Lab 3: Supervised Learning

Due date: Monday, October 18, 11:59pm.

Lab Assignment

In this assignment, you are asked to implement a variant of C4.5 decision tree classifier which uses information gain measure to induce decision trees. Your implementation will not be domain-specific but will be restricted by the types of attributes that can be found in the datasets you are given. You will run two sets of tests for your implementation: one will test the accuracy of the classifiers you are building, and the other will serve as the endurance tests for your implementation.

Assignment Preparation

This is a pair programming assignment. Pick your partner at will.

Datasets

The assignment uses a number of synthetically generated datasets. The main dataset for this assignment is called `ELECTIONS` and is described in detail below. Other datasets may be made available to you as well.

All datasets are constructed in the same way and have a number of file formats associated with them. In particular, the following data formats are used in this lab:

- **Domain description.** The structure (schema) of each dataset is described in an XML file. The format for domain description files is specified below. Your program will take these files as input in order determine the attributes of the dataset, and the various attribute values.

```
<!ELEMENT domain (variable+ Category)>
<!ELEMENT variable ( group+ )>
<!ELEMENT category ( choice+)>
<!ELEMENT group (EMPTY)>
<!ELEMENT choice (EMPTY)>

<!ATTLIST variable name CDATA #REQUIRED>
<!ATTLIST group name CDATA #REQUIRED
                p CDATA #IMPLIED>
<!ATTLIST Category name CDATA #REQUIRED>
<!ATTLIST choice name CDATA #REQUIRED
                 type CDATA #REQUIRED>
```

Figure 1: The DTD for the dataset domain description XML files.

- **Words data format.** This is a CSV (comma-separated values) file format for representing the actual dataset. Data in this format will be presented in its raw form, with the **actual** values of each attribute being reported. For example, if a dataset contains an attribute `Gender` with possible values `"Male"` and `"Female"`, the CSV file in the **words** format will include the actual values `"Male"` and `"Female"` in. This format is not necessarily convenient for processing, but is more human readable of the two CSV data formats provided to you.
- **Numbers data format.** This is a CSV file format for representing the datasets. Data in this format is presented by encoding each attribute value with an integer value in the range from 1 to the size of the domain of the attribute. For example, if a `Gender` attribute has two values, `"Male"` and `"Female"`, these values will be coded as 1 and 2 (in the order in which the values appear in the domain description file). This format is more convenient for automated processing, but is not human-readable.

Domain Description File Format

For each dataset you encounter in this lab, you are provided with an XML file, describing its domain. The DTD of the domain XML files is shown in Figure **??**.

A brief explanation of the file format follows.

Root element. The root element of the domain XML files is the `<domain>` element.

Structure. The domain described by the XML file is essentially a relational schema which includes a *special category attribute*. For our lab, one more restriction is in order: all dataset columns have a finite (and usually, small) number of possible values. The XML file lists each attribute in turn, enumerates their values and identifies the category variable and the list of categories.

The `<domain>` element includes one or more `<variable>` elements describing the columns of the dataset, followed by a single `<Category>` element describing the category variable.

Columns. Dataset columns are described by the `<variable>` elements. Each `<variable>` element must include the `name` attribute, which provides the name of the column. Each `<variable>` element contains a list (one or more) of possible values, represented as `<group>` elements.

Column values. A `<group>` element describes a single value a dataset column take. The element is empty, but has two attributes. The mandatory `name` attribute specifies the value itself. The optional `p` attribute provides the probability of occurrence of this value in the dataset[1].

Category variable. The category variable is introduced via the `<Category>` element. A single `<Category>` element is allowed in each domain XML file. The mandatory `name` attribute stores the name of the category variable. The content of the `<Category>` element is one or more `<choice>` elements, each encoding an individual category (class).

Categories. Individual categories/classes are described by `<choice>` elements. These elements have empty content, but come with two mandatory attributes. The `name` attribute contains the name of the category used in the **words** formatted CSV filed. The `type` attribute provides the numeric label for it, that will be used in the **numbers** formatted CSV files.

An example of a simple domain, containing two data columns, `Gender` with values `"Male"` and `"Female"`, and `Major` with possible values `"CS"` and `"ENG"` and a category variable `"Genre"` with category labels `"SCIFI"` and `"Romance"` is shown in figure **??**,

Your program will take XML files specifying domains as input. Your code is responsible for parsing the files and using the information contained in them in preparing the computed decision trees.

ELECTIONS dataset

The main dataset you will be working with is the `ELECTIONS` dataset. As such, it is described in detail here. Other datasets may be released throughout the lifetime of the assignment. Their descriptions will be supplied using `README` files and the XML domain description files.

[1] This attribute is used in dataset generation, but you can ignore it in your work.

```xml
<?xml version="1.0" encoding="utf-8"?>
<domain>
    <variable name = "Gender">
       <group name = "Male" p =".5" />
       <group name = "Female" p =".5" />
    </variable>
    <variable name = "Major">
       <group name = "CS" p = ".40" />
       <group name = "ENG" p = ".60" />
    </variable>
    <Category name = "Genre">
       <choice name = "SCIFI" type ="1" />
       <choice name = "Romance" type ="2" />
    </Category>
</domain>
```

Figure 2: An XML file describing a simple dataset.

The `ELECTIONS` dataset describes the voting preferences of individual voters in the 2008 US Presidential elections. *The dataset is synthetic, i.e., it was artificially created, but the information contained in the dataset roughly follows the Presedential election exit polls.*

Each record in the dataset describes a variety of characteristics of an individual voter. The category variable of the dataset is Vote: the choice of the voter in the Presidential elections. For simplicity, vote has only two values: `Obama` and `McCain`.

A record in the `ELECTIONS` dataset has the following format:

```
(Id, Party, Ideology, Race, Gender, Religion, Income, Education, Age, Region,
BushApproval,Vote)
```

1. `Id`: unique identifier of the record. Used to identify individual voters. This attribute shall be excluded from decision tree induction.

2. `Party`. Political party of the voter. The values are:

Numeric Value	Actual Value	XML Value
1	`Democratic`	`"Democratic"`
2	`Republican`	`"Republican"`
3	`Independent`	`"Independent"`

3. `Ideology`. Political ideology of the voter. The values are:

Numeric Value	Actual Value	XML Value
1	`Liberal`	`"Liberal"`
2	`Moderate`	`"Moderate"`
3	`Conservative`	`"Conservative"`

4. `Race`: race of the voter. The values are:

Numeric Value	Actual Value	XML Value
1	Black (African-American)	"Black"
2	White (Caucasian)	"White"
3	Other	"Other"

5. `Gender`: the gender of the voter. The values are

Numeric Value	Actual Value	XML Value
1	Male	"Male"
2	Female	"Female"

6. `Religion`: religion of the voter. The values are:

Numeric Value	Actual Value	XML Value
1	Protestant	"Protestant"
2	Catholic	"Catholic"
3	Other	"Other"

7. `Income`: the income bracket (annual income) of the voter's family. The values are:

Numeric Value	Actual Value	XML Value
1	Less than $30,000	"Less than 30000"
2	$30,000 --- $49,999	"30000-49999"
3	$50,000 --- $74,999	"50000-74999"
4	$75,000 --- $99,999	"75000-99999"
5	$100,000 --- $149,999	"100000-149999"
6	Over $150,000	"150000+"

8. `Education`. The highest level of education for the voter. The values are

Numeric Value	Actual Value	XML Value
1	High school diploma or less	"H.S. diploma or less"
2	Undergraduate study/degree	"College"
3	Post-graduate study/degree	"Post-Grad"

9. `Age`. The age group of the voter. The values are:

Numeric Value	Actual Value	XML Value
1	18 -- 29	"18-29"
2	30 -- 44	"30-44"
3	45 -- 64	"45-64"
4	65 and above	"65+"

10. `Region`: the geographic region where the voter lives. The values are:

Numeric Value	Actual Value	XML Value	States
1	Northeast	"Northeast"	ME, NH, VT, MA, RI, CT, PA, NY, NJ, DE, MD, DC
2	South(east)	"South"	VA, WV, KY, NC, SC, TN, GA, FL, AL, MS, LA, AR, TX, OK
3	Midwest	"Midwest"	OH, IN, MI, IL, MO, IA, MN, WI, ND, SD, NE, KS
4	West	"West"	MT, ID, WA, AK, HI, WY, CO, UT, OR, NV, NM, AZ, CA

11. `Bush Approval`: information on whether or not the voter approves of George W. Bush in his capacity as the President of the US. The values are:

Numeric Value	Actual Value	XML Value
1	Approve	"Approve"
2	Disapprove	"Disapprove"

12. `Vote`. The choice of the voter in the 2008 Presidential Election. This is the **category** variable in this dataset. The values are:

Numeric Value	Actual Value	XML Value
1	`Barack Obama`	`"Obama"`
2	`John McCain`	`"McCain"`

You are provided with the following CSV files.

Set	Filename	Tree	Format	Size
1	`tree02-20-words.csv`	*small tree*	words	20 voters
	`tree02-20-numbers.csv`	*small tree*	numbers	20 voters
2	`tree02-100-words.csv`	*small tree*	words	100 voters
	`tree02-100-numbers.csv`	*small tree*	numbers	100 voters
3	`tree01-100-words.csv`	*large tree*	words	100 voters
	`tree01-100-numbers.csv`	*large tree*	numbers	100 voters
4	`tree01-1000-words.csv`	*large tree*	words	1000 voters
	`tree01-1000-numbers.csv`	*large tree*	numbers	1000 voters

Formats. There are two formats in which data files are represented. Files in the words format contain the XML values of each attribute from the tables above. Files in the numbers format represent the same data using Numeric values from the tables above.

For example, here is a record in the words format and the same record in the numbers format:

```
3,Independent,Moderate,White,Male,Catholic,30000-49999,College,45-64,Northeast,Approve,McCain

3,3,2,2,1,2,2,2,3,1,1,2
```

These files were produced using two different decision trees. *Small tree* is a decision tree constructed out of a relatively small subset of all attributes. *Large tree* is a decision tree constructed out of all or almost all attributes (not every attribute appears on every path, though). The files within the same "set" contain the same records: thus, `tree02-20-words.csv` and `tree02-20-numbers.csv` contain the same 20 records in two different formats. Records in files from different sets are different, so, overall, you have access to 120 voter records for the *small tree* and 1100 voter records for the *large tree*.

Note: Each CSV file contains three header lines. The first line contains the comma-separated list of names, the second line — the information about the domain size for the domain of each attribute. The third line specifies (by name) the category attribute.

For the `ELECTIONS` dataset, the training set file header looks as follows:

```
Id,Political Party,Ideology,Race,Gender,Religion,Family Income,Education,Age,Region,Bush Approval,Vote
-1,3,3,3,2,3,6,3,4,4,2,2
Vote
```

Conventions: If the second line contains `-1`, the corresponding attribute is a *unique Id* and should not be used in decision tree induction.

Note: All attributes in the `ELECTIONS` dataset are *categorical* for the purposes of this assignment. (Technically speaking, `Income` is ordinal, and `Education` and `BushApproval` can also be viewed as ordinal, but this will not affect your implementation). Thus, your program will be **required** to handle properly categorical attributes, but proper handling of numeric attributes is not required.

The files can be downloaded from the course web page:

http://www.csc.calpoly.edu/∼dekhtyar/466-Fall2010/labs/lab03.html

The full description of the `ELECTIONS` dataset and all the files are available from the datasets wiki:

https://wiki.csc.calpoly.edu/datasets/wiki/Voting

Decision Tree Induction

Your assignment is *thee-fold*. You will

1. Write an implementation of the C4.5 decision tree induction algorithm with information gain/information gain ratio measure. Your implementation shall take as input the data from the `ELECTIONS` dataset and output an XML representation of the decision tree (the format is described below).
2. Write the classification program, that takes as input an XML representation of a decision tree, a CSV file of test cases, and outputs the category for each test case.
3. Evaluate the performance of your implementation using techniques learned in class. Perform accuracy and endurance analyses of your implementation.

Task 1: C4.5 Decision Tree induction

Program Input

You will write a program `InduceC45.java` implementing the C4.5 classifier that uses either information gain or information gain ratio measures (or both) to determine the next attribute on each step of the decision tree construction process.

Your program shall be run as follows:

```
java InduceC45 <domainFile.xml> <TrainingSetFile.csv> [<restrictionsFile>]
```

Here, `<domainFile.xml>` is the name of the XML file containing the domain description for the dataset, `<TrainingSetFile.csv>` is the name of the input training set file. `<restrictionsFile>` is the name of text file. This text file will contain a single vector. The size of the vector is equal to the number of columns in the dataset without the category variable. Each element of the vector is either 0 or 1.

The meaning of the restrictions file is straightforward. This (optional) file indicates which attributes of the dataset to use when inducing the decision tree. A value of 1 means that the attribute in the corresponding position is to be used in the decision tree induction; a value of 0 means that the attribute is to be omitted. For example, to induce a decision tree using only `Race`, `Gender` and `Region` attributes of the `ELECTIONS` dataset, we can create a restrictions file `restrictions01.csv` with the following content:

```
1,0,0,1,1,0,0,0,0,1,0
```

(Note, that we also included `Id` in the dataset, although it won't be used for decision tree induction.)

If `<restrictionsFile>` is absent from the command line, your program shall use all non-ID attributes of your dataset to induce the decision tree.

Program Flow

C4.5 Your program shall read and parse input files. It shall correctly identify the domains for all columns, and the list of available categories. It shall also correctly determine the training set. It shall then implement the `C4.5` algorithm for decision tree induction.

Information Gain. Because the target datasets contain mostly categorical attributes with small domains (only one attribute has the domain of size 6), it is expected that you will implement the computation information gain measure to determine the splitting attribute on each step of the decision tree induction process.

Information Gain Ratio. However, if you determine that information gain does not lead to robust results, feel free to implement instead (or, better yet, *in addition to*) the information gain ratio measure.

Internal data structures. The design of the internal data structures in support of decision tree induction is left up to you. Essentially, you need to determine the following:

- *How to store the input training set.* Possible options (this list is NOT exhaustive!) include storing all data in an array or a list in main memory; using Oracle or MySQL DBMS to store the records, and JDBC to retrieve necessary information; no memory-resident storage at all — the .csv file is reread and re-parsed on every scan.

- *How to split a training set.* On each step of the `C4.5` process training sets get split. Because this is such a common operation, your training set representation mechanism should come with a `split` method that separates it into two or more pieces.

- *How to store the emerging decision tree.* You will have to design an internal representation of the decision tree, complete with the traditional operations for of insertion of new node and traversal. Note, that this representation will also be needed in the second task of this assignment.

Program Output

Once you generate the final decision tree, your program shall output it to `<stdout>` in an XML format described below.

Note: There exists a general XML standard called `PMML` for representation of decision trees (as well as other data mining information). However, this standard is overly complex for the purposes of our assignment. Instead, your program will generate an XML document in a simplified format. The structure of the XML document will mimic the structure of the decision tree it is designed to represent.

Output Format. The DTD of the XML format for the output decision tree is shown in Figure 1. The XML description of decision trees consists of four XML elements.

- **Root.** The root of the XML document is an XML element `<Tree>`. This element has an attribute `name` associate with it. The name of the tree is not used in processing, only in identification. Feel free to assign a name in any way you deem necessary.

