Fall 2021	**CSC 466:** Knowledge Discovery from Data	Alexander Dekhtyar

Lab 3: Supervised Learning: Classification Algorithms

Due dates:
Part 1: Tuesday, October 19, 10:00pm
Part 2: Tuesday, October 26, 10:00pm

Lab Assignment

This is a pair programming lab. If you want to, you can keep your partner from Labs 1 and/or 2. After this lab, we will force a change of partners. A team of three people will incur additional incremental tasks as part of this lab.

The lab consists of two parts, each with its own set of deliverables and tasks to complete. Both parts of the lab are assigned at the same time to allow each team additional flexibility in working on the assignments, and to give you all a road map of all classification works you are to complete for this course.

For Part 1 of the Lab you have to implement the version of C4.5 decision tree induction algorithm that works only with categorical independent variables. You shall

1. implement C4.5 in your language of choice (Python or Java are suggested languages but you can pick any language)
2. output the decision tree in a "proprietary" format
3. implement the classification algorithm that uses the output of your C4.5 implementation
4. implement cross-validation
5. run your implementation on three datasets, report results.

For Part 2 of the Lab you shall

1. extend your implementation of C4.5 to allow your program to handle numeric attributes
2. implement a Random Forest classification algorithm that uses your C4.5 implementation to build individual decision trees.
3. implement the K Nearest Neighbors classifier
4. test and run your implementations on several datasets
5. hypertune your Random Forest classifier to produce highest accuracy results on each dataset.
6. report results of your investigations

Datasets

We will be using a number of datasets from the UCI (Univeristy of California at Irvine) Machine Learning Dataset Repository. Each of the datasets has a long history of being used for classification.

Below is a list of the datasets provided with brief description of each. Please note, for most datasets, a file with additional information is also made available. In the interest of saving space in this document, I will not repeat the information found in those files - you should consult them in order to determine the variables provided to you and their domains.

All datasets can be downloaded from the Lab 3 data page:

http://users.csc.calpoly.edu/~dekhtyar/466-Fall2019/labs/lab03.html

We also provide links to the UCI Repository pages for each dataset. Please note, that in some cases, we have changed file names, and, modified the files themselves to make parsing easier (You can, however, use any files as input). You are also allowed to modify the format of input files to your liking (adding or removing header lines, etc...)

Data Format

To give you a little bit of help when parsing data, the datasets are all provided in a modified CSV format. The data itself is comma-separated rows, like standard CSV format. However, the first three lines of the file have to be parsed separately.

The **first line** contains the names of all columns (a standard practice when creating CSV data files).

The **second line** provides information about the domain of each variable. The line contains a comma-separated list of integer numbers. The meaning of the numbers is as follows:

-1	The column is a `rowId`. It should be ignored by classificaiton algorithms.
0	The column is a numeric variable.
$n > 0$	number of possible values of the variable.

The **third line** contains the name of the *class variable*. For all problems studied in this lab, the class variable is unique for each dataset.

For example, the following header of your data file:

```
No,Name,Income,Sex,Education,Occupation,Rating
-1,-1,0,2,5,7,3
Rating
```

describes a data file that contains seven columns, of which the first two (`No` and `Name`) are essentially row Ids (or otherwise contain meta data not necessary for classification), the next four columns (`Income`, `Sex`, `Education` and `Occupation`) represent independent variables, with `Income` being a numeric attribute, while the remaining three attributes having two, five, and seven different levels (values) respectively[1]. The last column, `Rating` is a class variable, and it has three different levels (values).

Part 1 Datasets

Datasets for Part 1 contain only categorical independent variables. Such datasets are somewhat rare – in many classification tasks numeric attributes present, and in some classification datasets contain only numeric attributes. Nevertheless, for pedagogical reasons (and because in some situations, e.g., survey analysis purely categorical datasets do arise), we limit Part 1 of the Lab to categorical data only.

All datasets are available as CSV files where the first row lists the names of the attributes. Some datasets have unique row Ids - where is it the case, this is specified explicitly. Your classification algorithms must ignore rowId columns.

Balloons Dataset

This is a very simple toy dataset that is very convenient to use for debugging purposes. The Balloons dataset[2] documents observations of people performing some activities with inflatable balloons. Each observation consists of four independent variables documenting the color of the balloon (yellow or purple), the size of the balloon (small or large), the activity performed with the balloon (stretch or dip), and the age of the person who handled the balloon (adult or child). The *class variable* is the state of the balloon after this action: inflated or not.

The dataset contains four small data files. Each file contains a subset of overall data that satisfies a single clear-cut classification criterion. The classification criteria are specified in the dataset description file `balloons.names`.

As such, you can use this dataset to test your code, and see if your C4.5 implementation has successfully built the appropriate decision tree for each case. Note, that each dataset is sufficiently small, that you can

[1] Historically, government and businesses used binary attributes to record sex and/or gender. As most available datasets come from such sources, many legacy datasets use binary identifiers for sex and/or gender and also occasionally use "sex" and "gender" as synonyms.

[2] https://archive.ics.uci.edu/ml/datasets/Balloons

compute *Information Gain* and *Information Gain Ratio* measures manually — this gives you (sort of you yourself synthesizing own data) enough power to do some unit-testing of your implementation.

Mushroom Dataset

The Mushroom dataset[3] contains over 8,000 descriptions of physical appearance of gilled mushrooms from 23 species of the *Agaricus and Lepiota* family. Some species are edible, some are poisonous or not edible. The dataset, stored in the `agaricus-lepiota.data.csv` file has 22 categorical attributes describing the physical characteristics of each mushroom (e.g., cap size, cap color, odor, gill size, stalk size, veil type, and so on). The `agaricus-lepiota.names.txt` file contains the full description of each column, including all possible values. The first attribute in the data file is the class variable which takes two values: `"p"` for poisonous/not edible mushrooms, and `"e"` for edible mushrooms.

Your goal is to predict whether a given mushroom is edible or poisonous/not edible.

Nursery Dataset

The Nursery dataset[4] contains information about decision-making process for recommending children for nursery care. Each data point contains a unique combination of values collected during an intake interview with parents, documenting family status and circumstances. The class variable is the recommendation the application resulted in. There are five levels of recommendations: `not recommended, recommended, very recommended, priority, special priority` (with `recommended`, strangely enough being an outlier class with only two instances in the dataset).

Of the three fully categorical datasets, Nursery is the most complex: its attributes often have more than two different values, and for this dataset perfect classification using decision trees may be either impossible, or *hairy* - i.e., result in large decision trees (this is a good dataset to watch for overfit in).

Part 1 Tasks

Task 1: C4.5 Decision Tree induction

Program Input

You will write a program `InduceC45.java` or `InduceC45.py`[5] 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 <TrainingSetFile.csv> [<restrictionsFile>]
```

or

```
python3 InduceC45 <TrainingSetFile.csv> [<restrictionsFile>]
```

Note on Python development. You are welcome to develop your Python code using Jupyter notebooks. **However, your deliverables shall be Python programs** that can be run from command line, without Jupyter.

[3] https://archive.ics.uci.edu/ml/datasets/Mushroom

[4] https://archive.ics.uci.edu/ml/datasets/Nursery

[5] This assignment assumes Java or Python as a programming language of choice w/o loss of generality. Feel free to implement in any language you want as long as you provide compilation/running instructions.

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

(Note, the restrictions file is a bit of an atavism from the previous versions of this lab, but it still may be a useful way to limit the columns used in a specific decision tree induction process. This might come in handy when you are implementing the Random Forest classifier).

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 class labels (aka 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 for Part 1 contain mostly categorical attributes with small domains, it is expected that you will implement the computation of the 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*) 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. If implementing in Python, pandas is your friend.
- *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 a JSON format described below.

Output Format. The decision tree will be represented as a single JSON document. The document shall contain a single element named `node` and additional metadata. The contents (value) of this `node` element shall represent the entire decision tree.

The JSON object has the following syntax:

```
{
  dataset:<fileName>,
  node: {
        var: <varName>,
        edges: [<listOfEdges>]
```

```
    }
}
```

Here, <varName> is the name of one of the columns in the dataset representing an independent variable. <fileName> is the name of the file for which the decision tree is built.

<listOfEdges> is a list of edge elements. Each edge element has the following syntax

```
edge: {
       value: <varValue>,
       node: <nodeDescription>
      }
```

or

