Lab 1: Decision Trees (`sklearn Version`)

Evgueni N. Smirnov
smirnov@maastrichtuniversity.nl

4 November 2020

1. Introduction

Given a classification problem, you need data, classification models that are suitable for the data, and a machine-learning library that contains algorithms capable of training these models. In this lab you will study two well-known classification problems. You will train classification models for these problems based on decision trees. You will use `sklearn` as a machine-learning library that provides algorithms to train and test these models. Please check **Appendix A** with basic commands in `Python`, `pandas`, and `sklearn` that you need to know to execute the lab.

2. Classification Problems

The classification problems under study are two:

- Diabetis classification problem, and
- Glass classification problem.

A brief explanation of the data sets for these two problems are provided in **Appendix B**. Please note that the data for these two problems are given in the `csv` format.

3. Training Algorithms in `sklearn`

To train classifiers will use one learning algorithm `DecisionTreeClassifier` provided in `sklearn`. To make this algorithm compatible with ID3 decision trees, set its parameter `criterion` to "`entropy`". The experiments will be performed with two types of decision trees: one-level decision trees and multi-level decision trees.

- for one-level decision trees set the option `max_depth` of `DecisionTreeClassifier` set to 1;
- for multi-level decision trees set the option `max_depth` of `DecisionTreeClassifier` set to `None`;

4. Lab Tasks

a. Study the classification problems (given in Section 2) using the info from Appendix B.

b. Study `sklearn`***:*** you need to study `DecisionTreeClassifier` and `sklearn.model_selection` module (the latter allows us to estimate training accuracy rate and hold-out accuracy rate). A basic info is provided in Appendix A.

c. Train one-level decision trees and multi-level decision trees on the two data sets. Determine the accuracy rates of the resulting classifiers using the training set and hold-out validation[1]. Explain why there is a difference in the accuracy rates. Compare one-level decision trees and multi-level decision trees in terms of explainability.

d. Experiment with multi-level decision trees and error pre-pruning by changing the option `min_samples_leaf` from 0 to the size of the datasets (use some step). (The option `min_samples_leaf` determines the min number of training instances in the leaf nodes of the decision trees.) Estimate the accuracy rates of the resulting decision trees using the training set and hold-out validation. Plot the accuracy rates based on the training

[1] Hold-out validation is explained in Appendix A.

set and hold-out validation for `min_samples_leaf` from 1 to the size of the datasets with step of 5. Identify the regions of underfitting, optimality, and overfitting. Explain how you have identified these regions.

Report: Prepare a pdf file of the Jupiter notebook with your code and answers in markdown fields for points c and d from Section 4.

Appendix A: Commands in `Python`, `pandas`, and `sklearn`

1. **File reading**

   To read a `csv` file using `pandas` you need first to import `pandas`:

   ```
   import pandas as pd
   ```

   and then to read the file of the data set into `pandas` frame with name `data`, e.g.,

   ```
   data = pd.read_csv('diabetes.csv')
   ```

2. **Preparing the `data` frame for training classification models**

   To prepare the `data` frame for training classification models we need clearly to specify the input matrix `X` in `data` (i.e. the sub-matrix given by input attributes) and the output column (vector) `Y` in `data` (given by the output attribute) [2]. In both data sets the output attribute has a name `class` and one possible sequence of operations that can be used is:

   ```
   Y = data['class']
   X = data.drop(['class'],axis=1)
   ```

3. **Setting and training decion-tree classifiers**

   To set a decision tree classifier you need first to import `tree` module in `sklearn`:

   ```
   from sklearn import tree
   ```

   and then to set the decision tree classifier `clf` according to the specification proposed in the lab description:

   ```
   clf = tree.DecisionTreeClassifier(criterion = 'entropy',
         min_samples_leaf = n)
   ```

   where `n` can be any value from 1 to the size of the training set.

   To train the classifier `clf` from training data `X` and `Y` you need to execute command:

   ```
   clf = clf.fit(X, Y)
   ```

[2] A *k*th row in `X` corresponds to the input values for the *k*th instance and a *k*th element in `Y` corresponds to the output value for the *k*th instance.

4. **Predicting with decision-tree classifiers**

   Assume that we have test data with input matrix `Xt` and output vector `Yt`. To compute the vector Yp of predictions for `Xt` provided by the decision-tre classifier `clf` we execute command:

   ```
   Yp = clf.predict(Xt)
   ```

   To compute accuracy of predictions we need first to import function `accuracy_score` from module `sklearn.metrics`:

   ```
   from sklearn.metrics import accuracy_score
   ```

   and then to compute the accuracy rate `acc`:

   ```
   acc = accuracy_score(Yt, Yp)
   ```

   We note that, if `Xt` is `X` and `Yt` is `Y`, the estimate rate `acc` is the training accuracy rate.

5. **Preparing data for hold-out validation of decision trees**

   In the lab description we are asked to perform hold-out validation of decision trees. This means that the original data set is first split randomly into training data set and test data set. Then a decision tree is trained on the training data and tested on the test data.

   In `sklearn` there exists a function `train_test_split` that allows us to get the training data set and test data set given by their input matrices and output vectors. For example, the command:

   ```
   X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
   test_size=0.66, random_state=10)
   ```

   will randomly select 66% of the rows of `X` and `Y` for training and will assign the selected rows to `X_train` and `Y_train`. The remaining 34 of the rows of `X` and `Y` will be assigned to `X_test` and `Y_test`.