- **Nodes.** The root element will have a single child element `<Node>`. The `<node>` element has an attribute `var` whose value is the name of one attribute from the dataset. For the `ELECTIONS` dataset, the name will from the following list:

```
"Political Party"   "Ideology"    "Race"     "Gender"    "Religion"
"Family Income"     "Education"   "Age"      "Region"    "Bush Approval"
```

- **Edges.** Each `<node>` element has as its chldren a number of `<edge>` elements. *Typically*, the number of children of a `<node>` element is equal

```
<!ELEMENT Tree (node)*>
<!ATTLIST Tree
    name CDATA #IMPLIED
  >

<!ELEMENT node (edge)*>
<!ATTLIST node
    var CDATA #IMPLIED
  >

<!ELEMENT edge ( (node*)|(decision)*)>
<!ATTLIST edge
    var CDATA #IMPLIED
    num CDATA #IMPLIED
>

<!ELEMENT decision EMPTY>
<!ATTLIST decision
    p CDATA #IMPLIED
  choice CDATA #IMPLIED
    end CDATA #IMPLIED
      >
```

Figure 3: DTD for the decision tree XML documents.

to the number of possible values in the domain of the dataset attribute specified as the value for its `var` attribute (e.g.,`<node var="Education">` is expected to have three children `<edge>` nodes, while `<node var="Region">` is expected to have four).

Each `<edge>` element has two attributes: `var` and `num` attribute. The value of the `var` attribute of an `<edge>` element represents one of the values in the domain of the column represented by the partent `<node>` element, while the the value of the `num` attribute contains the numeric label assigned to this value, as shown, for example, in the following XML fragment:

```
<node var="Party">
  <edge var = "Democratic" num="1">...</edge>
  <edge var = "Republican" num="2">...</edge>
  <edge var = "Independent" num ="3">...</edge>
</node>
```

The **exact values** the the `var` attribute for each domain value are found in the input domain XML file. The numeric values corresponding to them are assigned in the order of their appearance in the XML file.

- **Recursion.** Each `<edge>` node will contain either exactly one `<node>` element child or exactly one `<decision>` element child. The `<node>` element will have the structure as described above. No two `<node>`

elements on the same path from the root node may have the same value for the `var` attribute.

- **Leaf nodes.** The leaf nodes in the XML document are `<decision>` element nodes. A single `<decision>` element node will appear as a child of an `<edge>` node. Each path in the XML tree will be terminated with a `<decision>` node. This node has three attributes. The `end` attribute takes values from "1" to the total number of classes, while the `choice` attribute has values from the class labels described by `<choice>` elements in the domain XML file. In case of the `ELECTIONS` dataset, `end` can take values 1 (meaning the path leads to an Obama voter) or 2 (meaning the path leads to a McCain voter), while the `choice` attribute takes accordingly the values `"Obama"` and `"McCain"`.

  The third attribute is `p`. This attribute represents the probability that a record matching the path to the decision node will belong to the specified class. This attribute can be omitted in your output, set to 1, or can be set to the actual probability computed by the **C45 Algorithm**.

Example. The following is a simple example of a decision tree in the XML format:

```
<Tree name = "test">
   <node var ="Gender">
       <edge var ="Female"  num="2">
           <node var = "Bush Approval">
              <edge var = "Approve" num="2" >
                  <decision end = "2" choice = "McCain" p = "0.9"/>
              </edge>
               <edge var = "Disapprove" num="1">
                   <decision end = "1" choice="Obama" p = "0.95"/>
               </edge>
            </node>
        </edge>
        <edge var = "Male" num="1">
            <node var = "Ideology">
              <edge var = "Liberal" num = "1">
                  <decision end = "1" choice ="Obama" p = "0.99"/>
              </edge>
               <edge var = "Moderate" num="2">
                   <decision end = "1" choice = "Obama" p = "0.7"/>
               </edge>
               <edge var = "Conservative" num ="3">
                   <decision end = "1" choice = "McCain" p = "0.95"/>
               </edge>
            </node>
        </edge>
    </node>
</tree>
```

This XML file represents the decision tree depicted in Figure 2.

