); i++) {

if ((int) node.getParameter(i, number) > Value.LEGS[number - 11]) {

count\_l++;

}

else {

count\_r++;

}

}

left = new Value[count\_l];

right = new Value[count\_r];

for (int i = 0; i < node.length(); i++) {

if ((int) node.getParameter(i, number) > Value.LEGS[number - 11]) {

left[l] = node.getValue(i);

l++;

}

else {

right[r] = node.getValue(i);

r++;

}

}

}

else {

for (int i = 0; i < node.length(); i++) {

if (!(boolean) node.getParameter(i, number)) {

count\_l++;

}

else {

count\_r++;

}

}

left = new Value[count\_l];

right = new Value[count\_r];

for (int i = 0; i < node.length(); i++) {

if (!(boolean) node.getParameter(i, number)) {

left[l] = node.getValue(i);

l++;

}

else {

right[r] = node.getValue(i);

r++;

}

}

}

node.setLeft(new Node(left));

node.setRight(new Node(right));

}

public void pruning() {

pruningOne(root);

}

private void pruningOne(Node node) {

if (!node.getLeaf()) {

node.getLeft().setLook(false);

node.getRight().setLook(false);

node.setLeaf(true);

double er = calculateError();

if ((error - er) <= 0) {

node.getLeft().setLook(true);

node.getRight().setLook(true);

node.setLeaf(false);

pruningOne(node.getLeft());

pruningOne(node.getRight());

}

else {

node.setLeft(null);

node.setRight(null);

return;

}

}

}

public void test(Value[] test\_values, Boolean flag) {

if (root != null) {

countErr = 0;

for (int i = 0; i < test\_values.length; i++) {

MainFrame.getInstance().setTextToFrame(test\_values[i].toString());

testOne(test\_values[i], root, flag);

if (flag) {

try {

Thread.sleep(1000);

}

catch (InterruptedException e) {

e.printStackTrace();

}

}

setUnFill(root);

}

testError = (double) countErr / (double) test\_values.length;

MainFrame.getInstance().setErrorTest(testError + "");

MainFrame.getInstance().endTesting();

if (flag) {

MainFrame.getInstance().setVisible(true);

}

}

else {

MainFrame.getInstance().showMessage("У дерева нет корня");

}

}

private void testOne(final Value value, Node node, Boolean flag) {

if (flag) {

try {

Thread.sleep(1000);

}

catch (InterruptedException e) {

e.printStackTrace();

}

}

node.setIsFill(true);

if (node.getLeaf()) {

if (value.getType() != node.getClassIndex()) {

countErr++;

}

MainFrame.getInstance().showMessage(value.toString() + "CLASS = " + node.getClassIndex());

MainFrame.getInstance().repaintTree();

}

else {

Object val;

switch (node.getIndexPartition()) {

case 0:

val = value.getHair();

break;

case 1:

val = value.getFeathers();

break;

case 2:

val = value.getEggs();

break;

case 3:

val = value.getMilk();

break;

case 4:

val = value.getAirborne();

break;

case 5:

val = value.getAquatic();

break;

case 6:

val = value.getPredator();

break;

case 7:

val = value.getToothed();

break;

case 8:

val = value.getBackbone();

break;

case 9:

val = value.getBreathes();

break;

case 10:

val = value.getVenomous();

break;

case 11:

val = value.getFins();

break;

case 16:

val = value.getTail();

break;

case 17:

val = value.getDomestic();

break;

case 18:

val = value.getCatsize();

break;

default:

val = value.getLegs();

}

if (node.getIndexPartition() >= 12 && node.getIndexPartition() <= 15) {

if ((int) val > Value.LEGS[ node.getIndexPartition() - 11]) {

MainFrame.getInstance().repaintTree();

testOne(value, node.getLeft(), flag);

}

else {

MainFrame.getInstance().repaintTree();

testOne(value, node.getRight(), flag);

}

}

else {

if (!(boolean) val) {

MainFrame.getInstance().repaintTree();

testOne(value, node.getLeft(), flag);

}

else {

MainFrame.getInstance().repaintTree();

testOne(value, node.getRight(), flag);

}

}

}

}

private void setUnFill(Node node) {

node.setIsFill(false);

if (node.getLeft() != null) {

setUnFill(node.getLeft());

}

if (node.getRight() != null) {

setUnFill(node.getRight());

}

}

private double calculateError() {

int er\_count = 0;

ArrayList<Node> leavs = getLeafs();

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

if (leavs.get(i).getLook()) {

int max\_count = 0;

int value\_class = 0;

int[] counts = new int[7];

int t = 0;

for (int j = 1; j < 8; j++, t++) {

for (int l = 0; l < leavs.get(i).getValues().length; l++) {

if (leavs.get(i).getValues()[l].getType() == j) {

counts[t]++;

}

}

}

for (int j = 0; j < 7; j++) {

if (max\_count < counts[j]) {

max\_count = counts[j];

value\_class = j + 1;

}

}

for (int j = 1; j < 8; j++) {

if (j != value\_class) {

if (counts[j - 1] > 0) {

er\_count += counts[j - 1];

}

}

}

}

}

return (double) er\_count / (double) root.getValues().length;

}

public ArrayList<Node> getLeafs() {

ArrayList<Node> leafs = new ArrayList();

getLeafs(root, leafs);

return leafs;

}

private void getLeafs(Node node, ArrayList leafs) {

if (!node.getLeaf()) {

getLeafs(node.getLeft(), leafs);

getLeafs(node.getRight(), leafs);

}

else {

leafs.add(node);

}

}

Результаты работы программы:



