CSCI_B 565 DATA MINING
Homework Number 3
Morning Class Computer Science Core
Spring
Indiana University,
Bloomington, IN

Puneet Loya
ploya

Mar 16,2015

All the work here is solemnly mine

1. (a) Possible number of trees:
   Consider C being the label,
   If 'A' becomes the root. It will have 3 children due to its variety (good, bad and ugly).

   At this stage, the children can be partitioned by either attribute 'B' or 'D'.

   Therefore number of trees with 'A' as root and label 'C' = $2^{variety} = 2^3 = 8$

   Number of trees with 'B' as root and label 'C' = $2^3 = 8$

   Number of trees with 'D' as root and label 'C' = $2^4 = 16$

   So if label is 'C', the number of trees are = $2^{variety(A)} + 2^{variety(B)} + 2^{variety(D)} = 8 + 8 + 16 = 32$

   So if label is 'A', the number of trees are = 8 + 4 + 16 = 28

   So if label is 'B', the number of trees are = 8 + 4 + 16 = 28

   So if label is 'D', the number of trees are = 8 + 8 + 4 = 20

   Total trees with one attribute as label = 32 + 28 + 28 + 20 = 108

   Consider two attributes as label,

   Then number of trees = 6 * 2 = 12

   Consider three attributes as label,

Then number of trees = 4

Hence, the possible number of trees = 108 + 12 + 4 = 124

(b) The number of possible functional dependencies:

Considering the attributes are: A, B, C and D.

Taking single attribute on LHS,
$A \rightarrow \{B\ or\ C\ or\ D\ or\ BC\ or\ CD\ or\ BD\ or\ BCD\}$

Therefore, number of functional dependencies for one attribute on LHS = 7 * 4 = 28
Taking two attribute on LHS,
$AB \rightarrow \{C\ or\ D\ or\ CD\ \}$
Therefore, number of functional dependencies for two attribute on LHS = 3 * 6 = 18
Taking three attribute on LHS,
$ABC \rightarrow D$
Therefore, number of functional dependencies for three attribute on LHS = 1 * 4 = 4

Hence, the total number of functional dependencies are: 28 + 18 + 4 = 50

(c) If a functional dependency is found, then few nodes in the decision tree can be discarded as they will always remain empty. The level of the decision tree may shorten and decision can be obtained with less traversal.

2. The procedure followed for building the tree.

- The input file is read to obtain details about the file that holds the data and the attributes required for the classification.
- The data file is then read and loaded into a mysql database as a table.
- Then data structures are built for the tree and handling other parts of the algorithm.
- Each node in the tree contains:
  - The query required to fetch its contents from the database.
  - A dictionary which contains label value as the key and value as its count.
  - List of children nodes
  - Parent reference, pointing to the parent node.
- The algorithm for the tree.
  - Create a root node of the tree
  - Calculate entropy of each non-label attribute.
  - The information gain is from the attribute with the least entropy. Select the attribute 'x'.
  - The data is now further partitioned on attribute 'x' and the children nodes are created.
  - The process is repeated with each child.
  - The algorithm stops when all attributes required for partition taken as input are considered.
- The code description:
  - **Id3Algo.java** - The algorithm is implemented in it.
  - **TreeNode.java** - Implements the N-ary tree.
  - **MysqlOps.java** - All the mysql data loading and querying.
  - **FileParser.java** - For the file handling operations.

- The procedure for testing the tree.
  - Each test entry in the test file is treated as a tuple.
  - **Naive Bayes(NB) Classifier** using [2] is implemented which returns the probability for each possible label.
    For Eg:
    $p(C = Y|A = good\ and\ D = rain) = p(A = good|C = Y)\ p(D = rain|C = Y)\ p(C = Y)$
  - The conditional probability for each given value in the test entry has to be calculated.
  - The conditional probability is obtained by traversing the tree.
  - If a node is unlabeled (zero elements in it) then the root node probability is chosen as the conditionaly probability.
  - Using these conditional probabilites, the probability for each label is calculated.
- The procedure for **missing values**:
  - All the possible unique values of that attribute are considered for the missing value.
  - The probability for each missing value is obtained from the tree.
  - **Weighted Average** probabilities with weight of each unique value of the attribute and its conditional probability is chosen as the conditional probability for the value.

    For Eg: Required query Attribute (A D)
    Query : good *
    The unique values of D = $snow, cloudy, rain, sun$
    $p(D = rain|C = Y),\ p(D = snow|C = Y),\ p(D = cloudy|C = Y),\ p(D = sun|C = Y)$ are calculated.

    Then $P(D = * \mid C = Y)$ = Weighted average of above probailities.

(a) The program will provide the probability even if the inputs are all *. An example output from the program:

```
Input:
table.txt
B
A D
ugly rain
1 .533
2 .200
3 .267
bad rain
1 .776
2 .173
3 .051
* *
1 .479
2 .266
3 .255
```

The answers are expected to be less accurate when the inputs are all *. It provides the overall average proaibility of the labels for the whole training set.

(b) The class probabilities are less accurate when * is added because a weighted average method has to be applied rather as accurate conditional probability is not available.

(c) The comparisons from having all inputs as * to none Example output:

```
Input:
table.txt
B
A D

* *
1 .479
2 .266
3 .255
* snow
1 .372
2 .496
3 .132
good snow
1 .149
2 .793
3 .059
bad snow
1 .495
2 .440
3 .065
ugly snow
1 .286
2 .429
3 .286
```

3. The 3-Fold Cross-Validation procedure:

   - The item set is divided into three sets using a bucket hash function.
     $f(x) = x\%3$
   - Three sets are formed.
   - Two sets are chosen as training and the remaining set is the test set.
   - The training set is used to build the tree.
   - Required attributes are extracted from the test set and used as the test cases against the trained decision tree.
   - The algorithm returns the probability for each label.
   - The test set and training set are again picked in round-robin fashion and the procedure repeats.
   - The code for Cross Validation is implemented in **CrossValidation.java**
   - output:

```
Input:

table.txt
C
A D B


Trial1
bad cloudy 1
Y .571
N .429
good snow 2
```

```
Y .500
N .500
bad snow 2
Y .500
N .500
Trial2
good cloudy 2
Y .742
N .258
bad rain 1
Y .923
N .077
bad rain 1
Y .923
N .077
Trial3
good snow 1
Y .667
N .333
ugly sun 3
Y .500
N .500
ugly sun 3
Y .500
N .500
```

- From the above output:
  Average probability for Y = 0.647
  Average probability for N = 0.342
  The average probability comes to be 0.999.
  The average probability gives the verdict that after multiple trials that average converges towards the mean of the underlying distribution. Referred [1] for the average probability inference. The average probability should be almost near to the output obtained when the inputs are all *.

4. Appendix

All the code is attached in this section.

```
Id3Algo.java
```

```java
package DataMining.Assignment;

import java.io.IOException;
import java.sql.SQLException;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
```

```
public class Id3Algo {

MysqlOps my;
List<String> attributes;
List<String> allAttributes;
String dataFile,label,entityName;
TreeNode root;

public Id3Algo() throws SQLException {
my = new MysqlOps();
my.connect();
attributes = new ArrayList<String>();
allAttributes = new ArrayList<String>();
entityName = "table1";
root = null;
}

// load the given data into the Database
public void LoadData(){
my.LoadDataIntoDB(dataFile,allAttributes);
}

public void setEntityName(String name){
entityName = name;
}

public List<String> getUniqueLabels() {
return my.GetUniqueValues(label,entityName);
}


public String getHighestGainAttribute(TreeNode node) {

List<String> variety = new ArrayList<String>();
List<String> varietyLabels = my.GetUniqueValues(label,entityName);
int totalCount = my.getCount(node.memberQualifier,entityName);
String initialQuery = ""; int sum1,sum2;
double score = 0,totalScore = 0,gain = Double.MAX_VALUE;
String decidedAttr = "";
for(String attr : attributes) {
variety = my.GetUniqueValues(attr,entityName);
for(String attrValue : variety) {
initialQuery = formDynamicQuery(node.memberQualifier,attr,attrValue);
sum1 = my.getCount(initialQuery,entityName);
sum2 = 0;
for(String vlab : varietyLabels ) {
sum2 = my.getCount(formDynamicQuery(initialQuery,label,vlab),entityName);
if(sum2 == 0)
score = 0;

else {
// calculate entropy
score = sum2/(double)sum1;
score = -1 * score * Math.log(score)/Math.log(2);
totalScore += score;
}
}
if(totalCount != 0)
totalScore = totalScore * (sum1/(double)totalCount);
else
totalScore = 0;
}
if(totalScore < gain) {
gain = totalScore;
decidedAttr = attr;
}
totalScore = 0;
}
return decidedAttr;
}

public String formDynamicQuery(String query,String label,String value){

if(query == null || query.length() == 0)
query = "1=1";
query += String.format(" AND %s = '%s'", label,value);
return query;
}

//update node with label values it is holding
public void getLabelCountOnNode(TreeNode node,String query) {
node.labelMap = new HashMap<String, Integer>();
List<String> variety = my.GetUniqueValues(label,entityName);
for(String attrValue: variety)
node.labelMap.put(attrValue,my.getCount(formDynamicQuery(query,label,attrValue),entityName));
}

//Build the decision tree
public void getClassificationTree(TreeNode node) {

if(attributes.size() > 0 && node != null) {
String query = node.memberQualifier;
node.partitionBasedOn = getHighestGainAttribute(node);
List<String> varietyLabels = my.GetUniqueValues(node.partitionBasedOn,entityName);
node.children = new ArrayList<TreeNode>();
String tempQuery;TreeNode tempNode = null;
int index;
for(String attrValue: varietyLabels) {
tempQuery = formDynamicQuery(query, node.partitionBasedOn, attrValue);
tempNode = new TreeNode(tempQuery,node);
getLabelCountOnNode(tempNode,tempQuery);
index = attributes.indexOf(node.partitionBasedOn);
if(index > -1)
attributes.remove(index);
tempNode.count = my.getCount(tempNode.memberQualifier,entityName);
```

```
getClassificationTree(tempNode);
node.children.add(tempNode);
attributes.add(node.partitionBasedOn);
}
}
}

// calculate conditional probability for attribute
public double getConditionalProbability(TreeNode node ,String query,String label) {

List<TreeNode> nodes = new ArrayList<TreeNode>();
node.findNodes(node, query, nodes);
int count = 0; double condProb = 0;
if(nodes.size() > 0) {
for(TreeNode item: nodes){
if(item.labelMap.containsKey(label))
count += item.labelMap.get(label);
}
}
if(count == 0)
condProb = node.labelMap.get(label)/(double)node.count;
else
condProb = count/(double)node.labelMap.get(label);
return condProb;
}

//if missing values then apply weighted conditional probability
public double getConditionalProbabilityForMissingValues(TreeNode node ,String query,String label) {

List<TreeNode> nodes = new ArrayList<TreeNode>();
node.findNodes(node, query, nodes);
int count = 0; double condProb = 0;
if(nodes.size() > 0) {
for(TreeNode item: nodes){
if(item.labelMap.containsKey(label))
count += item.labelMap.get(label);
}
}

if(count == 0)
condProb = node.labelMap.get(label)/(double)node.count;
else
condProb = count/(double)node.labelMap.get(label);
return condProb*count; // weighted conditional probability for missing values
}

// Naive bayes for predicting the class
public double applyNaiveBayes(TreeNode node,List<String> attrList,List<String> tuples,String labelVal
int i = 0;
double prob = 1,probTemp;
String temp; String attrValue;
List<String> possibleAttrValue = new ArrayList<String>();
for(String attr : attrList) {
```

```
attrValue = tuples.get(i);
if(attrValue.indexOf('*') > -1) {
possibleAttrValue = my.GetUniqueValues(attr, entityName);
probTemp = 0;
for(String possVal: possibleAttrValue) {
temp = String.format("%s = '%s'",attr,possVal);
probTemp += getConditionalProbabilityForMissingValues(node,temp,labelValue);
}
prob = prob * probTemp;
}
else {
temp = String.format("%s = '%s'",attr,attrValue);
prob = prob * getConditionalProbability(node,temp,labelValue);
}
i++;
}
prob = prob * node.labelMap.get(labelValue)/(double)node.count;
return prob;
}

//As probabilities may not sum up to 1, scale them ( because of Naive Bayes)
public void normalizeFinalProbability(List<Double> probList) {
List<String> labelVals = my.GetUniqueValues(label, entityName);
double sum = probList.stream().reduce(0d, (a,b) -> a+b);
int i = 0;
DecimalFormat df = new DecimalFormat("#.000");
for(double val:probList) {
System.out.print(labelVals.get(i));
System.out.print("\t");
System.out.println(df.format((double)val/sum));
i++;
}
}


public static void main(String[] args) throws IOException, SQLException{
//get the tree file
String filePath = args[0];
Id3Algo algo = new Id3Algo();
FileParser fileParser = new FileParser();
if(fileParser.FillInfoForAlgo(algo,filePath)) {

List<String> attrCopy = new ArrayList<String>();
for(String attr: algo.attributes)
attrCopy.add(attr);

TreeNode root = initialSetupForId3(algo);
//root.printPreOrder(root);
String testFile = args[1];
List<String> tuples = fileParser.getFileContent(testFile);
algo.getTestResults(root,attrCopy, tuples);
}
}
```

```
// Build the decision tree
public static TreeNode initialSetupForId3(Id3Algo algo) {
TreeNode root = new TreeNode("1=1",null); // need to assign this guy the total count
algo.getLabelCountOnNode(root, root.memberQualifier);
root.count = algo.my.getCount(root.memberQualifier, algo.entityName);
algo.getClassificationTree(root);
return root;
}

// for each test case get the test results
public void getTestResults(TreeNode node,List<String> attrCopy,List<String> tuples) {
List<String> attrValues;
List<Double> probList;
for(String tuple: tuples) {
attrValues = Arrays.asList(tuple.split("\\s+",-1));
probList = new ArrayList<Double>();
for(String labelValue: getUniqueLabels()) {
probList.add(applyNaiveBayes(node, attrCopy, attrValues, labelValue));
}
System.out.println(tuple);
normalizeFinalProbability(probList);
}
}
}
```

TreeNode.java

```
package DataMining.Assignment;

import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

public class TreeNode {

String memberQualifier; // query that defines the node contents
String partitionBasedOn;
int count;
TreeNode parent;
List<TreeNode> children;
Map<String,Integer> labelMap; // count of each label value


public TreeNode(String memberQualifier,TreeNode parent) {
this.memberQualifier = memberQualifier;
this.parent = parent;
}

public TreeNode findNode(TreeNode node,String query) {

if(node == null) return null;
if(query.equals(node.memberQualifier))
return node;
TreeNode temp;
if(query.indexOf(node.memberQualifier) > -1) {
for(TreeNode treeNode: node.children) {
temp = findNode(treeNode, query);
if(temp != null) return temp;
}
}
return null;
}

// print the pre-order form of tree
public void printPreOrder(TreeNode node) {

if(node == null) return;
String printStat = "Node Query : " + node.memberQualifier.trim() + " Count : " + node.count;
System.out.println(printStat.replace("1=1 AND","").replace("1=1",""));

if(node.labelMap != null){
for(Entry<String,Integer> pair: node.labelMap.entrySet()){
System.out.println(" label : " + pair.getKey() + " value : " + pair.getValue());
}
}
if(node.children != null) {
for(TreeNode treeNode: node.children){
printPreOrder(treeNode);
}
}
}


//get the nodes which obey the query
public void findNodes(TreeNode node,String query,List<TreeNode> nodes) {

if(node == null) return;
if(node.memberQualifier.indexOf(query) > -1) {

if(node.children == null || node.children.size() == 0) {
if(node.count > 0)
nodes.add(node);
}
}

if(node.children != null) {
for(TreeNode treeNode: node.children) {
findNodes(treeNode, query,nodes);
}
}
}
}
```

MysqlOps.java

```
package DataMining.Assignment;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;



public class MysqlOps {

Connection con = null;
Statement st = null;
ResultSet rs = null;
String url = "", user = "",passwd = "";


public MysqlOps() {

url = "jdbc:mysql://localhost:3306/Classification";
user = "root";
passwd = "quititdude";
}

public void connect() throws SQLException{

con = DriverManager.getConnection(url,user,passwd);

}

public void LoadDataIntoDB(String path,List<String> columns) {

Statement st;
try {

st = con.createStatement();
st.execute("DROP TABLE IF EXISTS table1");
st.execute(createTableQuery(columns));
String loadQuery = String.format("LOAD DATA LOCAL INFILE '%s' INTO TABLE table1 FIELDS TERMINATED BY
st.execute(loadQuery);

} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}

public String createTableQuery(List<String> columns) {
```

```
StringBuilder query = new StringBuilder();
query.append("CREATE TABLE table1(");
query.append(" tupleId MEDIUMINT NOT NULL AUTO_INCREMENT,");
query.append(String.join(" varchar(20)," , columns));
query.append(" varchar(20), primary key(tupleId) )");
return query.toString();
}



public List<String> GetUniqueValues(String column,String tableName) {

List<String> resList = new ArrayList<String>();
String query = String.format("SELECT DISTINCT(%s) from %s",column,tableName);
try {
st = con.createStatement();
ResultSet rs = st.executeQuery(query);
while(rs.next())
resList.add(rs.getString(1));
} catch (SQLException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
return resList;
}

public int getCount(String queryCond,String tableName){
Integer count = 0;
String query = String.format("SELECT count(*) from (%s) WHERE %s",tableName, queryCond);
try {
st = con.createStatement();
ResultSet rs = st.executeQuery(query);
while(rs.next()) {
count = rs.getInt(1); break;
}
} catch (SQLException e) {
e.printStackTrace();
}
return count;
}

public void createView(String queryCond,String view) {

String query = String.format("CREATE OR REPLACE VIEW %s AS SELECT * FROM table1 WHERE tupleId in (%s)
try {
st = con.createStatement();
st.executeUpdate(query);
} catch (SQLException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
```

```
public List<String> GetData(String tableName,String column) {

List<String> resList = new ArrayList<String>();
String query = String.format("SELECT %s from %s",column,tableName);
try {
st = con.createStatement();
ResultSet rs = st.executeQuery(query);
while(rs.next())
resList.add(rs.getString(1));
} catch (SQLException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
return resList;
}

}
```

FileParser.java

```
package DataMining.Assignment;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import com.mysql.jdbc.BufferRow;

public class FileParser {


public FileParser(){}


public Boolean extractColumns(List<String> temp,String fileName) throws IOException {
File file = new File(fileName);
if(file.exists()) {
InputStream in = new FileInputStream(file);
BufferedReader bufferReader = new BufferedReader(new InputStreamReader(in));
String columnHeader = bufferReader.readLine();
temp.addAll(Arrays.asList(columnHeader.split("\t")));
bufferReader.close();
return true;
}
return false;
}
public Boolean FillInfoForAlgo(Id3Algo algo,String filename){
Boolean status = false;
File file = new File(filename);
if(file.exists()) {
InputStream in;
try {
in = new FileInputStream(file);
BufferedReader bufferReader = new BufferedReader(new InputStreamReader(in));
algo.dataFile = file.getParent() + "/" + bufferReader.readLine();
algo.label = bufferReader.readLine().trim();
String columnHeader = bufferReader.readLine().trim();
algo.attributes.addAll(Arrays.asList(columnHeader.split("\\s+")));
bufferReader.close();
status = extractColumns(algo.allAttributes,algo.dataFile);
} catch (FileNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
return status;
}

public List<String> getFileContent(String fileName){
List<String> content = null;
File file = new File(fileName);
if(file.exists()) {
InputStream in;
BufferedReader bufferReader;
try {
content = new ArrayList<String>();
in = new FileInputStream(file);
bufferReader = new BufferedReader(new InputStreamReader(in));
String line;
while((line = bufferReader.readLine()) != null){
content.add(line.trim());
}
bufferReader.close();
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
}
return content;
}

}
```

