Illustrative example of running RSSalg software: News 2x2 dataset

The purpose of this document is to illustrate the main functionalities of RSSalg software on the example of running News2x2 experiment. Section 1 describes the problem we are trying to solve and the experimental setting for evaluating the solution. It also lists the applied algorithms and the values of algorithm parameters we want to use in the experiment. Section 2 describes how to install and start RSSalg software. Section 3 describes how to set the experiment and parameter values in *RSSalg* software. Finally, section 4 describes how to run the set experiment and interpret the software output.

1 Problem setting

News 2x2 dataset is a semi-artificial dataset introduced in [1]. It was artificially created to be a binary classification problem with class-conditional independence, ideal for testing the performance of co-training [2]. The examples are categorized in two classes named “positiveClass” and “negativeClass”.

Our goal is to minimize the number of examples needed for quality classification. That is, starting from a small number of labeled instances and a sufficiently large number of unlabeled instances our goal is to achieve roughly the performance of a supervised classifier trained using both labeled and unlabeled instances with the correct label assigned.

1.1 Experimental setting

We will test our solution using a 10-fold-cross validation procedure introduced in [3]. In this experiment, the data is divided in 10 stratified folds. In each round of fold-cross validation, a different fold will be used for the selection of initial labeled data: 5 instances belonging to “positiveClass” and 5 instances belonging to the “negativeClass”. The rest of the data from this fold, as well as five adjacent folds will be used as unlabeled data. The four remaining folds are used as test data.

The total number of instances in News2x2 dataset is 2000 (200 per fold). Thus, starting from just 10 labeled instances we strive to achieve the performance of a supervised classifier trained on 1200 instances (6 folds used as labeled and unlabeled data). News2x2 is a balanced dataset with equal number of instances belonging to “positiveClass” and “negativeClass”, thus, we will use accuracy as the measure of performance.

1.2 Tested algorithms

In this experiment we will test the performance of the following algorithms:

1. Supervised settings:
   - L_{acc}: Supervised learner trained on the labeled portion of the data (10 labeled instances)
   - All_{acc}: Supervised learner trained on both labeled and unlabeled instances with correct label assigned (1200 labeled instances)
2. Semi-supervised settings (trained using 10 labeled and 1990 unlabeled instances):
   - *Natural*: co-training run with natural feature split. This setting is applicable on the problem domains where the features naturally divide in two disjoined sets, e.g. based on the existence of two independent information sources that describe the same data. News2x2 dataset is semi-artificially created to have a natural feature split [1].
   - *Random*: co-training run with random feature split. We will report the average performance co-training of 100 different random feature splits.
   - *MajorityVote*: a majority vote of 100 co-training classifiers trained using 100 different random splits (same classifiers used in *Random* setting)
   - *RSSalg*: *RSSalg* as introduced in [4]. RSSalg is applied on the same 100 co-training classifiers created in *Random* setting.

- *RSSalg$_{best}$*: *RSSalg* with thresholds optimized on test data (*RSSalg$_{best}$* in [4]). *RSSalg$_{best}$* serves as the indication of upper bound performance of *RSSalg*. It is used to test the threshold optimization procedure in *RSSalg*.

1.3 Co-training parameters

Each tested co-training style algorithm will use the following parameter values:

- the number of iterations of co-training algorithm (denoted *k* in [2]) is 20;
- a small unlabeled pool (denoted *u*' in [2]) of 50 instances is used;
- growth size (denoted as *p* and *n* in [2]): in each iteration of co-training 5 instances most confidently labeled as "positiveClass" and 5 instances most confidently labeled as "negativeClass" will be added to the initial training set
- Naïve Bayes classifier will be used for both views

In *Random*, *MajorityVote* and *RSSalg* settings, each round of co-training is performed using the parameters listed in this section. All of these parameter values were empirically chosen. The same parameter values were used in [3] when applying co-training on News2x2 dataset.

2 Installing and starting RSSalg software

RSSalg software requires Java Runtime Environment (JRE) 1.7 or above[1]. Once Java is installed, you can start RSSalg software:

1) By running the JAR packaged executable `RSSalg.jar` located inside `/dist` folder[2].
2) By loading the source code located inside `/src` folder[3] inside your favorite IDE. Note that *RSSalg* software implementation depends on Weka (version 3.4 or above). Thus, you will need to add weka.jar library into your project[4].

In order to start experiment execution, user must define several property files containing the desired values for algorithm parameters and experiment setting. The detailed description of the required files will be given in section 3 and commented examples of these files for running News2x2 experiment described in this document can be found in `dist/data/News2x2/experiment` folder[5].

RSSalg can be run in two modes: GUI (Graphical User Interface) mode (section 2.1) or console mode (section 2.2). GUI mode allows for the easy, intuitive creation of the needed property files, as well as algorithm execution. Alternatively, user may choose to create the property files by hand using his favorite text editor. Console mode assumes that the needed property files are already created.

2.1 GUI mode

In order to run RSSalg in GUI mode, in command prompt navigate to the location of the downloaded executable *RSSalg.jar* and type the following command:

```
java –jar RSSalg.jar
```

If you chose to load the source code in your favorite IDE, a starting point for running RSSalg GUI is the class `RSSalgFrame`, located inside package `application.GUI`.

[1] To install Java go to http://www.oracle.com/technetwork/indexes/downloads/index.html#java

[2] https://github.com/slivkaje/RSSalg-software/tree/master/dist

[3] https://github.com/slivkaje/RSSalg-software/tree/master/src

[4] Weka can be downloaded from http://www.cs.waikato.ac.nz/ml/weka/downloading.html. In order to compile *RSSalg software* code, Weka does not need to be installed; only the library weka.jar needs to be added to the project build path.

[5] https://github.com/slivkaje/RSSalg-software/tree/master/dist/data/News2x2/experiment

2.2 Console mode

In order to run the console mode, user must firstly define the property files. In order to run *RSSalg software* in console mode, in command prompt navigate to the location of the downloaded executable *RSSalg.jar* and type the following command:

```
java –jar RSSalg.jar <properties_folder> <experiment_properties>
```

where:

`<properties_folder>` represents the folder containing the following property files (needed for execution of all experiments): data.properties, cv.properties, co-training.properties and GA.properties. For the detailed description of these files please refer to section 3.

`<experiment_properties>` represents the property file containing the desired settings for the concrete experiment that should be run (which algorithm to run, etc.). The detailed description of this file can be found in section 3.4.

For example, if a user wishes to execute the L_{acc} setting on News2x2 introduced in section 1.2 he should download the `/data` folder[6] and place it in the same folder as `RSSalg.jar` and type the following command:

```
java -jar RSSalg.jar ./data/News2x2/experiment experiment_L.properties
```

If you chose to load the source code in your favorite IDE, a starting point for running RSSalg in console mode is the class `StartExperiment`, located inside the `application` package.

