Applied Machine Learning - EDAN95
Assignment 2

Machine Learning Data in Weka
Clustering, Classifying, Decision Tree

Hicham Mohamad
hsmo@kth.se

november 18, 2018

1 Introduction

The goal in first part of this lab assignment is to:

- get acquainted with **WEKA**, Waikato Environment for Knowledge Analysis;
- learn how to load data sets into the WEKA tool;
- experiment with some clustering algorithms;
- experiment with some classification algorithms.

In addition, in the second part of this lab session we will

- explore some programming concepts in Python, SciKitLearn, and partially Numpy to implement a **decision tree** classifier and compare it with the provided one in SciKitLearn,
- get acquainted with the simplified version of the MNIST dataset provided in SciKitLearn, and
- explore the effect of (some) data preprocessing on the learning process.

1.1 Data set

For the experiments you may use the classical Iris data set (three iris flower species characterized by four measurements, 50 individuals of each species, together 150 data points). There exist several versions of the data set available, locate at least two in different formats (arff is WEKA's native one, find also something else like e.g. CSV). Force WEKA to load them. In the meantime you should get acquainted with WEKA's help system and documentation.

If you have time left, you may also perform similar experiments to the ones described below with another data set of your choice. It should have more than two classes and at least hundred data points.

1.2 Clustering

Attempt to cluster the data using k-means. How well do the clusters correspond to the actual classes? Can you improve your result? How? Can you find another clustering algorithm in the WEKA library and use it instead? Is the result better? Visualize and show your result to the TA.

1.3 Classifying

1. Split your data file in two parts, training (80%) and test (20%). Apply k-NN in order to classify the iris flowers from the test set using the rest as training data. How should the distance function look like? Can you change it? How does your classifier perform? Perform several 80/20 splits. Does the result change? How well can you classify the flowers in the best case? Show the results to the TA.

2. Try now to classify iris flowers using a decision tree. You have to create a decision tree for the data set, splitting the data appropriately into training and test sets. How does your classifier perform? Can you improve it by changing some of the parameters? How or why not? Show your results to the TA.

1.4 Decision Tree and Tools

A classic algorithm for training / constructing **decision trees** is **ID3**. However, in SciKitLearn, only one decision tree **classifier** is provided, which is based on another algorithm, **CART**.

ID3 has the advantage of being able to handle **multivalued attributes** in the decisions, i.e., it is possible to split the tree into arbitrarily many **subtrees** in any node, not only two. While it is of course still possible to use a **binary tree** structure also to represent rather complex value landscapes by making the decision as "one against all", the tree would obviously have to grow deeper, as the remaining decisions on the same attribute might still have to be made.

To compare the built-in tree with an ID3-implementation, you will implement your own version of an **ID3-decision tree** classifier. For this, feel free to consult other implementations, one is for example available here: `https://github.com/tofti/python-id3-trees`, but you should refer to your source when presenting a solution as your own when it contains code provided by others.

A zip-file with some **code skeletons**, useful snippets and a visualised tree for comparison is provided HERE: `DT_LabHandout.zip`. OBS, there was a mismatch between the `toyTree.pdf` and the dataset behind it, the code should now be updated to again contain the third attribute value for color ('b', for 'blue', which does, intentionally, not show in any of the samples, to test for this situation).

OBS: The **GraphViz module is not installed** on the computers in E:Alfa and E:Beta yet (2019-11-12). The respective administrator is informed. For visualisation, you should rely on simple text-printouts, e.g., by printing "nodestring" directly, instead of sending it to the dot-data in "`add_node_to_graph`". The handout has been updated accordingly. For tips regarding the visualisation of the built-in classifier, see below the ID3-algorithm.

2 How to Talk About Data in Weka

Machine learning algorithms are primarily designed to work with arrays of numbers.

This is called tabular or structured data because it is how data looks in a *spreadsheet*, comprised of rows and columns.