CrossValidation.java

```
package DataMining.Assignment;

import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.stream.Collectors;

public class CrossValidation {

List<Integer> inputTupleIds;
MysqlOps mysql;

public CrossValidation() throws SQLException{

mysql = new MysqlOps();
mysql.connect();
}

public Map<Integer,List<Integer>> GetCrossValidationInput(int range,int numberOfGroups) {
Map<Integer,List<Integer>> testSet = new HashMap<Integer, List<Integer>>();
List<Integer> listOfInts;
int temp;
for( int i = 1; i <= range; i++){
temp = i%numberOfGroups+1;
if(testSet.containsKey(temp)) {
listOfInts = testSet.get(temp);
listOfInts.add(i);
testSet.put(temp,listOfInts);
}
else{
listOfInts = new ArrayList<Integer>();
listOfInts.add(i);
testSet.put(temp,listOfInts);
}
}
return testSet;
}

public static void main(String[] args) {

try {
CrossValidation crVal = new CrossValidation();
String filePath = args[0];
Id3Algo algo;
algo = new Id3Algo();
FileParser fileParser = new FileParser();
List<String> attrCopy = new ArrayList<String>();
if(fileParser.FillInfoForAlgo(algo,filePath)) {
for(String attr: algo.attributes)
attrCopy.add(attr);
algo.LoadData();
int range = crVal.mysql.getCount("1=1","table1");
int noOfSets = 3;

Map<Integer,List<Integer>> crMap =  crVal.GetCrossValidationInput(range,noOfSets );
List<Entry<Integer,List<Integer>>> entryList = new ArrayList<Map.Entry<Integer,List<Integer>>>(crMap.
List<Integer> trainingSet = new ArrayList<Integer>();
List<Integer> testSet = new ArrayList<Integer>();
String temp; TreeNode root;
for(int i = 0; i < noOfSets; i++) {
trainingSet.clear(); testSet.clear();
trainingSet.addAll(entryList.get(i).getValue());
trainingSet.addAll(entryList.get((i+1)%3).getValue());

temp = trainingSet.stream().map(num -> String.valueOf(num)).collect(Collectors.joining(", "));
crVal.mysql.createView(temp,"trainingView");
algo.setEntityName("trainingView");
root = Id3Algo.initialSetupForId3(algo);
root.printPreOrder(root);

testSet.addAll(entryList.get((i+2)%3).getValue());
temp = testSet.stream().map(num -> String.valueOf(num)).collect(Collectors.joining(", "));
crVal.mysql.createView(temp,"testView");

List<List<String>> tempTuples = new ArrayList<List<String>>();
for(String attr:attrCopy){
tempTuples.add(crVal.mysql.GetData("testView",attr));
}

String temp1;
List<String> tuples = new ArrayList<String>();
for(int  j = 0; j < range/noOfSets; j++){
temp1 = "";
for(List<String> list: tempTuples){
temp1 += " " + list.get(j);
}
tuples.add(temp1.trim());
}
algo.getTestResults(root, attrCopy, tuples);
}
}
} catch (SQLException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
}
```

5. References

[1] Stack Exchange. Bayesian posterior: mean vs highest probability - cross validated.

[2] Dewan Md Farid, Nouria Harbi, and Mohammad Zahidur Rahman. Combining naive bayes and decision tree for adaptive intrusion detection. *arXiv preprint arXiv:1005.4496*, 2010.