   We note that option `random_state` is the random seed. So, if you need to repeat the hold-out validation please use each time another value for this option. Otherwise, each new run of the function `train_test_split` will generate the same splits.

   To use the function `train_test_split` we need import:

   ```
   from sklearn.model_selection import train_test_split
   ```

Appendix B: Data Sets

I. **Diabetes Data**

1. Title: Pima Indians Diabetes Database

2. Sources:
   (a) Original owners: National Institute of Diabetes and Digestive and
   Kidney Diseases
   (b) Donor of database: Vincent Sigillito (vgs@aplcen.apl.jhu.edu)
   Research Center, RMI Group Leader
   Applied Physics Laboratory
   The Johns Hopkins University
   Johns Hopkins Road
   Laurel, MD 20707
   (301) 953-6231

3. Relevant Information:
   Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. ADAP is an adaptive learning routine that generates and executes digital analogs of perceptron-like devices. It is a unique algorithm; see the paper for details.

4. Number of Instances: 768

5. Number of Attributes: 8 plus class

6. For Each Attribute: (all numeric-valued)
   1. Number of times pregnant
   2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
   3. Diastolic blood pressure (mm Hg)
   4. Triceps skin fold thickness (mm)
   5. 2-Hour serum insulin (mu U/ml)
   6. Body mass index (weight in kg/(height in m)^2)
   7. Diabetes pedigree function
   8. Age (years)
   9. Class variable (0 or 1)

7. Missing Attribute Values: None

8. Class Distribution: (class value 1 is interpreted as "tested positive for diabetes")

Class Value	Number of instances
0	500
1	268

9. Brief statistical analysis:

Attribute number:	Mean:	Standard Deviation:
1.	3.8	3.4
2.	120.9	32.0
3.	69.1	19.4
4.	20.5	16.0
5.	79.8	115.2
6.	32.0	7.9
7.	0.5	0.3
8.	33.2	11.8

Relabeled values in attribute 'class'

From: 0	To: tested_negative
From: 1	To: tested_positive

2. Glass Data

1. Title: Glass Identification Database

2. Sources:
   (a) Creator: B. German
       -- Central Research Establishment
          Home Office Forensic Science Service
          Aldermaston, Reading, Berkshire RG7 4PN
   (b) Donor: Vina Spiehler, Ph.D., DABFT
          Diagnostic Products Corporation
          (213) 776-0180 (ext 3014)

3. Relevant Information:n
   Vina conducted a comparison test of her rule-based system, BEAGLE, the nearest-neighbor algorithm, and discriminant analysis. BEAGLE is a product available through VRS Consulting, Inc.; 4676 Admiralty Way, Suite 206; Marina Del Ray, CA 90292 (213) 827-7890 and FAX: -3189. In determining whether the glass was a type of "float" glass or not, the following results were obtained (# incorrect answers):

Type of Sample		Beagle	NN	DA
Windows that were float processed	(87)	10	12	21
Windows that were not:	(76)	19	16	22

   The study of classification of types of glass was motivated by criminological investigation. At the scene of the crime, the glass left can be used as evidence...if it is correctly identified!

4. Number of Instances: 214

5. Number of Attributes: 10 (including an Id#) plus the class attribute
   -- all attributes are continuously valued

6. Attribute Information:
   1. RI: refractive index
   2. Na: Sodium (unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
   3. Mg: Magnesium
   4. Al: Aluminum
   5. Si: Silicon
   6. K: Potassium
   7. Ca: Calcium
   8. Ba: Barium
   9. Fe: Iron
   10. class: type of glass
      -- 1 building_windows_float_processed
      -- 2 building_windows_non_float_processed
      -- 3 vehicle_windows_float_processed
      -- 4 vehicle_windows_non_float_processed (none in this database)
      -- 5 containers
      -- 6 tableware
      -- 7 headlamps

7. Missing Attribute Values: None

Summary Statistics:

Attribute:	Min	Max	Mean	SD	Correlation with class
2. RI:	1.5112	1.5339	1.5184	0.0030	-0.1642
3. Na:	10.73	17.38	13.4079	0.8166	0.5030
4. Mg:	0	4.49	2.6845	1.4424	-0.7447
5. Al:	0.29	3.5	1.4449	0.4993	0.5988
6. Si:	69.81	75.41	72.6509	0.7745	0.1515
7. K:	0	6.21	0.4971	0.6522	-0.0100
8. Ca:	5.43	16.19	8.9570	1.4232	0.0007
9. Ba:	0	3.15	0.1750	0.4972	0.5751
10. Fe:	0	0.51	0.0570	0.0974	-0.1879

8. Class Distribution: (out of 214 total instances)
    -- 163 Window glass (building windows and vehicle windows)
       -- 87 float processed
          -- 70 building windows
          -- 17 vehicle windows
       -- 76 non-float processed
          -- 76 building windows
          -- 0 vehicle windows
    -- 51 Non-window glass
       -- 13 containers
       -- 9 tableware
       -- 29 headlamps

```
Relabeled values in attribute class
  From: '1'                    To: 'build wind float'
  From: '2'                    To: 'build wind non-float'
  From: '3'                    To: 'vehic wind float'
  From: '4'                    To: 'vehic wind non-float'
  From: '5'                    To: containers
  From: '6'                    To: tableware
  From: '7'                    To: headlamps
```