# USING WEKA FRAMEWORK IN DOCUMENT CLASSIFICATION

*Radu G. Creţulescu[1], Daniel I. Morariu[2], Macarie Breazu[3]*

[123] *"Lucian Blaga" University of Sibiu, Engineering Faculty, Computer Science and Electrical and Electronics Engineering Department, Romania*

**Abstract**

Text document classification problem is a special case of a supervised data mining problem. In order to solve a text document classification problem some steps are required to fulfill. The common steps are: feature extraction, feature selection, classification, evaluation and visualization. The WEKA is a framework that helps us with all these steps. WEKA was initially developed as a library of java classes that help us to implement data mining applications. In the last years, in order to avoid java programming skills, the components from WEKA are also available into a visual form inside "WEKA Knowledge Flow Environment". We have studied and present in this paper some of the most important visual components that are available in the WEKA framework for the previously presented steps. These components are: "Arff Loader", "Attribute Selection", "Normalize", "Train Test Split Maker", a lot of classifier algorithms, "Performance Evaluator" and "Text Viewer". In order to prove the functionality of the visual framework in text document classification we have made and present some experiments. The most important advantage of the visual WEKA framework is the possibility to test different approaches without programming abilities.

*Keywords:* Document Classification, WEKA Framework, Visual Tools.

## 1. Introduction

Text document classification problem is a special case of a supervised data mining problem. In order to solve a text document classification problem some steps are required to fulfill. The common steps are: feature extraction, feature selection, classification, evaluation and visualization. The WEKA is a framework that helps us with all these steps. WEKA was initially developed as a library of java classes that help us to implement data mining applications. In the last years, in order to avoid java programming skills, the components from WEKA are also available into a visual form inside "WEKA Knowledge Flow Environment"[7].

## 2. WEKA framework

WEKA (Waikato Environment for Knowledge Analysis [7]) offers a framework that contains a collection of machine learning algorithms implemented for solving a lot of data mining problems. The algorithms are written in java and are available open source for integrate in your projects. But, for avoiding the programing part, WEKA offers also a framework that permits you to describe your data mining application as a flow of actions and to evaluated it, without the need to write code. If you want to use WEKA for analyze your specific data, you need to write some code in order to transform your dataset into a format that is accepted by WEKA.

WEKA offers four different options for realizing your data mining process. The WEKA ***Knowledge Explorer*** is an easy to use framework with a graphical user interface that offers all the facilities of WEKA package, grouped in some general steps as preprocessing, classification, clustering, attribute selection, etc. For an unexperienced user, that has only some knowledge in data mining, this framework helps you, because it forces you to consider all the steps that need to be used into a standard data mining process.

Another framework is Weka ***Experiment Environment*** that permits you to create, run and modify an experiment into a simple manner. The experiment can be described into a text file and tested into the WEKA framework. This is for a more experimented user.

WEKA ***KnowledgeFlow Environment*** permits you to describe your experiment as a flow of steps with some visual connections between them.

The last framework is the WEKA ***Workbench*** that contains a lot of state of the art data preprocessing and machine learning algorithms. In this framework the user can quickly try out existing machine learning methods on new datasets in a very flexible way.

In this paper, for feature selection, classification and evaluation steps, we have used the WEKA KnowledgeFlow Environment [1]. The project flowchart realized in WEKA is presented in Fig. 1.