3 Specifying parameter values in RSSalg software

RSSalg software was designed to be easily configurable - user is able to supply the values for all co-training and RSSalg parameters through the usage of property files. Algorithm properties are organized in 5 basic groups:

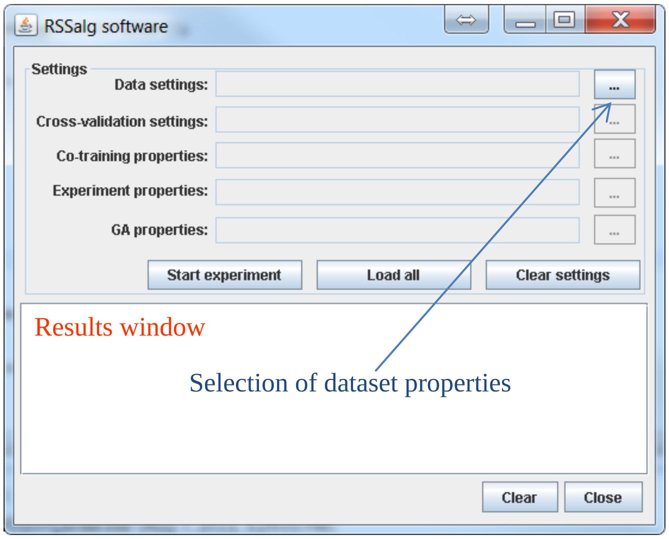
- **dataset settings** provide the basic dataset properties: dataset file location, the names of class and id attribute, etc. They can be found in the file named 'data.properties' described in section 3.1.6
- **cross-validation experiment settings** dictate how the cross-validation experiment will be constructed: the number of folds used in n-fold-cross validation, number of initially labeled instances per each class, etc. These settings are saved in the file entitled 'cv.properties' described in section 3.2.1
- **co-training settings** dictate the properties of co-training algorithm: number of iterations, size of the unlabeled pool *u*', etc. These settings are saved in the file entitled 'co-training.properties' described in section 3.3.1
- **experiment settings** define the conducted experiment: the algorithm that should be run, performance measures to be calculated, etc. These settings are saved in property file described in section 3.4.7
- **genetic algorithm settings** are only needed for *RSSalg* experiment and provide the basic properties of genetic algorithm that is the bases of threshold optimization process in *RSSalg* [4]. These settings are saved in the file entitled 'GA.properties' described in section 3.5.1.

The applications GUI allows the easy creation of the needed propety files. Figure 1 illustrates the main window that is opened on the startup of *RSSalg software* application. In order to run the News2x2 experiment, the user must firstly provide the dataset properties by selecting „...“ button on the right hand side of „Data settings:“ label (see figure 1).

3.1 Specifying dataset settings

The dialog for dataset properties entry is illustrated in Figure 2. User can choose between the two options: (1) prepare a new cross-validation experiment (explained in section 3.1.1) or (2) load previously prepared cross-validation experiment from files (explained in section 3.1.2). After selection, user is able to define general dataset properties (class and id attribute names, random number generator seed, etc.) which is described in section 3.1.3. Section 3.1.4 describes how the user can specify a classification model used for each of the individual views. Finally, section 3.1.5 describes how to save the selected settings and describes the resulting property files. Section 3.1.5 also describes how to load a previously created property file.

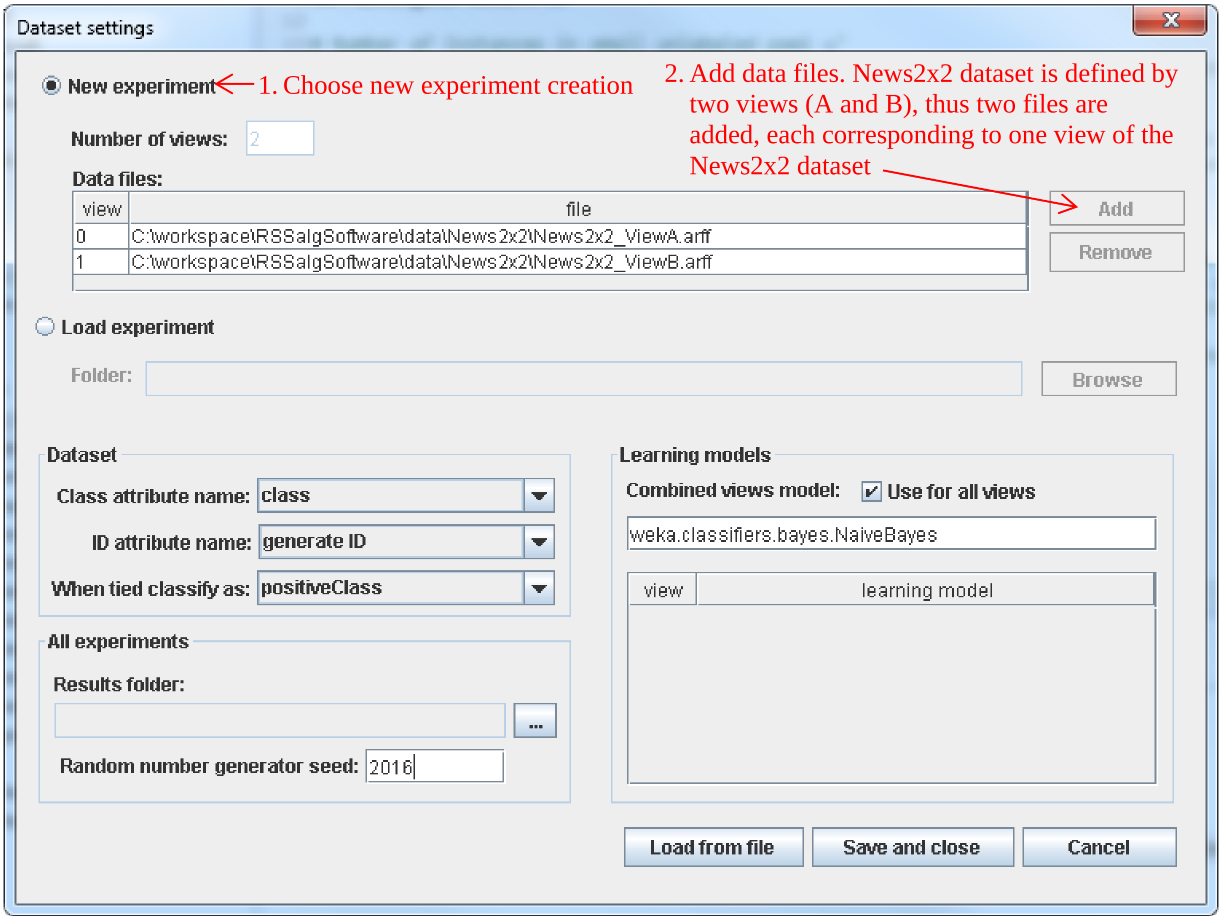
[6] Located inside https://github.com/slivkaje/RSSalg-software/tree/master/dist folder



