M2 IMIS/MIAGE
Université d'Orléans

TP *Fouille de données et textes*
Document filtering

---

1 Introduction

Aims:

- To reinforce some of the concepts of *fouille de données et textes* from the lectures, mainly the concepts of *fouille de textes* (text mining).
- To gain familiarity with the WEKA software package.
- To produce some assessable work for this subject.

Procedure:

WEKA (Waikato Environment for Knowledge Analysis)

https://www.cs.waikato.ac.nz/ml/weka/

is a freely available machine learning software package that is very well-known and widely used. The software is written in Java and full source code and code documentation is provided. There is also an accompanying book.

WEKA contains implementations of many machine learning algorithms. It also has many other features such as the ability to run large batches of experiments, data visualization and pre-processing.

1.1 WEKA: brief description

WEKA can be launched either from the main Program Menu or by double-clicking on any ARFF file. However, since the data sets that we will work are large, in order to avoid problems with memory on the Java heap, run the following command line to start WEKA:

```
java -Xmx4G -jar weka.jar
```

This increases the Java heap to 1024m. When WEKA starts, a GUI with different working modes is shown: `Explorer`, `Experimenter`, `Knowledge-Flow`, and `SimpleCLI`. In our exercises, we will use the `Explorer`. This mode is used to explore data interactively.

The WEKA Explorer

At the very top of the window, just below the title bar, is a row of tabs (panels/*onglets*). When the `Explorer` is first started only the first tab is active; the others are grayed out. This is because it is needed to open (and potentially pre-process) a data set before starting to explore the data. Some examples of the tabs are the following:

- `Preprocess`. Choose and modify the data being used.
- `Classify`. Train and test learning schemes that classify or perform regression.
- `Cluster`. Learn clusters for the data.
- `Associate.` Learn association rules for the data.

- *Select attributes*. Select the most relevant attributes in the data.
- *Visualize*. View an interactive 2D plot of the data.

A short tutorial on the *Explorer* can be found here (https://www.cs.waikato.ac.nz/~ml/weka/gui_explorer.html).

Data format

Data sets for WEKA should be formatted according to the ARFF (Attribute Relationship File Format) format. However, there are several converters included in WEKA that can convert other file formats to ARFF (e.g., CSV files). According to the ARFF format, a data set has to start with a declaration of its name `@relation` name, followed by a list of all the attributes in the data set (including the class attribute). These declarations have the form:

`@attribute` *attribute name specification* (`numeric`, `nominal`, `string`, ...)

After the attribute declarations, the actual data is introduced by a `@data` tag, which is followed by a list of all the instances. The instances are listed in comma-separated format, with a question mark representing a missing value.

To represent text information in WEKA we can use the `string` attribute type. A `string` attribute can have in principle infinite number of values and therefore it cannot be handled by any classifier in WEKA. That is why we have to convert `string` values into a set of attributes that represent the frequency of each word/term in the strings (documents). This type of string conversion can be realized by the `StringToWordVector` filter in WEKA. The figure below provides an example of string conversion implemented by this filter.

```
@relation text

@attribute docs string
@attribute class {animals, no_animals}

@data
'cat mouse lion', animals
'tree flower', no_animals

@relation text-converted

@attribute class {animals, no_animals}
@attribute cat numeric
@attribute mouse numeric
@attribute lion numeric
@attribute tree numeric
@attribute flower numeric

@data
animals,1,1,1,0,0
no_animals,0,0,0,1,1
```

2 Data and task

We will work with the problem of text classification, that is, the task of assigning predefined categories to free-text documents. The learning task can be defined as follows: *given training documents together with their category label (class), return a hypothesis function (a classifier) that can predict the label of a document*. In this context, the instance space includes the set of all available documents. The feature space is the set of descriptors specified by the learner's background knowledge.

For example, spam filters are used to detect unsolicited emails and prevent these messages from reaching a user's mailbox. The instance space is the set of all messages, and the concept is defined as the set of spam messages. As for the feature space, text documents are usually represented as a bag-of-words (*sac de mots*) --- https://en.wikipedia.org/wiki/Bag-of-words_model. The basic idea is to represent a document in a word-based feature space, i.e., a high-dimensional vector where each dimension corresponds to a word.