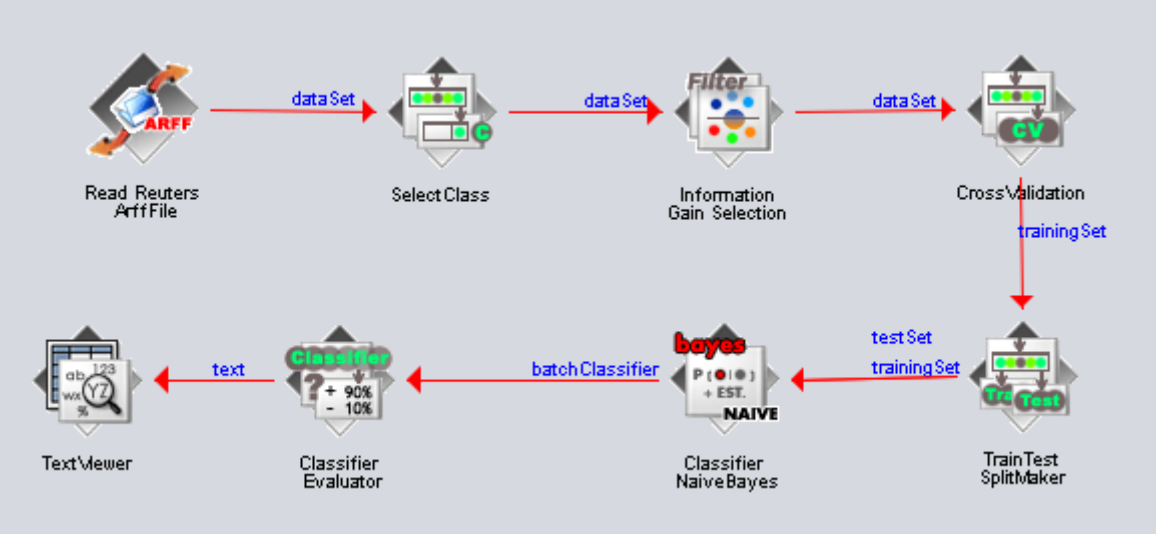


Figure 1 The WEKA project

In the following we present the components used in this project and a brief description of them as given by the WEKA framework help [8].

## 2.1 The ArffLoader Component

The first think that need to be done in each text mining project is to specify the dataset. For loading the dataset the WEKA framework has a lot of components as ArffLoader, DatabaseLoader, LibSVmLoader, XRFFLoader, etc. Those components can be found in the DataSource group and offer 12 different format input files. For our experiment we have chosen the arff format and we decided to use the Reuters dataset [6].

The Reuters dataset that is a collection of news published by Reuters agency into a XML format. All the preprocessing steps for transforming the dataset from plain text into an arff

format were done into a different java application. The arff format that must be given to WEKA contain a list with all attributes used into the dataset (defined with name and type) and after the "@data" directive a list of each sample (one on a line) with values for each attribute. We prefer that the last attribute is the class (if the document is contained into a specific class or not in).

The format for the arff file is:

```
@relation Reuters
@attribute 'A0' numeric
@attribute 'A1' numeric
….
@attribute 'A6998' numeric
@attribute 'A6999' numeric
@attribute 'class' {'yes','no'}
@data
2,2,1,1,1,…,0,0,0,0,0,no
0,0,1,0,1,…,0,2,0,1,0,yes
…
```

For this component, only the input file needs to be specified, as in Figure 2. The file needs to contain the entire dataset (both the training and the testing part).