Results window

Selection of dataset properties

Figure 1 The main window of *RSSalg software* application



● New experiment

1. Choose new experiment creation

2. Add data files. News2x2 dataset is defined by two views (A and B), thus two files are added, each corresponding to one view of the News2x2 dataset

Number of views: 2

Data files:

view	file
0	C:\workspace\RSSalgSoftware\data\News2x2\News2x2_ViewA.arff
1	C:\workspace\RSSalgSoftware\data\News2x2\News2x2_ViewB.arff

○ Load experiment

Folder:

Dataset

Class attribute name: class

ID attribute name: generate ID

When tied classify as: positiveClass

All experiments

Results folder:

Random number generator seed: 2016

Learning models

Combined views model: ☑ Use for all views

weka.classifiers.bayes.NaiveBayes

view	learning model

Figure 2 Dataset settings input form: creation of the new cross-validation experiment for News2x2 dataset

3.1.1 Creating a new cross-validation experiment

Figure 2 illustrates the process of preparing a new cross-validation experiment for News2x2 dataset. Choosing the „New experiment“ radio button will allow the user to define the dataset that is to be partitioned in the desired number of folds for the cross-validation experiment.

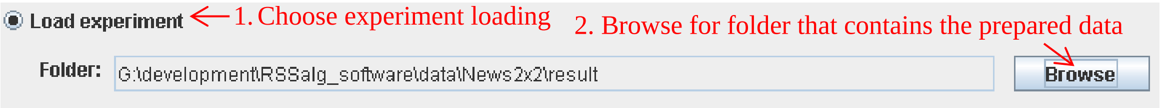
By selecting the “Add” button user can provide one or two dataset files[7]. It is assumed that all user provided dataset files contain same instances but unique sets of attributes (i.e. they correspond to the different views of the same dataset). The only common attributes for all provided datasets should be the class attribute whose presence is obligatory in all provided views and the and id attribute whose presence is optional[8].

Currently, RSSalg software only allows two views of the data. If the user provides only one dataset file, RSSalg software will automatically create the second view by using only class and id attribute. Later, during experiment execution, other features might be added to the second view if needed (e.g. when performing a random feature split). If the user provides two views, the provided features separation is considered to be the „natural feature split“ of the dataset.

The prepared cross-validation experiment will be saved to disk inside the folder specified as the *result folder*. Please refer to section 3.1.2 for the structure of the recorded cross-validation experiment and to section 3.1.3 for the explanation on specifying the *result folder*.

3.1.2 Loading an already prepared cross-validation experiment

User is allowed to load an already prepared cross validation experiment by selecting the „Load experiment“ radio button (figure 3).



● Load experiment

Folder: G:\development\RSSalg_software\data\News2x2\result

1. Choose experiment loading

2. Browse for folder that contains the prepared data

Figure 3 Loading an already prepared experiment (part of the dataset settings input form, figure 2)

User must define the directory to load the prepared experiment from by clicking on the „Browse“ button. The chosen directory must have the following structure (figure 4): it contains subdirectories that are named fold_0, fold_1, etc. Subdirectory fold_*i* contains all the data needed for running and testing the algorithms on the *i*th fold of *n*-fold-cross validation experiment, *i* ϵ {0,..,n-1}. Each subdirectory must contain 3 × the number of views ARFF files, for example, in the case of 2 views the files are named:

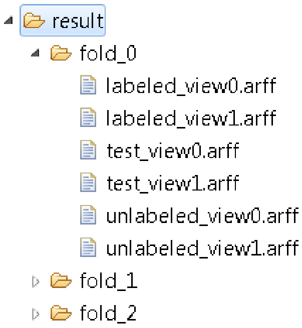
- labeled_view0.arff, labeled_view1.arff (two views of labeled data);
- unlabeled_view0.arff, unlabeled_view1.arff (two views of unlabeled data);
- test_view0.arff, test_view1.arff (two views of test data);

Optionally, two more files may be provided named pool_view1.arff and pool_view2.arff, which correspond to a small pool of unlabeled data *u’*.

It is assumed that the two views of the same data contain the same instances described by different sets of attributes. The only common attributes for all views should be the class attribute whose presence is obligatory in all views and the and id attribute which is optional.

[7] Currently, only ARFF file format is accepted. Originaly developed for Weka [5], ARFF format is widely used in many data mining tools, e.g. Weka and RapidMiner (http://rapidminer.com/). Using these datamining tools it is easy to convert the datasets written in other file formats to ARFF format.

[8] If the dataset does not contain the id attribute it is assumed that the provided views have the same ordering of instances, i.e., the *n*-th instance in the first view corresponds to the *n*-th instance in all other views



```
result
  fold_0
    labeled_view0.arff
    labeled_view1.arff
    test_view0.arff
    test_view1.arff
    unlabeled_view0.arff
    unlabeled_view1.arff
  fold_1
  fold_2
```

Figure 4 File structure of the directory *result* that contains an already prepared 3-fold-cross-validation experiment for the dataset that has two views

3.1.3 Defining general data properties

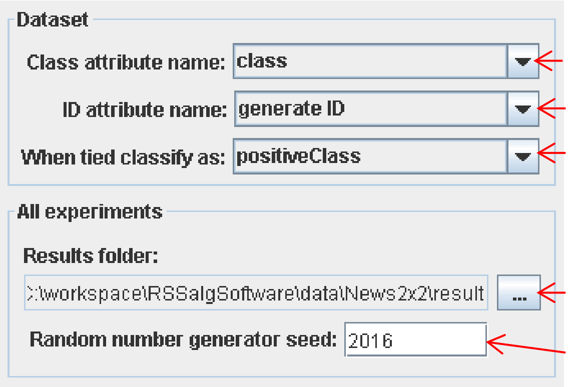
After the selection of whether to create a new or load an existing cross-validation experiment, the general data properties listed in figure 5 should be provided:

- name of the class and id attribute – selection of the dataset will cause the list of possible class and id attribute names to populate. As class and id attributes are the only attributes that should be present in both views, the lists selections are populated with attributes present in both views. The class attribute name is also limited to nominal attributes. If the id attribute is not present in the dataset ('generate ID' choice, figure 5), the dataset will be automatically tagged with id attribute named 'ID' (numerical, incremental).

- optionally, one can specify the class which should be assigned to the instances for which the algorithm provides equal probability for all classes;

- the result folder – the directory to which the created experiment will be written (according to the file structure described in section 3.1.3) along with the experiment results.