More specifically, in the TPs, we will use one data set. It contains a collection of newspaper documents on various topics. The file has been processed into ARFF format, so it is made up of a number of instances – each instance consists of a `string` attribute (the document itself) and a binary class attribute indicating whether the document is related to the title of the data set, or not. More specifically, the data set contains documents classified either as "Corn" (Corn news) or "Non-Corn" (not mainly about corn)

- `ReutersCorn-train.arff`. This data set contains 1554 documents, 45 of which are about corn.

The goal is to verify if different learning algorithms implemented in WEKA can correctly classify documents based solely on the words in the document. In order to do so, as previously mentioned, we will use a word-based feature space. For example, by using a binary attribute indicating whether the word is found in a document or not. We can also use TDF/IDF document representation.

In terms of performance of the classifiers generated, besides the global error rate (`Incorrectly Classified Instances`), will be recording and comparing the `Precision` and `Recall` (https://en.wikipedia.org/wiki/Precision_and_recall).

The classifiers we will use in the TP are:

- Naive Bayes – NB (`weka.classifiers.bayes.NaiveBayes`)
- Naive Bayes Multinomial – NBM (`weka.classifier.bayes.NaiveBayesMultinomial`).
- Support Vector Machine Classifier – SVM (`weka.classifiers.functions.SMO`).
- k-NN (`weka.classifiers.lazy.IBk`).
- Decision trees – DT (weka.classifiers.trees.J48)

All of the learning algorithms will be used with their default parameters. In order to work with feature selection, we will employ the `Attribute.Selected.Classifier` with `InfoGainAttributeEval` as the evaluator and `Ranker` as the search method. Finally, all the experiments will be developed using 10-fold cross-validation.

3 Classifying/filtering documents using binary representation for the attributes

What we are doing in this TP is trying to find a model that can classify documents into *Corn* (Corn news) and *Non-Corn* (not mainly about corn) with the best possible trade-off *precision/recall*. This can also be seen as the *problem of filtering a set of documents that are relevant for the user* from a much larger set of mixed documents. Here, for example, the class of interest is that of *Corn*.

Task 01: Study the data file `ReutersCorn-train.arff` (open it in a text editor). As you can see the file is in WEKA ARFF format which is simple: a header section which includes description of attributes and the class label. So, for our case, we have one attribute which is *Text* of type `string` and *class-att* of type nominal: `0` for *Non-Corn* and `1` for *Corn*. From the point of view of text mining, there are many issues in the text file, such as UPERCASE/lowercase letters and special characters. We will be fixing all these issues in the next stage.

Task 02: Load the data file `ReutersCorn-train.arff`. Is this date set balanced? That is, is the number of instances in each class is the same?

3.1 Data Preparation

For the classification task to be done, a preliminary phase of text preprocessing and feature extraction is essential. We want to transform each text/document in a vector form (bag-of-words), in which each document is represented by the presence of some “important” terms/words. These terms are the ones contained in the collection vocabulary. To build the vocabulary, various operations are typically performed (many of which are language-specific):

- **Cleaning and Text Tokenization:** in this phase, each document is analyzed with the purpose of extracting the terms/words. Separator characters must be defined, along with a tokenization strategy for particular cases such as accented words, hyphenated words, acronyms, etc.
- **Stop-words removal:** a very common technique is the elimination of frequent usage words: conjunctions (e.g., or, and, nor, but, yet), prepositions (e.g., about, across, among, before, above), base verbs, etc. This kind of terms/words should be filtered as they have a poor discrimination power, making them useless for the text classification.
- **Lemmatization and stemming:** the lemmatization of a word is the process of determining its lemma. The lemma can be thought of as the “common root” of various related inflectional forms. For instance, the words *walk*, *walking* and *walked* all derive from the lemma *walk*. A simple technique for approximated lemmatization is the stemming. Stemming algorithms work by removing the suffix of the word, according to some grammatical rules.
- **Term selection/feature extraction**: the term set resulting from the previous phases has still to be filtered, since we need to remove the terms that have poor prediction ability (w.r.t the document class) or are strongly correlated to other terms. This term selection task also leads to a simpler and more efficient classification.