Weka has a specific computer science centric vocabulary when describing data:

- **Instance:** A row of data is called an instance, as in an instance or observation from the problem domain.
- **Attribute:** A column of data is called a feature or attribute, as in feature of the observation.

Each attribute can have a different type, for example:

- Real for numeric values like 1.2.
- Integer for numeric values without a fractional part like 5.
- Nominal for categorical data like "dog" and "cat".
- String for lists of words, like this sentence.

On classification problems, the output variable must be nominal. For regression problems, the output variable must be real.

2.1 Data in Weka

Weka prefers to load data in the ARFF format.

ARFF is an acronym that stands for `Attribute-Relation File Format`. It is an extension of the CSV file format where a header is used that provides metadata about the data types in the columns.

For example, the first few lines of the classic iris flowers dataset in CSV format looks as follows:

Listing 1: iris flowers dataset in CSV format

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
```

The same file in ARFF format looks as follows:

Listing 2: the same file in ARFF format

```
@RELATION iris

@ATTRIBUTE sepallength REAL
@ATTRIBUTE sepalwidth REAL
@ATTRIBUTE petallength REAL
@ATTRIBUTE petalwidth REAL
@ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}

@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
```

You can see that directives start with the at symbol (@) and that there is one for the name of the dataset (e.g. @RELATION iris), there is a directive to define the name and datatype of each attribute (e.g. @ATTRIBUTE sepallength REAL) and there is a directive to indicate the start of the raw data (e.g. @DATA).

Lines in an ARFF file that start with a percentage symbol (%) indicate a comment.

Values in the raw data section that have a question mark symbol (?) indicate an unknown or missing value. The format supports numeric and categorical values as in the iris example above, but also supports dates and string values.

Depending on your installation of Weka, you may or may not have some default datasets in your Weka installation directory under the data/ subdirectory. These default datasets distributed with Weka are in the ARFF format and have the .arff file extension.

2.2 Load CSV Files in the ARFF-Viewer

Your data is not likely to be in ARFF format.

In fact, it is much more likely to be in `Comma Separated Value` (CSV) format. This is a simple format where data is laid out in a table of rows and columns and a comma is used to separate the values on a row. Quotes may also be used to surround values, especially if the data contains strings of text with spaces.

The CSV format is easily exported from Microsoft Excel, so once you can get your data into Excel, you can easily convert it to CSV format.

Weka provides a handy tool to load CSV files and save them in ARFF. You only need to do this once with your dataset.

Using the steps below you can convert your dataset from CSV format to ARFF format and use it with the Weka workbench. If you do not have a CSV file handy, you can use the iris flowers dataset. Download the file from the UCI Machine Learning repository and save it to your current working directory as iris.csv.

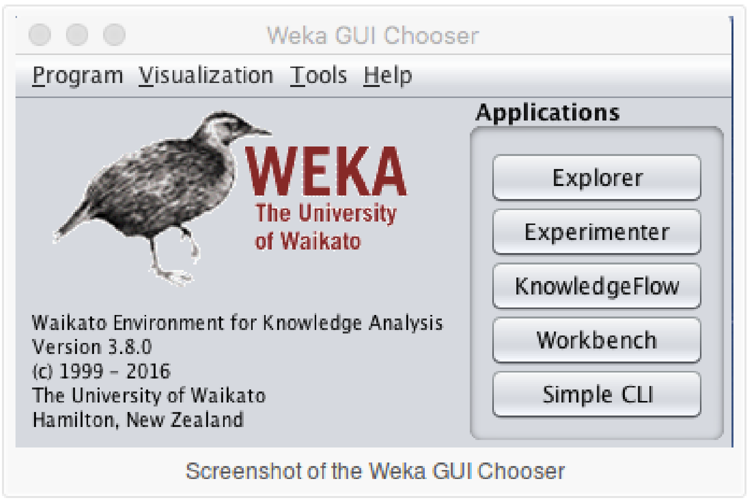
Weka GUI Chooser
Program Visualization Tools Help
Applications
WEKA
The University of Waikato
Explorer
Experimenter
KnowledgeFlow
Workbench
Simple CLI
Waikato Environment for Knowledge Analysis
Version 3.8.0
(c) 1999 - 2016
The University of Waikato
Hamilton, New Zealand
Screenshot of the Weka GUI Chooser

Figure 1: Start Weka chooser.

1. Start the Weka chooser.
2. Open the ARFF-Viewer by clicking "Tools" in the menu and select "ArffViewer".
3. You will be presented with an empty ARFF-Viewer window.
4. Open your CSV file in the ARFF-Viewer by clicking the "File" menu and select "Open". Navigate to your current working directory. Change the "Files of Type:" filter to "CSV data files (*.csv)". Select your file and click the "Open" button.

You can now load your saved .arff file directly into Weka.

Note, the ARFF-Viewer provides options for modifying your dataset before saving. For example you can change values, change the name of attributes and change their data types.

It is highly recommended that you specify the names of each attribute as this will help with analysis of your data later. Also, make sure that the data types of each attribute are correct.

2.3 Load CSV Files in the Weka Explorer

You can also load your CSV files directly in the Weka Explorer interface.

This is handy if you are in a hurry and want to quickly test out an idea.

