Machine Learning with Weka

PU Machine learning mit Java, Weka und UIMA

Nils Reiter,
nils.reiter@uni-koeln.de

November 24, 2020
Winter term 2020/21

Section 1

Exercise 03

Exercise 3

- Data source: ‘Hackatorial’

Exercise 3

- Data source: 'Hackatorial'

Workflow

- You need lists of
  - the filter steps you selected
  - the attributes you removed
  - the classifier (settings) that work best
- Update your repository
- Open the file `test.arff` using the Explorer
  - Apply apply all filter steps
  - Remove the attributes you selected
  - Save the file `test.arff` under a new name
- Load `data.arff` and apply preprocessing steps (again)
- Switch to the classify tab and load the modified test file
  - Select the classifier `weka.classifiers.misc.InputMappedClassifier`
    click in its option window and select the classifier you want to use, with its properties
- Run the classification – copy/paste the mean f-score results in the discord chat

Section 2

Weka as Java API

Introduction

- Weka can be used as a Java API
- Not a lot of documentation
- Javadoc: `https://javadoc.io/doc/nz.ac.waikato.cms.weka/weka-stable/latest/index.html`

Introduction

- Weka can be used as a Java API
- Not a lot of documentation
- Javadoc: https://javadoc.io/doc/nz.ac.waikato.cms.weka/weka-stable/latest/index.html

Listing 2: Maven Dependency for Weka

```
<dependency>
  <groupId>nz.ac.waikato.cms.weka</groupId>
  <artifactId>weka-stable</artifactId>
  <version>3.8.4</version>
</dependency>
```

Beware: Coding Machine Learning

- Using machine learning in code relies on libraries
- Libraries (this week: Weka) do all the (probabilistic|numerical|statistical) 'heavy lifting'

Beware: Coding Machine Learning

- Using machine learning in code relies on libraries
- Libraries (this week: Weka) do all the (probabilistic|numerical|statistical) 'heavy lifting'
- What remains?
  - Input
    - Derive suitable input representation
    - Prepare data to be in the right format
  - Output
    - Do something with the result or the model

Beware: Coding Machine Learning

- Using machine learning in code relies on libraries
- Libraries (this week: Weka) do all the (probabilistic|numerical|statistical) 'heavy lifting'
- What remains?
  - Input
    - Derive suitable input representation
    - Prepare data to be in the right format
  - Output
    - Do something with the result or the model

$\Rightarrow$ Code distribution

- Most of the code will not be about machine learning
- But about input/output handling
- This also consumes most of the development time

API Overview

Packages

- `weka.core`: Data IO, some preprocessing, conversion
- `weka.gui`: Everything related to the GUI
- `weka.filters`: Filters. Further subdivision into `feature` vs. `instance` and `super-` vs. `unsupervised`
- `weka.classifiers`: Classifiers. Further subdivision into types
- `weka.clusterers`: Clustering methods. Not so many.

Most Important Classes

- `weka.core.Instances`: Container class for a data set, supports weighting and ordering
- `weka.core.Instance`: Interface for a single instance. Classes are: `DenseInstance`, `SparseInstance`, `BinarySparseInstance`
- `weka.core.Attribute`: Represents a feature (not a feature value!)
  - Attributes have a name and a data type (numeric, nominal, string, date)
- `weka.classifiers.Classifier`: Interface that all classifiers implement
- `weka.filters.Filter`: Interface that all filters implement

Instance Representation

- Internally, all attribute values are represented as `double` values
- Mapping nominal/string values is done by the class
- Methods to set attribute values come in two variants:

  `void setValue(Attribute att, String value)` and `void setValue(Attribute att, double value)`

Instance Representation

- Internally, all attribute values are represented as `double` values
- Mapping nominal/string values is done by the class
- Methods to set attribute values come in two variants:

`void setValue(Attribute att, String value)` and `void setValue(Attribute att, double value)`

DenseInstance vs. SparseInstance

- `DenseInstance`: The naive implementation, representing each feature value as a double
- `SparseInstance`: Stores only non-zero values
  - Nominal values: First value not stored (because it has index zero)
- `SparseInstance` useful for large data sets with many default values

`https://javadoc.io/static/nz.ac.waikato.cms.weka/weka-stable/3.8.4/weka/core/Instances.html`

weka.classifiers.Classifier

Methods

- Train a classifier

```
public void buildClassifier(Instances data) throws Exception
```

- Classify a single instance

```
public double classifyInstance(Instance instance) throws Exception
```

  - return value represented as double!

- Classify a single instance

```
public double[] distributionForInstance(Instance instance) throws Exception
```

  - May return a distribution of class probabilities

- Meta data about the classifier

```
public Capabilities getCapabilities()
```

https://javadoc.io/static/nz.ac.waikato.cms.weka/weka-stable/3.8.4/weka/classifiers/Classifier.html

weka.filters.Filter

Methods

- Filter a single instance

```
public boolean input(Instance instance) throws Exception
```

- Retrieve output

```
public Instance output()
```

- Filter an entire data set

```
public static Instances useFilter(Instances data, Filter filter) throws Exception
```

- ...

https://javadoc.io/static/nz.ac.waikato.cms.weka/weka-stable/3.8.4/weka/filters/Filter.html

```java
class Main {
  public static void main(String[] args) throws IOException, Exception {

    // load data
    ArffLoader loader = new ArffLoader();
    loader.setFile(new File("data.arff"));
    Instances instances = loader.getDataSet();
    instances.setClassIndex(instances.numAttributes() - 1);

    // train classifier
    OneR classifier = new OneR();
    classifier.buildClassifier(instances);

    // serialize model
    weka.core.SerializationHelper.write(
      instances.relationName() + ".OneR"), classifier);
  }
}
```

Exercise 4

`https://github.com/idh-cologne-machine-learning-mit-java/exercise-04`