Cleaning and Tokenization

Both cleaning and tokenization can be done using the `StringToWordVector` filter in WEKA. The default text retrieval model used by the `StringToWordVector` filter is binary: each document is represented with an *n*-dimensional binary vector (bag-of-words), where *n* is the size of the vocabulary, and each value represents the presence (i.e., `1`) or the absence (i.e., `0`) of a vocabulary term/word in the document.

1. Click on `Choose` button below `Filter`
2. Choose `unsupervised->attribute->StringToWordVector`
3. Click on `StringToWordVector` to open the options
4. Change the following parameters:
   4.1. `doNotOperateOnPerClassBasis` = True (If this is set, the maximum number of words and the minimum term frequency is based on the documents in all the classes.)
   4.2. `lowerCaseTokens` = True (If set to True, then all the word tokens are converted to lower case before being added to the dictionary.)
   4.3. `stopwordsHandler->Choose->WordsFrom`, click on `WordsFromFile` and load the file with stop words (you can download it from CELENE).
   4.4. `tokenizer->Choose->WordTokenizer`, click on `WordTokenizer` and change the parameter/delimiters to:
   `` \r\n\t.,;:\"\'()?!-¿¡+*&#$%\\/=<>\[\]_`@0123456789 `` (you can copy the delimiters from CELENE. The first character is “space”). Split words by these characters.
   4.5. `Stemmer = NullStemmer` (No stemming algorithm will be used.)
   4.6. `wordsToKeep` = 2500 (The number of words, per class, to attempt to keep.)
   4.7. Click `OK`.
   4.8. Click `Apply` in the `Filter` panel --- far right.

Very important. After the StringToWordVector filter is applied, the class attribute for the data set becomes the first attribute in the list, instead of the last one (which is WEKA's default). Thus, before running any classifier, we must change this by following the steps below:

1. Click on Choose button below Filter
2. Choose unsupervised->attribute->Reorder
3. Click on Reorder to open the options
   3.1. Change attributeIndices to 2-last,1
   3.2. Click OK.
4. Click Apply in the Filter panel --- far right.

3.2 Classification

Task 04: Creating the classifiers **WITHOUT** feature selection. Go to the Classify panel (tab/*onglet*).

1. Classify the data using NB, NBM, SVM (SMO), k-NN (IBk) and DT (J48). Record the global error rate (Incorrectly Classified Instances), the Precision and Recall for the class *Corn*.
2. Compare the results of the task above. Which model was better at classifying the documents and why (include your knowledge of Precision/Recall rates in your answer)?

Task 05: Creating the classifiers **WITH** feature selection. The word/term set resulting from the previous task can still be filtered, since we need to remove the terms that have poor prediction ability (w.r.t the document class) or are strongly correlated to other terms. This term selection task can lead to a simpler and more efficient classification.

1. Go to the Classify panel (tab/*onglet*).
2. Click on Choose button below Classify.
3. Choose meta->AttributeSelectedClassifier
4. Click on AttributeSelectedClassifier to open the options
5. Change the following parameters:
   5.1. classifier (choose either Naïve Bayes, Naïve Bayes Multinomial, SVM or k-NN)
   5.2. evaluator->InfoGainAttributeEval
   5.3. search->Ranker
      5.3.1. In the Ranker settings, change numToSelect to 25.

The algorithm will rank all the attributes but will keep only the top 25, according to their information gain (https://en.wikipedia.org/wiki/Information_gain_ratio), and classify the data based on those attributes.

- Again, record the global error rate (Incorrectly Classified Instances), the Precision and Recall for the class Corn for each classifier generated (NB, NBM, SVM, k-NN and DT).

Task 06: Scroll up in the output window to view the Ranker results from **Task 05**. Browse the words (attributes) that have been kept. How do you think the evaluator performed? Are the words relevant to documents about corn? Inspect the words and write down any that you think are unimportant (or not specific to corn documents). Explain why you think they made it on the list.