- the random number generator seed – this parameter ensures that the sequence of created random numbers stays the same for each of the experiments, making all experiments exactly replicable.



Dataset

Class attribute name: class

ID attribute name: generate ID

When tied classify as: positiveClass

All experiments

Results folder:

:\workspace\RSSalgSoftware\data\News2x2\result

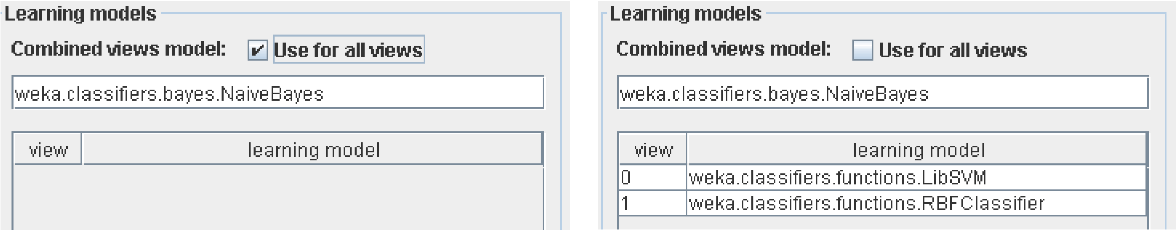
Random number generator seed: 2016

1. Specify the class attribute name
2. Specify the ID attribute name
3. Specify the name of the class to classify the instances to when all class probabilities are equal
4. Specify the results folder – all experiment data and results will be recorded in this folder
5. Specify the random number generator seed

Figure 5 Entering the general data properties (part of the dataset settings input form, figure 2)

3.1.4 Choosing classification models

User can choose to use the same classification model for the combined classifier as well as the individual views (as shown in figure 6, left) or to use different classification models (as shown in figure 6, right). User can provide arbitrary classifiers from Weka libraries [5] or any class implementing Weka's *Classifier* interface. A full name of the implementation class as well as the package structure must be provided.



Learning models

Combined views model: ☑ Use for all views

weka.classifiers.bayes.NaiveBayes

view	learning model

Learning models

Combined views model: □ Use for all views

weka.classifiers.bayes.NaiveBayes

view	learning model
0	weka.classifiers.functions.LibSVM
1	weka.classifiers.functions.RBFClassifier

Figure 6 Choosing classification models (part of the dataset settings input form, figure 2). Left: the same model is chosen for the combined classifier and for all individual views; Right: different classification models are used: SVM for the first view, RBF for the second view and Naïve Bayes for the combined classifier.

Individual learning models are used for training individual view classifiers in co-training. A combined model is the model used for training on the whole feature set (merged views), i.e. a training model used in supervised experiments *L* and *All*, as well as the model used for the final classifier in *RSSalg*.

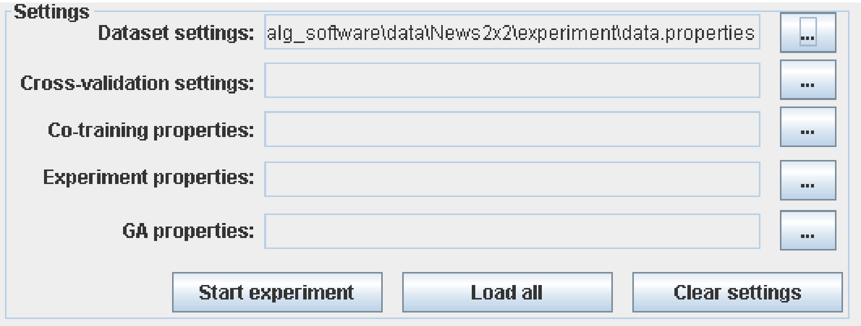
For News2x2 experiment, a Naïve Bayes classifier is specified to be used for both individual views as well as for the final classifier, as shown in in figure 6, left.

3.1.5 Saving/loading the data settings

After specifying the desired settings, user can save them in *.properties* file by clicking on the “Save and close“ button on the bottom right side of dataset settings dialogue (figure 2). User can specify the directory the settings will be saved to in the file automatically named ‘data.properties’.

User can also load the existing *.properties* file by clicking on „Load from file“ button located on the bottom right side of dataset settings dialogue (figure 2). User will be prompted to select a directory which must contain the file named ‘data.properties’ from which the properties will be loaded.

Once the dataset properties have been successfully saved, application returns to the main window and the user is allowed to specify other properties, figure 7.



Settings

Dataset settings: alg_software\data\News2x2\experiment\data.properties

Cross-validation settings:

Co-training properties:

Experiment properties:

GA properties:

Figure 7 The main window of the application after the successful save of data properties

3.1.6 The resulting data.properties file

Table 1 lists all available data properties as written in ‘data.propertes’ file, their meaning and possible values. An example of data.properties file with comments is distributed with RSSalg software and can be found in `/dist/data/News2x2/experiment` folder.

Property name	Description	Value
resultFolder	Folder in which the performed cross-validation experiment and results will be recorded	String representing a folder location
classNames	Names of the categories	Quoted Strings separated with whitespace character
combinedClassifier	The combined classifier: supervised model trained on full attribute set acquired by merging all views. It is used in L_{acc} and All_{acc} settings, as well as the final classifier in *RSSalg*	String: a full-package name of the class implementing Weka's `Classifier` interface
classifiers	Classification models used for individual views. If only one classifier is listed and there is more than one view, the same classification model will be used for all individual views. Otherwise, the number of strings in the *classifiers* parameter as and *dataFiles* parameter must be same	Quoted Strings separated with whitespace characters. Each String should be a full-package name of the class implementing Weka's `Classifier` interface
noViews	The number of views (currently fixed to 2)	2
classAttributeName	Class attribute name	String
dataFiles	Files containing the data. Each file corresponds to one view of the data. The views should contain same instances but different sets of attributes. The only attributes which should appear in both views are class and (optionally) id attribute (class and id values of one instance should be same in all views). If the id attribute is not present in the dataset it is assumed that the *n*th instance of the first view corresponds to the *n*th instance in all other views.	Quoted Strings separated with whitespace characters. Each string should be an ARFF file location
randomGeneratorSeed	Seed for random number generator	Integer
loadPresetExperiment	If true, the experiment will be loaded from the *resultFolder* (properties *dataFiles*. *noViews* and all properties from cv.properties file will be ignored). If false, a new cross-validation experiment will be prepared	Boolean value (true/ false)
idAttributeName	ID attribute name. If the property value is unspecified, an id attribute named 'ID; will be automatically generated	String

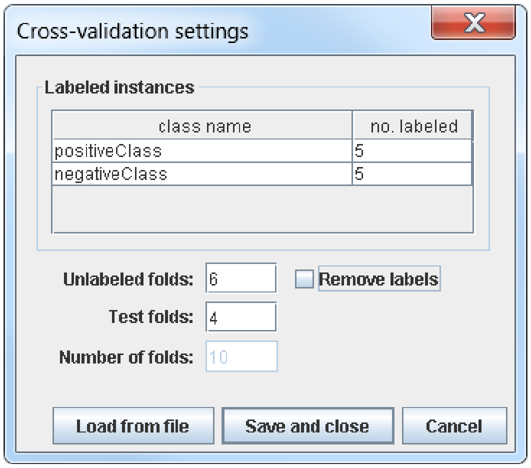
Table 1 The meaning of properties saved in 'data.properties' file

3.2 Specifying cross-validation experiment settings

The cross-validation experiment settings are used if user chose the "New experiment" option (chapter 3.1.1). Otherwise (if the user chose the "Load experiment" option, section 3.1.2), user will still be able to access these settings in order to review the loaded experiment, but will be unable to change them.