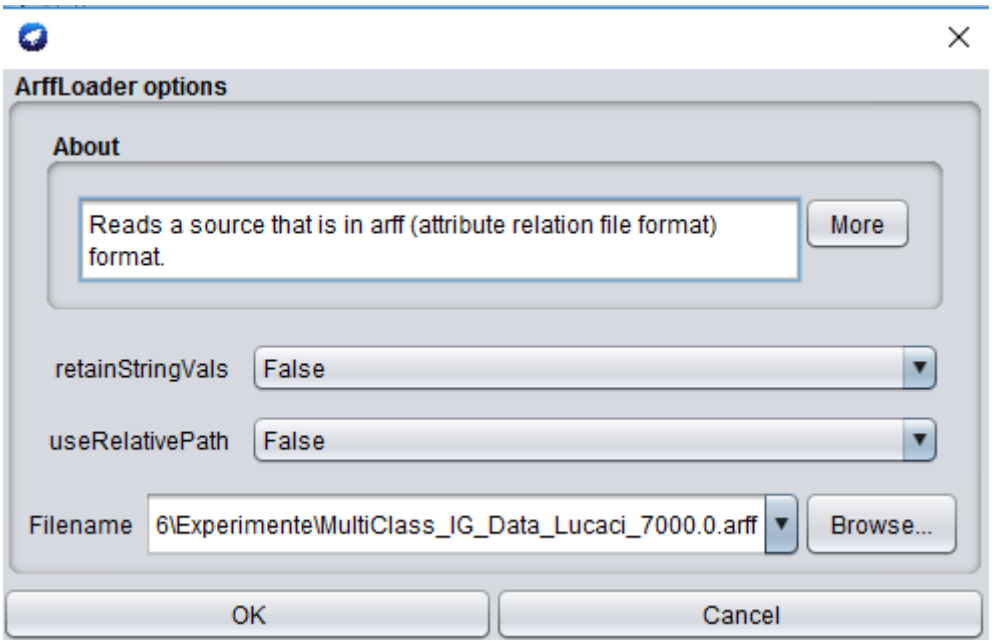


Figure 2. The ArffLoader Component

The two existing configurations are only for specify in what format the string arguments will be kept in memory (as string or not) and how the path for the input file is specified (useful when the experiment is moved in other part).

## 2.2 The ClassAssigner Component

After choosing the dataset, you have to use a component where to specify what attribute is considered as the desired class in the dataset (because we talk about a supervised learning)[5]. Such a component can be found into the "Evaluation" folder. The component is included in this folder because it is related to the evaluation algorithm and depends on how the evaluation will be done. The only configuration that needs to be done in this component is to select from a list the name of the attribute that is considered to be the class. In the presented experiments we have considered that our documents are into the class (and labeled with "yes") or not in the class (and labeled with "no"). Thus, we have considered a binary classification. The ArffLoader component permits you to specify a dataset with more classes, not only for binary classification.

In this framework, in order to establish what type of data will be transferred, a connection between components needs to be realized. Between ArffLoader and ClassAssigner components a "*dataSet*" connection should be used.

## 2.3 Attribute Selection Component

We decide to put this component in our experiment because we want to have the possibility to choose only a reduced number of attributes. We use the AttributeSelection component for selecting the best features. The component is in the Filters tab in the supervised area.