This section shows you how you can load your CSV file in the Weka Explorer interface. You can use the iris dataset again, to practice if you do not have a CSV dataset to load.

1. Start the Weka GUI Chooser.
2. Launch the Weka Explorer by clicking the "Explorer" button.
3. Click the "Open file. . . " button.
4. Navigate to your current working directory. Change the "Files of Type" to "CSV data files (*.csv)". Select your file and click the "Open" button.

   You can work with the data directly. You can also save your dataset in ARFF format by clicking he "Save" button and typing a filename.

3 Part 2 - Instructions

To pass the assignment, you need to show an implementation that runs, produces sensible results and that you can explain individually when asked to do so. Also, you should have answers to the questions posed in the different sub tasks.

Your task consists of the following two blocks:

3.1 Use and experiment with the built in SciKitLearn DecisionTreeClassifier (based on CART)

1. Use the code skeleton / snippets provided or the notebook used in the tutorial `IntroMaterial.zip` from the first course week to load the digits dataset from the datasets provided in SciKitLearn. Inspect the data. What is in there?

2. Split your data set into 70% training data (features and labels), and 30% test data.

3. Set up a DecisionTreeClassifier as it comes in SciKitLearn. Use it with default parameters to train a decision tree classifier for the digits dataset based on the training data. Follow the tutorial (or the respective documentation) and produce a plot of the tree with graphviz if you have it available (if not, see tips below the ID3 algorithm). What can you learn from this about how the used algorithm handles the data?

4. Test the classifier with the remaining test data and analyse it using the metrics packages of SciKitLearn (classification report, confusion matrix). What do you see?

5. Change the parameters of the classifier, e.g., the minimum number of samples in a leaf / for a split, to see how the tree and the results are affected.

3.2 Implement your own decision tree classifier based on the ID3 algorithm and compare the results.

1. Make a decision regarding the data structure that your tree should be able to handle. In the code handout (see above), you will find the tree assumed to be implemented with nodes that are dictionaries.

2. Inspect other parts of the code provided. You will find one example for how it is easily possible to construct the visualisation data (dot-data) for the graphviz-visualisation in parallel to the actual decision tree. Whenever a node is added to the tree, it can also immediately be added to the graph. Feel free to use this for your own implementation.

3. Simply running main in the handout will produce a tree with one node, visualised in testTree.pdf. Make sure that this works, i.e., that you have all the necessary libraries installed.

4. The code handout contains a mere skeleton for the ID3 classifier. Implement what is needed to actually construct a decision tree classifier. Implement the ID3 algorithm, e.g., according to what is provided in the lecture or on this page below. Use information gain as criterion for the best split attribute.

5. Test your classifier with the toy example provided in the ToyData class given in the skeleton. In main you can also see how to make use of the dot-data to produce a visualisation with graphviz. The tree rendered in the file given to the respective method should look like the one given in toyTree.pdf.

6. When you are sure that everything works properly, run the ID3-training for the digits training data you used in part 1. Do not constrain the training, i.e., run with default parameters. What do you see in the plot? Analyse the result (produce a confusion matrix and a classification report) and compare with the result from part 1 (when running with default parameters).

7. One striking difference should be in the ratio of breadth and depth of the two trees. Why is that the case? Modify your data set to contain only three values for the attributes (instead of potentially 16), e.g., 'dark', 'grey', and 'light', with for example 'dark' representing pixel values < 5.0, and 'light' those > 10.0. Train and test the classifier again. Do your results improve? Can you match the SKLearn implementation's accuracy? If not, why do you think this is the case?

8. (Bonus: If interested, explore the effects of different parameters regulating the depth of the tree, the maximum number of samples per leaf or required for a split, initially on the SKLearn version, but of course you can also implement them for your own classifier.)

3.3 The ID3 algorithm in pseudocode

ID3 builds a tree structure recursively. If you are unfamiliar with trees or recursion, it is strongly recommended to consult material from EDAA01 (Programming techniques, advanced course).