Figure 8 shows the dialogue for cross-validation settings entry populated with the settings needed for News2x2 experiment: in each round of 10-fold-cross validation, 6 adjacent folds will be used as the unlabeled data and 4 adjacent folds will be used as the test data (the total number of folds is always equal to the number of unlabeled and test folds). The labeled dataset will be constructed by randomly selecting instances labeled as 'positiveClass' and 5 instances labeled as 'negativeClass'. Application will automatically populate the possible category names ('class name' column in the table on figure 8), the user only needs to specify the number of labeled instances per each class.

User can also select the option of removing the value of the class attribute for all instances that belong to the unlabeled set. However, in this case the supervised experiment *All* which uses the labels of unlabeled data cannot be ran properly (it will return the same result as the *L* experiment as the model will be trained on the same small labeled set). Keeping or removing the labels from unlabeled data does not affect the results of any other experiments that do not use these labels.



Cross-validation settings

Labeled instances

class name	no. labeled
positiveClass	5
negativeClass	5

Unlabeled folds: 6 □ Remove labels

Test folds: 4

Number of folds: 10

Figure 8 The cross-validation input form with values entered for the News2x2 experiment: 10 folds will be used for the cross-validation experiment, 6 adjacent folds as unlabeled data and 4 adjacent folds as the test data. As for the labeled data, 5 random positive and 5 random negative examples will be chosen from the first of the folds used as unlabeled data

User can save the specified cross-validation settings by clicking on the „Save and close“ button on the bottom right side of cross-validation input form (figure 8). User is prompted to specify the directory the settings will be saved to in the file automatically named ‘cv.properties’.

3.2.1 The resulting cv.properties file

Table 2 lists all available cross-validation properties as written in ‘cv.propertes’ file, their meaning and possible values. An example of cv.properties file with comments is distributed with RSSalg software and can be found in `/dist/data/News2x2/experiment` folder.

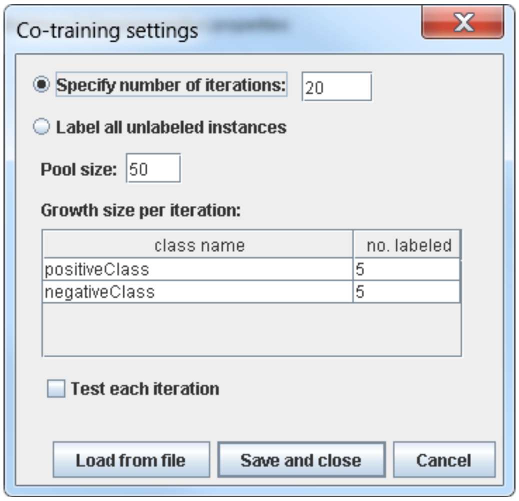
Property name	Description	Value
className	Names of the categories	Quoted Strings separated with whitespace characters
noLabeled	Number of labeled examples randomly chosen per each class – the *i*th value from the list corresponds to the *i*th category listed in *className* parameter	Integer values separated with whitespace characters
noTest	Number of adjacent folds merged and used as test data	Integer value
noUnlabeled	Number of adjacent folds merged and used as unlabeled data	
noFolds	Total number of folds (*n*) for *n*-fold-cross validation	Integer value
removeLabels	If true the label will be removed from all unlabeled instances (this will disable the execution of *All* experiment)	Boolean value (true/false)

Table 2 The meaning of properties saved in ‘cv.properties’ file

User can load the existing *.properties* file by clicking on the „Load from file“ button on the bottom right side of cross-validation input form (figure 8). User will be prompted to select a directory which must contain the file named ‘cv.properties’ from which the properties will be loaded.

3.3 Specifying co-training settings

The co-training settings are used for all underlying co-training experiments. For example, in *RSSalg* a predefined number of co-training classifiers is trained, each using these settings. The same co-training settings are used for running *Natural* setting or if running the *Random* setting. The input form for co-training settings entry is shown in figure 9.


Co-training settings

● Specify number of iterations: 20

○ Label all unlabeled instances

Pool size: 50

Growth size per iteration:

class name	no. labeled
positiveClass	5
negativeClass	5

□ Test each iteration

Figure 9 The co-training settings input form with values entered for the News2x2 experiment: co-training will be ran for 20 iterations using the small unlabeled pool (*u'*) of 50 instances. In each iteration of co-training, each underlying classifier will select and label 5 most confidently labeled positive examples and 5 most confidently labeled negative examples

As stopping criteria for co-training, user can either specify the desired number of iterations (20 for the News 2x2 dataset, figure 9), or specify to stop co-training when all unlabeled instances have been labeled (option „Label all unlabeled instances"). The size of the small unlabeled pool *u'* may be specified (50 for the News2x2 experiment). If the specified size of the small unlabeled pool is 0, the pool won't be used and the underlying co-training classifiers will select and label instances directly from the unlabeled set.

The „Growth size per iteration" defines the number of most confidently labeled instances per each class that will be labeled and transferred to the labeled set in each co-training iteration. The "class names" column is automatically populated and user only needs to specify the number of instances.

Check box „Test each iteration" allows the user to oversee the progress of each co-training classifier training by testing the classifier in each iteration. These results are documented in the log files named 'CTlog_split_*i*.txt' in the results folder (specified through data settings, chapter 3.1.3). For example, if the *RSSalg* experiment was ran using 3 different random splits, three different co-training classifiers will be trained and the progress of their training will be saved in 3 different log files, one for each split, as shown in figure 10.