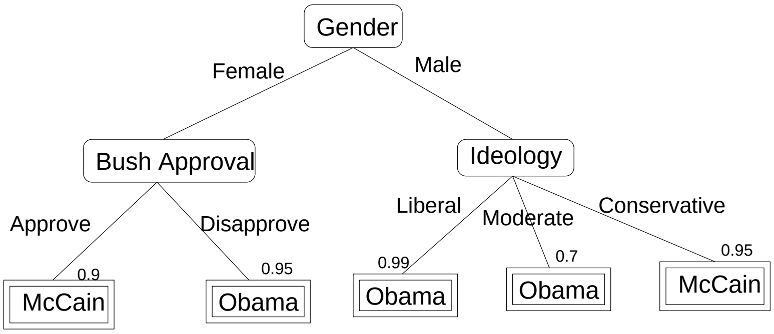

Figure 4: A simple decision tree.

Task 2: Classification

Write a program `Classify.java` that will take as input the XML description of a decision generated by your `InduceC45.java` program and a CSV file of records to be classified and outputs the classification result for each vector.

`Classify.java` is run as follows:

```
java Classifier  <CSVFile> <XMLFile>
```

If the input CSV file is a training set (i.e., if it comes with the category attribute), your program shall

- *ignore* this attribute while classifying the rest of the record.
- *compare the result* of classifying the record with the actual class label.
- *keep track* of the number of classification errors found while classifying all records in the file.
- *at the end of the run report*:
  1. total number of records classified;
  2. total number of records correctly classified;
  3. total number of records incorrectly classified;
  4. overall accuracy and error rate of the classifier.

Optionally, you may also output the confusion matrix for the run.

If the input CSV file does not contain the cateogory attribute, then your program only needs to output predicted class labels for each record.

Note: In general, your program shall for each input record, print the record and then print the class label for it. However, you may want to include a `silent run` option in your program, that will report only `rowID` and class label pairs (possibly - comma-separated), but won't print full contents of each record. This may by useful elsewhere in the lab.

Task 3: Evaluation

You shall perform *cross-validation analysis* of the accuracy of your classifiers using the *training set data* made available to you.

To that extent, you shall implement a program `Validation.java`, which will take as input the training test file, the optional *restrictions file* and an *integer number* n specifying how many-fold the cross-validation has to be.

For simplicity we assume that $n = 0$ represents no cross-validation (i.e., use entire training set to construct a single classifier), while $n = -1$ represents **all-but-one cross-validation**.

The `Validation.java` program shall perform the n-fold cross-validation evaluation of your `InduceC45` implementation. It shall produce as output the following information:

1. The **overall** confusion matrix.
2. The **overall** recall, precision, pf and f-measure (compute these numbers with respect to recognizing **Obama's voters**).
3. **Overall** and **average** accuracy (two-way) and error rate of prediction.

Note: Overall measures are reported for the entire result of cross-validation. E.g., if you are running s 2-way cross-validation on 20 records, **overall** accuracy is computed by adding up all correct predictions from all both iterations of the cross-validation process, and dividing this number by 20. The **average** accuracy is computed as the mean of the accuracies achieved on the first and the second iterations of the cross-validation process.

Submission Instructions

Use the handin tool to submit your files. Each group submits exactly one copy of all materials. You will submit the following files:

- `README`. Shall contain the names and email addresses of all students in the team. Also, put any specific instructions and notes in this file. (e.g., if you choose a different implementation language, include translation/running instructions).
- `InduceC45.java`, `classifier.java`, `Evaluate.java` and any supplementary files. (if you choose to use a different programming language, submit all source files in that language).
- The output of `Evaluate.java` on the `tree01-1000-numbers.csv` (or `tree01-1000-words.csv`) input with 10-fold cross-validation. Dump the output into a text file and submit.

Archive your files into a `.zip` or a `.tar.gz` archive.

To submit use the following command:

```
$ handin dekhtyar-grader lab03 lab03.ext
```