```
ID3 (Samples, Target_Attribute, Attributes)
  Create a (root) node Root for the tree

  If all samples belong to one class <class_name>
      Return the single-node tree Root, with label = <class_name>.

  If Attributes is empty, then
      Return the single node tree Root, with label = most common class
          value in Samples.
  else
      Begin
          Let A be the attribute a in Attributes that generates the maximum
              information gain
                when the tree is split based on a.

          Set A as the target_attribute of Root

          For each possible value, v, of A, add a new tree branch below
              Root,
               corresponding to the test A == v, i.e.,
              Let Samples(v) be the subset of samples that have the value v
                   for A.
              If Samples(v) is empty, then
                  Below this new branch add a leaf node with label
                        = most common class value in Samples.
              else
                  Below this new branch add the subtree ID3 (Samples(v), A,
                       Attributes/{A})
        End
  Return Root
```

3.4 Predicting with a Decision Tree

The prediction (finding the class for an example x) with a decision tree boils then obviously down to a tree search, which follows the branch of the tree that represents the combinations of attribute values given in x, until a leaf is reached. The predicted class for x is then the class that the leaf is labeled with. This is, again, easiest implemented recursively:

```
predict_rek( node, x)
    if node is leaf
        return the class label of node
    else
        find the child c among the children of node
             representing the value that x has for
             the split_attribute of node
        return predict_rek( c, x)
```

3.5 Plotting without GraphViz

You can use matplotlib, which should be installed. Using it without any parameters (and with classifier being your DecisionTreeClassifier)

```
import matplotlib.pyplot as plt
from sklearn import tree, metrics, datasets
from sklearn.tree import DecisionTreeClassifier

plt.figure()
```

```
tree.plot_tree(classifier)
plt.show()
```

will produce the full tree (good to get some idea how it looks overall), while using it with parameters

```
plt.figure(figsize=(25,5))
tree.plot_tree(classifier, max_depth=2)
plt.show()
```

will give you some understanding on how the decisions are formed.

4 Appendix

References

[1] VanderPlas, A Whirlwind Tour of Python. O'Reilly 2016. Online reading: https://jakevdp.github.io/WhirlwindTourOfPython/

[2] Richard Johansson, "Scientific Computing with Python". https://github.com/jrjohansson/scientific-python-lectures

[3] Toft, Python ID3 Trees, github: https://github.com/tofti/python-id3-trees

[4] Geron, Jupyter notebook on linear algebra from Machine Learning and Deep Learning in Python using Scikit-Learn, Keras and TensorFlow: https://github.com/ageron/handson-ml2

[5] Kevin P. Murphy. *Macine Learning: A Probabilistic Perspective*, The MIT Press, Cambridge, Massachusetts.

[6] Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning.* MIT Press, 2016, ISBN: 9780262035613.