```
edge: {
       value: <varValue>,
       leaf: <leafNode>
      }
```

The former syntax corresponds to an edge leading to another decision node. The latter syntax corresponds to an edge leading to a decision node in the leaf of the tree. Here <varValue> is a value of the independent variable listed in the parent node element. <nodeDescrpition> has been discussed above.

The leaf element has the following syntax:

```
leaf: {decision: <classValue>,
       p: <probability>}
```

Here, <classValue> is a value of the class variable, and <p> is the probability that an data point that matches the leaf node belongs to the labelled class.

In addition to these elements, your tree structure can include any additional JSON elements if you want to store additional information in the root of the tree, nodes, edges, or leaves of your decision tree.

Example. The following is a simple example of a decision tree from the Nursery dataset in the JSON format:

```
{ dataset: "nursery.csv"
node: { var: "finance",
        edges: [ edge:{value:"convenient",
                       node:{var:"parents",
                             edges: [
                                    edge:{value:"usual",
                                          leaf:{decision: "not_recom",
                                                p: 0.74
                                               }
                                          },
                                    edge:{value:"pretentious",
                                          leaf: {decision: "priority",
                                                 p: 0.78
                                                 }
                                         },
                                    edge:{value:"great_pret",
                                          leaf:{decision: "spec_prior",
                                                p: 0.9
                                                }
                                          }
                                    ]
                              }
                       },
                edge:{value:"inconv",
```

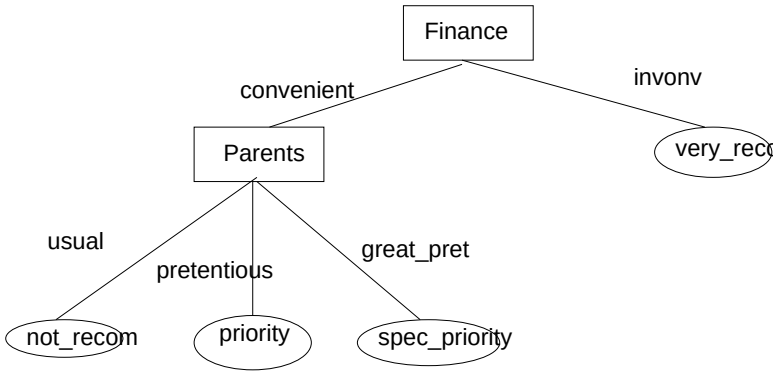

Figure 1: Decision tree for which the sample JSON code was presented.

```
                leaf:{decision: "very_recom",
                      p: 0.8}
               }
          ]
     }

}
```

This JSON object represents a decision tree shown in Figure 1 with the root `finance`. `finance="inconv"` immediately leads to `"very_recom"` class. `finance="convenient"` leads to a new decision node, this time labelled with `"parents"` variable. Here, `parents="usual"` yield `"not_recom"` class, `parents="pretentious"` yield `"priority"` class, and `parents="great_pret"` yields `"spec_prior"` class label assignment.

Task 2: Classification

Write a program `classify.java`/`classify.py` that will take as input the JSON description of a decision tree generated by your `InduceC45.java`/`InduceC45.py` 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> <JSONFile>
```

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. (I highly recommend that you output the confusion matrix of the run. It will help you with other tasks).

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`/`validation.py`, which will take as input the training file, the optional *restrictions file* and an *integer number* n specifying how many-fold the cross-validation has to be.

$n = 0$ 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. See Submission section at the end of this document for submission instructions for Part 1

Submission Instructions

Part 1

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).
- Your programs: `InduceC45`, `classifier`, `evaluate`, and any supplementary files.
- The output of `evaluate` on all CSV files from the three datasets for Part 1 of the lab (Balloons, Nursery, Mushrooms). Each file shall be named `<dataFile>-results.out`, where `<dataFile>` is the name of the CSV file (without the `csv` extension). E.g., for the Nursery dataset, your output file name is `nursery-results.out`.

Archive your files into a `.zip` or a `.tar.gz` archive.
To submit use the following commands:
Section 01:

```
$ handin dekhtyar lab03-1-466-01 <files>
```

Section 03:

```
$ handin dekhtyar lab03-1-466-03 <files>
```