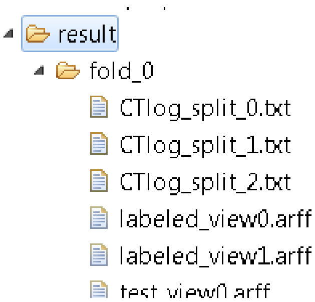
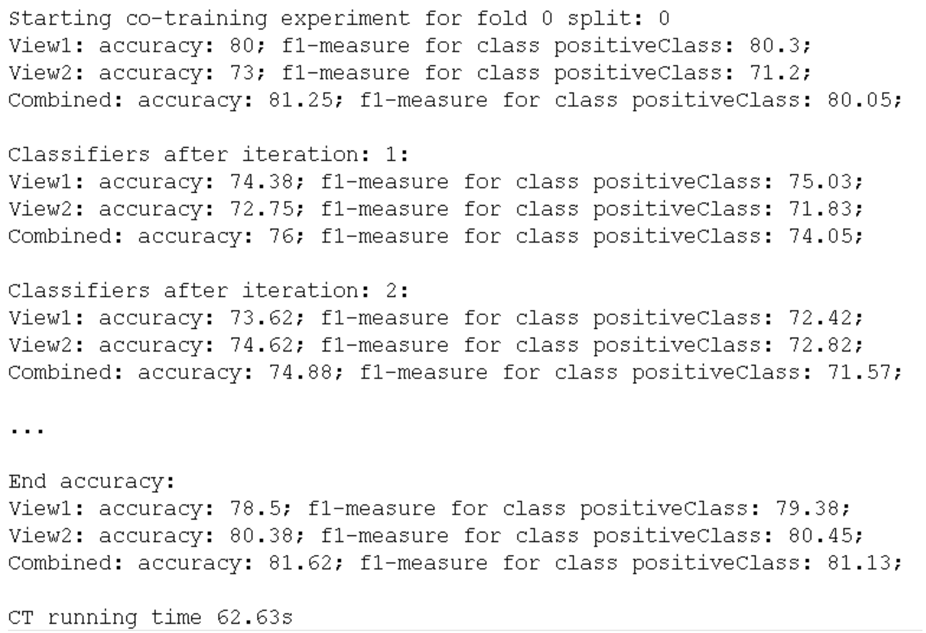

```
result
  fold_0
    CTlog_split_0.txt
    CTlog_split_1.txt
    CTlog_split_2.txt
    labeled_view0.arff
    labeled_view1.arff
    test_view0.arff
```

Figure 10 The results of testing each iteration of co-training classifiers trained in *RSSalg* using 3 different random splits are saved in log files denoted as 'CTlog_split_*i*.txt', *i* ϵ {0,..,2}

The example of one of these log files is shown in figure 11. Individual co-training classifiers, as well as the combined co-training style classifier[9] are tested in each iteration. The calculated measures (in this example accuracy and *f*1-measure for the 'positiveClass') are defined by settings of the performed experiment (chapter 3.4).

[9] The combined co-training style classifier classifies each test instance by multiplying the class probabilities output by individual underlying classifiers. The instance is assigned the class with the highest probability obtained this way



```
Starting co-training experiment for fold 0 split: 0
View1: accuracy: 80; f1-measure for class positiveClass: 80.3;
View2: accuracy: 73; f1-measure for class positiveClass: 71.2;
Combined: accuracy: 81.25; f1-measure for class positiveClass: 80.05;

Classifiers after iteration: 1:
View1: accuracy: 74.38; f1-measure for class positiveClass: 75.03;
View2: accuracy: 72.75; f1-measure for class positiveClass: 71.83;
Combined: accuracy: 76; f1-measure for class positiveClass: 74.05;

Classifiers after iteration: 2:
View1: accuracy: 73.62; f1-measure for class positiveClass: 72.42;
View2: accuracy: 74.62; f1-measure for class positiveClass: 72.82;
Combined: accuracy: 74.88; f1-measure for class positiveClass: 71.57;

...

End accuracy:
View1: accuracy: 78.5; f1-measure for class positiveClass: 79.38;
View2: accuracy: 80.38; f1-measure for class positiveClass: 80.45;
Combined: accuracy: 81.62; f1-measure for class positiveClass: 81.13;

CT running time 62.63s
```

Figure 11 The example of the log file obtained by testing each iteration of co-training

User can save the specified co-training settings by clicking on the „Save and close“ button on the bottom right side of co-training input form (figure 9). User is prompted to specify the directory the settings will be saved to in the file automatically named ‘co-training.properties’.

User can load the existing *.properties* file by clicking on the “Load from file“ button located on the bottom right side of co-training settings input form (figure 9). User will be prompted to select a directory which must contain the file named ‘co-training.properties’ from which the properties will be loaded.

3.3.1 The resulting co-training.properties file

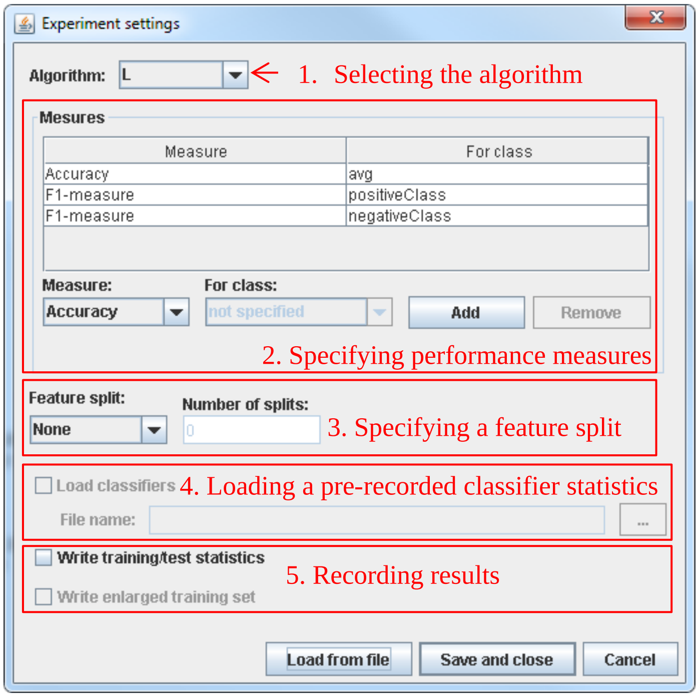
Table 3 lists all available co-training properties as written in ‘co-training.propertes’ file, their meaning and possible values.

Property name	Description	Value
className	Names of the categories	Quoted Strings separated with whitespace characters
Growth size	Number of most confidently labeled examples belonging to each class that will be labeled and transferred to the labeled set in each co-training iteration. The *i*th value from the list corresponds to the *i*th category listed in *className* parameter	Integer values separated with whitespace characters
labelAllUnlabeledData	If true, co-training process will continue until all examples from unlabeled set are labeled (*coTrainingIterations* parameter will be ignored); if false, co-training will run for a predefined number of iterations defined in *coTrainingIterations* parameter	Boolean value (true/false)
coTrainingIterations	Number co-training iterations *k*	Integer value
poolSize	Size (number of instances) of unlabeled pool *u’*. If set to 0, the pool is not used and the instances are sampled directly from the unlabeled set	Integer value
testEachIteration	If true – the obtained co-training classifiers will be tested in each iteration and the results will be written to log files	Boolean value (true/false)

Table 3 The meaning of properties saved in ‘co-training.properties’ file

3.4 Specifying experiment settings

The experiment settings allow the user to define the conducted experiment. The input form for experiment settings entry is shown in figure 12.