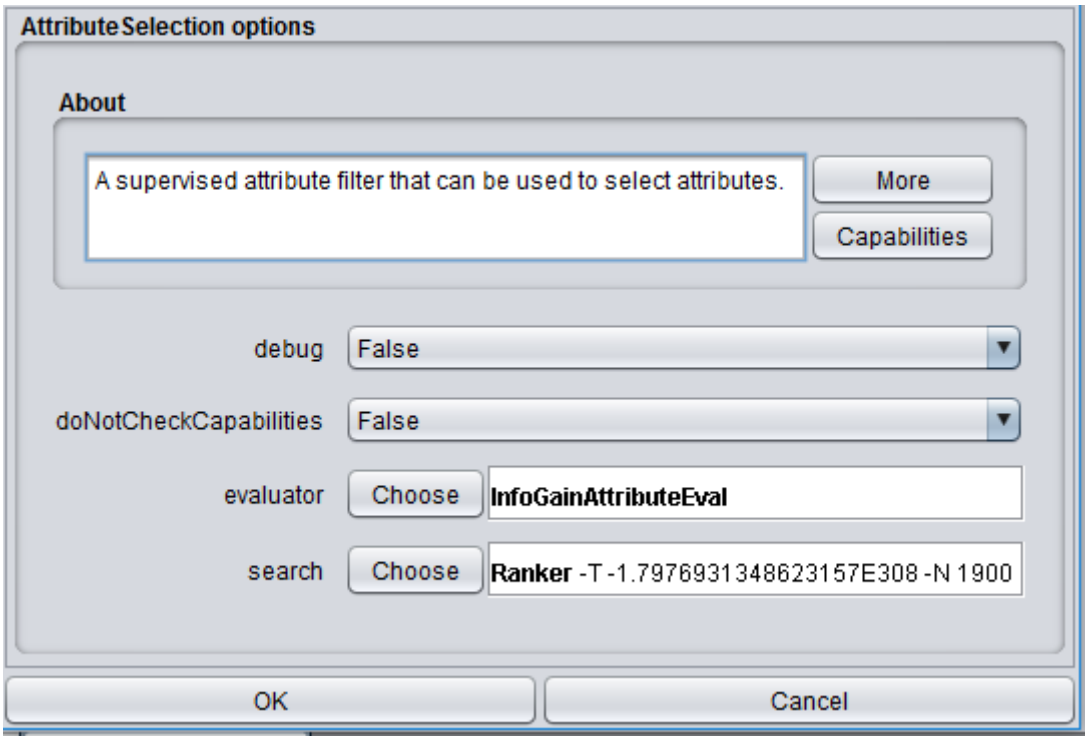


Figure 3. Attribute Selection Component

This component is a supervised attribute filter that can be used to select desired number of attributes. It is very flexible and allows various search and evaluation methods to be combined. In the evaluator property we can determine what method of attribute selection is used. The component supports attribute selection methods like: *InfoGainAttributeEval*, *GainRatioAttributeEval*, *OneRAttributeEval*, *PrincipalComponents* analyzer and more

others [2,3]. We have used the Information Gain Attribute Evaluation method. In the search property the method used in selecting the best attributes is specified (i.e. *Ranker*, *BestFirst* or *GreedyStepwise*). For the Ranker search method, that we have choose, the characteristic window as in figure 4 can to be configured. The Ranker component ranks attributes by their individual evaluations.

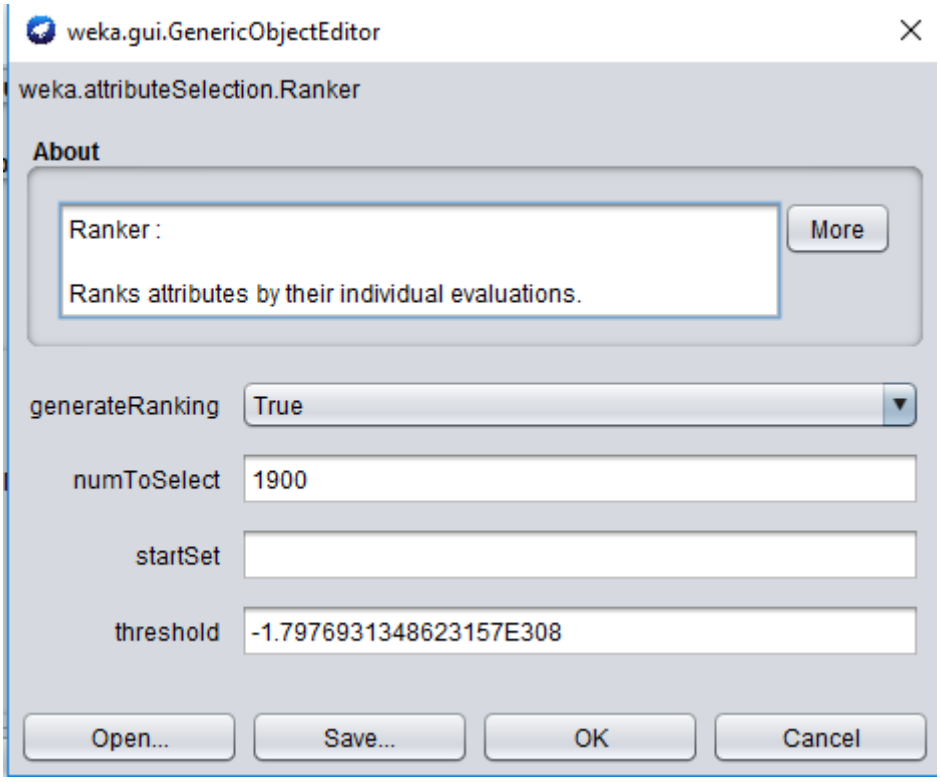


Figure 4. The Ranker properties

The *generateRanking* true is an imposed option. In the *numToSelect* field we can specify the number of attributes that are retained after selection. The default value (-1) indicates that all attributes are retained. In the field *startSet* we can specify a set of attributes that are overpassing the Ranker. When generating the ranking, Ranker will not evaluate the attributes that are in this list. In the last characteristic the *threshold* we can specify the threshold by which the attributes are discarded [2]. In our experiments we have used the default value for the threshold and have changed the numToSelect as desired in the range 200-5500 as presented in the experimental results section.