Task 07: Compare the results, in terms of Incorrectly Classified Instances, Precision and Recall, from **Task 04** and **Task 05** and discuss your findings (did ranking the attributes give a better or worse result? Why?).

Task 08: Load **AGAIN** the data file `ReutersCorn-train.arff.` Apply the `StringToWordVector` filter, but this time using **stemming**.

1. Click on `Choose` button below `Filter`
2. Choose `unsupervised->attribute->StringToWordVector`
3. Click on `StringToWordVector` to open the options
4. Change the following parameters:
   4.1. `doNotOperateOnPerClassBasis` = True
   4.2. `lowerCaseTokens` = True
   4.3. `stopwordsHandler->Choose->WordsFrom.` Click on `WordsFromFile` and load the file with stop words.
   4.4. `tokenizer->Choose->WordTokenizer`, click on `WordTokenizer` and change the parameter/delimiters to:
   `\r\n\t.,;:\"\'()?!-¿¡+*&#$%\\/=<>\[\]_`@0123456789`
   4.5. `Stemmer = IteratedLovinsStemmer`
   4.6. `wordsToKeep` = 2500 (The number of words, per class, to attempt to keep.)
   4.7. Click `OK.`
   4.8. Click `Apply` in the `Filter` panel --- far right.
5. By using the `Reorder` filter, set the first attribute as the class.

- Check the list of attribute/words generated. Do you observe that some end of words has been "cut"?

Task 09: In the context of stemming, creating the classifiers **WITH** feature selection. To do so, use the same configuration as that in **Task 05**.

Task 10: Compare the results, in terms of `Incorrectly Classified Instances`, `Precision` and `Recall`, from **Task 09** and **Task 05** and discuss your findings (did the use of stemming give a better or worse result? Why?).

4 Classifying/filtering documents using the TF-IDF representation for the attributes

TF-IDF (https://en.wikipedia.org/wiki/Tf–idf), short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word/term is to a document in a collection or corpus. It is often used as a weighting factor in text mining. The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the collection that contain the word, which helps to adjust for the fact that some words appear more frequently in general. TF-IDF is one of the most popular term-weighting schemes.

4.1 Data Preparation

Task 11: Load **AGAIN** the data file `ReutersCorn-train.arff`.

Cleaning and Tokenization

1. Click on `Choose` button below `Filter`
2. Choose `unsupervised->attribute->StringToWordVector`
3. Click on `StringToWordVector` to open the options
4. Change the following parameters:
   4.1. **`IDFTransform` = True**
   4.2. **`TFTransform` = True**
   4.3. `doNotOperateOnPerClassBasis` = True
   4.4. `lowerCaseTokens` = True (If set to True, then all the word tokens are converted to lower case before being added to the dictionary.)
   4.5. **outputWordCounts = True**
   4.6. `stopwordsHandler->Choose->WordsFrom`, click on `WordsFromFile` and load the file with stop words.
   4.7. `tokenizer->Choose->WordTokenizer`, click on `WordTokenizer` and change the parameter/delimiters to:
   ```
   \r\n\t.,;:\"\'()?!-¿¡+*&#$%\\/=<>\[\]_`@0123456789
   ```
   4.8. `Stemmer = NullStemmer`
   4.9. `wordsToKeep` = 2500
5. Click `OK.`
6. Click `Apply` in the `Filter` panel --- far right.
7. By using the `Reorder` filter, set the first attribute as the class.

- Do you observe that, now, the values of the attributes are not constrained to 0 or 1?

4.2 Classification

Task 12: In the context of TF-IDF, creating the classifiers **WITH** feature selection. To do so, use the same configuration as that in **Task 05**.

Task 13: Compare the results, in terms of `Incorrectly Classified Instances`, `Precision` and `Recall`, from **Task 12** and **Task 05** and discuss your findings.

References

- Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*, Morgan Kaufmann, Fourth Edition, 2016.
- *Tutorial: Document Classification using WEKA* https://medium.com/@karim_ouda/tutorial-document-classification-using-weka-aa98d5edb6fa
- *A data mining experiment: movie reviews classification using WEKA* https://www.stefanoscerra.it/movie-reviews-classification-weka-data-mining/