Experiment settings

Algorithm: L

1. Selecting the algorithm

Mesures

Measure	For class
Accuracy	avg
F1-measure	positiveClass
F1-measure	negativeClass

Measure: Accuracy

For class: not specified

2. Specifying performance measures

Feature split: None

Number of splits: 0

3. Specifying a feature split

□ Load classifiers

File name:

4. Loading a pre-recorded classifier statistics

□ Write training/test statistics

□ Write enlarged training set

5. Recording results

Figure 12 The experiment settings input form

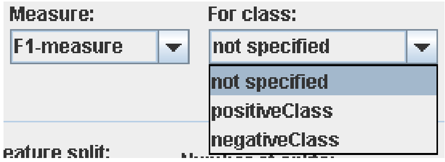
1. Selecting the algorithm

Using the “Algorithm” selection list user can choose to run one of the experiments listed in section 1.2. In the selection list ‘L’ denotes L_{acc} setting, ‘All’ denotes All_{acc} setting, ‘Co-training’ is used for *Natural* and *Random* settings, ‘MV’ is used for *MajorityVote* setting, ‘RSSalg’ is used for running *RSSalg* setting and ‘RSSalg_best’ is used for running $RSSalg_{best}$ setting.

2. Specifying performance measures

RSSalg software is able to calculate several measures of algorithm performance. Currently supported measures are: accuracy, *f*1-measure, precision and recall. User can also specify an arbitrary measure that implements *classificationResult.measures.MeasureIF* interface provided in *RSSalg software* implementation.

For category-specific measures such as *f1*-measure user can specify the category to calculate the measure for or select “not specified” choice in order to obtain the average measure for all classes (e.g. in figure 13 user chooses to obtain an average of *f*1-measure for ‘positiveClass’ and *f*1-measure for ‘negativeClass’).



Measure: F1-measure

For class: not specified

Figure 13 Choosing to calculate an average of *f*1-measure for ‘positiveClass’ and *f*1-measure for ‘negativeClass’ by selecting “not specified”. The category list is automatically obtained from the data settings by examining the provided class attribute

User may provide several measures which will be calculated. For example, in figure 12, user has chosen to calculate the accuracy and *f1*-measure for both positive and negative class in L_{acc} experiment.

3. Specifying a feature split

By specifying values in ‘Feature split’ list choice and ‘Number of splits’ field user can specify the feature split that will be performed before running the chosen algorithm, and the number of times this process will be repeated, respectively.

4. Loading a pre-recorded classifier statistics

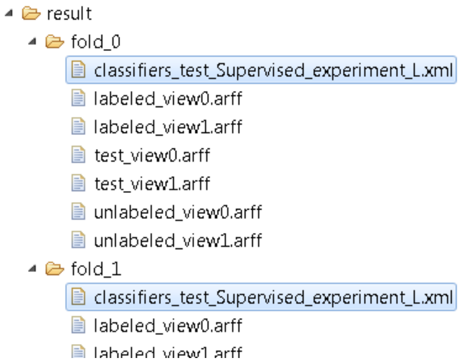
"Load classifiers" check box denotes whether to load a previously recorded classifier statistic or not. Selecting to use the classifier statistic will enable the user to specify the *xml* file that contains the recorded statistic.

A *training* classifier statistic provides the information about each instance *labeled during the training process* of the classifier (instances formally belonging to the unlabeled set, labeled and added to train data by the algorithm). This information encompasses the label that was assigned to the instance by a classifier, as well as the classifiers confidence for each possible category. For example, in each round of co-training, both underlying classifiers are allowed to select and label instances from unlabeled set and transfer them to the labeled set. A training classifier statistic encompasses a label for each such instance and the confidence of the underlying classifier that assigned the label. This information can be used later in order to aggregate the votes of several co-training classifiers. For example, in *RSSalg* multiple co-training classifiers are built. Statistics about building these classifiers is recorded and later used for building the final training set in RSSalg.

A *testing* classifier statistic provides the information about each instance from the test set *labeled during the algorithm evaluation*. As training statistics, this information encompasses the label assigned to the instance, as well as the classifiers confidence for the assigned label. In the case where the label is assigned by the ensemble of classifiers, all confidences are recorded. For example, when applying a trained co-training classifier to the test set, confidences of both underlying single-view classifiers for each category are recorded. This information can be later used in order to aggregate the classifier votes on the test set. For example, in *MV* multiple co-training classifiers are built. Each trained classifier is applied to the test set and statistics about testing these classifiers is recorded. This statistics is later used for labeling test instances by a simple majority vote of the created classifiers.

5. Recording results

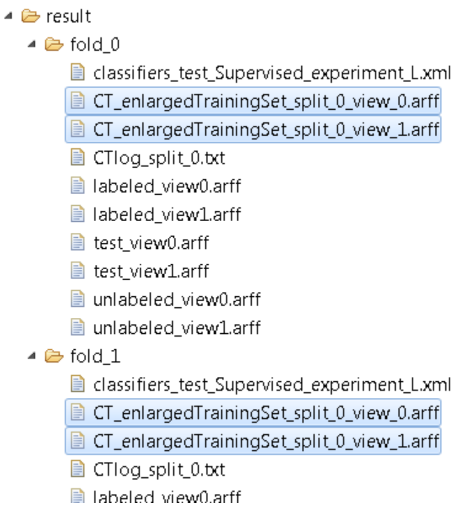
User can choose to record both training and testing classifier statistic for the executed algorithm by selecting the "Write training/test statistics" option. The statistic will be saved in the directory defined by the result folder (property resultFolder, table 1) to the file whose name will be automatically determined by the executed algorithm (please refer to later subchapters for the file names in which the statistics of the algorithms will be recorded). Figure 14 shows an example of written test classifier statistics for the *L* setting.



```
result
  fold_0
    classifiers_test_Supervised_experiment_L.xml
    labeled_view0.arff
    labeled_view1.arff
    test_view0.arff
    test_view1.arff
    unlabeled_view0.arff
    unlabeled_view1.arff
  fold_1
    classifiers_test_Supervised_experiment_L.xml
    labeled_view0.arff
    labeled_view1.arff
```