In the connections between the previous presented components we have selected as option to transfer form one component to another the *dataSet* option.

## 2.4 TrainTestSplitMarker

Until now we work with the entire dataset. Now we need a specific component that permits us to split randomly our dataset into a training part and a testing part. For this we can use *TrainTestSplitMaker* component that can be found in the Evaluation folder. This component is from *Evaluation* tab because it is also part of the evaluation process. This component permits at configuration to specify the percentage of training dataset and the random seed in the input file. This component generates two different outputs: one output

contains the training part form the dataset and the other contains the testing part. Therefor between this component and the next one we need to connect both outputs.

The Weka also permits to implement the CrossValidation idea, having a different component for this in the *Evaluation* tab, called *CrossValidationFoldMaker*. This component permits us to specify the number of experiments that are made with different splitting's of the dataset before showing the average result.

## 2.5 Classifier NaiveBayes

The WEKA framework contains a lot of learning algorithms, as classifier, clustering and association algorithms. The classifier algorithms have a specific tab with the *Classifier* name where a lot of algorithms from different categories (as *bayes*, *rules*, *trees*, *lazy* and more) can be found[3, 4]. WEKA has also a *Clusterers* tab with learning algorithms as *EM*, *Hierarchical*, *Simple KMeans* and more. For our experiments we use a classifier algorithm because we have a dataset that is already classified. We chose NaiveBayes classifier algorithm because it has a small number of characteristics that need to be specified.

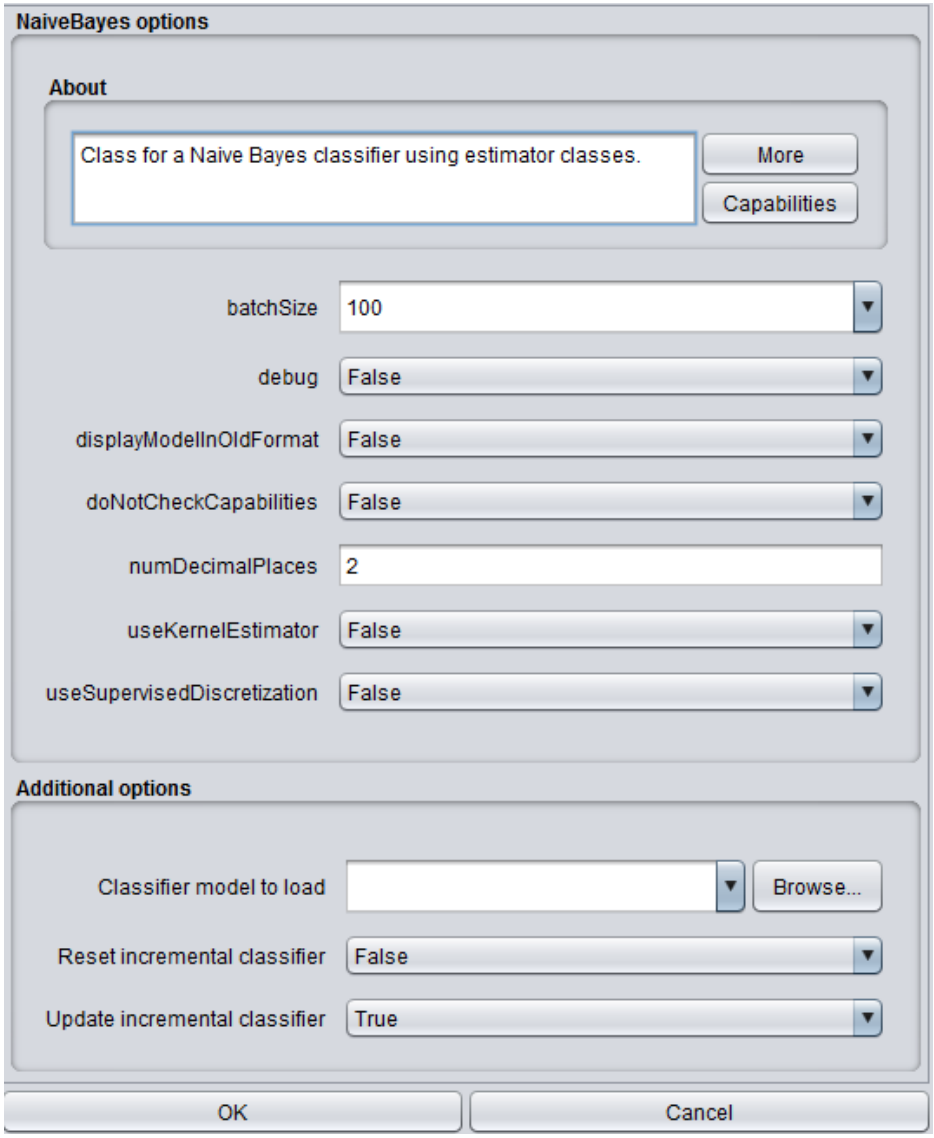


Figure 5. Naïve Bayes characteristics

This component is form *Classifiers* tab, in *bayes* region and is an implementation of the Naive Bayes classifier using class estimators. Precision of numeric estimators are computed based on the analysis of the training dataset. For all the parameters for this component (as *debug*, *displayModelInOldFormat*, *useKernelEstimator*, *useSupervisedDiscretization* and more) we have used the default values.

### 2.6 Classifier Evaluation

The previously presented component implements a learning algorithm. For evaluating the learning performance we need to use an evaluation component, from the *Evaluation* tab. We chose the *ClassifierPerformanceEvaluator,* that is designed to evaluate classifier algorithms. As configuration for this component we need to specify the evaluation metrics that has to be computed. The WEKA has a lot of evaluation metrics already implemented (as *accuracy*, *precision*, *recall*, *f-measure*, *TrueRate*, *NegativeRate* [9]). Default all 25 evaluation metrics already implemented in weka are selected. In our experiments we use only precision, recall accuracy and true rate.

### 2.7 TextViewer

For showing the results the WEKA propose a lot components grouped in the *Visualization* tab. Some components show the results graphically, while others write all the results in the text files. We chose for our application to write the results into a text file.

## 3. Conclusions

We have studied and present in this paper some of the most important visual components that are available in the WEKA framework for the previously presented steps. These components are: “Arff Loader”, “Attribute Selection”, “Normalize”, “Train Test Split Maker”, a lot of classifier algorithms, “Performance Evaluator” and “Text Viewer”. In order to prove the functionality of the visual framework in text document classification we have made and present some experiments. The most important advantage of the visual WEKA framework is the possibility to test different approaches without programming abilities.

## 4. References

[1] Ian H. Witten, Eibe F., Hall, M. A., Pal C.J., Data Mining – Practical Machine Learning Tools and Techniques with Java Implementation, Morgan Koufmann Press, 2000
[2] Han, J., Kamber, M., - Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2001;
[3] Manning, C., - An Introduction to Information Retrieval, Cambridge University Press, 2009;
[4] Tom. M. Mitchell, Machine Learning, The McGrow-Hill Companies, 1997
[5] Mitkov R., The Oxford Handbook of Computational Linguistics, Oxford University Press, 2005;

[6] Misha Wolf and Charles Wicksteed – Reuters Corpus: http://trec.nist.gov/data/reuters/reuters.html, accessed in 03.2016
[7] http://www.cs.waikato.ac.nz/ml/weka/, accessed in 03.2016
[8] http://www.cs.waikato.ac.nz/ml/weka/documentation.html, accessed 03.2016
[9] https://en.wikipedia.org/wiki/Precision_and_recall, accessed 03.2016