Figure 14 Example of written test classifier statistics for the *L* setting

Finally, *RSSalg software* provides the user with an option to save the enlarged training set obtained during the algorithm execution, i.e. the training set consisting of both labeled data and unlabeled instances labeled and added to the training set during algorithm execution time. The enlarged dataset will be saved in the directory defined by the result folder (property resultFolder, table 1) to the file whose name will be automatically determined by the executed algorithm (see later chapters for the file names for specific algorithms). Figure 15 shows an example of written enlarged training sets for the Natural setting.



```
result
  fold_0
    classifiers_test_Supervised_experiment_L.xml
    CT_enlargedTrainingSet_split_0_view_0.arff
    CT_enlargedTrainingSet_split_0_view_1.arff
    CTlog_split_0.txt
    labeled_view0.arff
    labeled_view1.arff
    test_view0.arff
    test_view1.arff
    unlabeled_view0.arff
    unlabeled_view1.arff
  fold_1
    classifiers_test_Supervised_experiment_L.xml
    CT_enlargedTrainingSet_split_0_view_0.arff
    CT_enlargedTrainingSet_split_0_view_1.arff
    CTlog_split_0.txt
    labeled_view0.arff
```

Figure 15 Example of written enlarged training sets for the *Natural* setting

6. Saving and loading properties

User can save the specified experiment settings by clicking on the „Save and close" button on the bottom right side of experiment settings input form (figure 12). User is prompted to specify the directory the settings will be saved to in the file automatically named according to the algorithm name.

User can load the existing *.properties* file by clicking on the „Load from file" button on the bottom right side of co-training settings input form (figure 12). User will be prompted to select a *.properties* file containing the desired experiment.

3.4.1 Running the L_{acc} setting

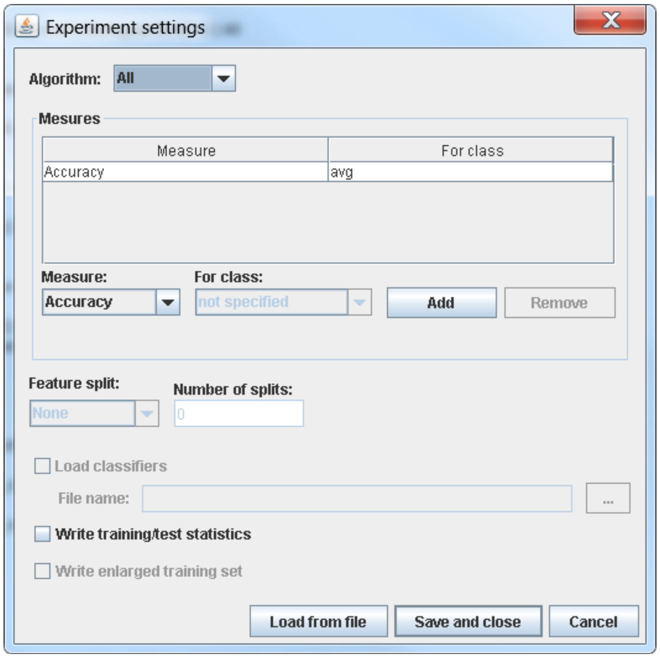
The input form for experiment settings entry with the selection of *L* setting is shown in figure 12. In L_{acc} setting, all features are merged in a unique attribute set and the classifier is trained on the labeled portion of the data (i.e. feature values from all other views are copied to the first view which is used for training). The 'Feature split' choice list and the 'Number of splits' field are disabled and set to 'None' and 0, respectively, as the supervised labeled experiment does not rely on a feature split.

As the L_{acc} setting does not rely on training or testing classifier statistics, 'Load classifiers' choice is disabled. Since none of the unlabeled instances is labeled and added to the training set during the L_{acc} setting execution, 'Write enlarged training set' option is disabled. For the same reason, if the 'Write training/test statistics' option is chosen, only the *test* statistics will be written. The test statistic file is automatically named 'classifiers_test_Supervised_experiment_L.xml', figure 14.

Figure 12 shows the experiment settings input form populated with the values needed in order to run the *L* setting for the News2x2 experiment. When the user choses to save the input properties for the *L* setting, a *.properties* file is automatically named 'experiment_L.properties'.

3.4.2 Running the All_{acc} setting

The input form for experiment settings entry adjusted for performing the All_{acc} setting for News2x2 experiment is shown in figure 16.



Algorithm: All

Measure	For class
Accuracy	avg

Measure: Accuracy

For class: not specified

Feature split: None

Number of splits: 0

□ Load classifiers

File name:

□ Write training/test statistics

□ Write enlarged training set

Figure 16 The experiment settings input form adjusted for running the *All* setting on News2x2 dataset

In *All*$_{acc}$ setting, all features are merged in a unique attribute set. Also, all unlabeled instances are transferred to the labeled data and their correct manually-assigned labels are used. Note that if the ‘Remove labels’ option was chosen when creating a cross-validation experiment (chapter 3.2), *All*$_{acc}$ setting will reduce to the L_{acc} setting as the labels for unlabeled data are unavailable.

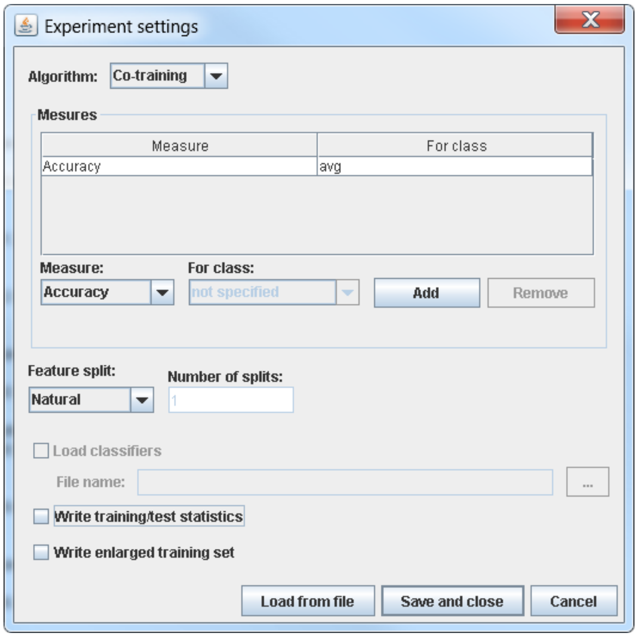
The ‘Feature split’ choice list and the ‘Number of splits’ field are disabled and set to ‘None’ and 0, respectively, as the supervised experiment *All*$_{acc}$ does not rely on a feature split. As the *All*$_{acc}$ setting does not rely on training or testing classifier statistics, ‘Load classifiers’ choice is also disabled.

Because the unlabeled instances are simply copied to the training set during the *All*$_{acc}$ setting execution ‘Write enlarged training set’ option is disabled. For the same reason, if the ‘Write training/test statistics’ option is chosen, only the *test* statistics will be written. The test statistic file is automatically named ‘classifiers_test_Supervised_experiment_All.xml’.

When the user choses to save the input properties for the *All*$_{acc}$ setting, a *.properties* file is automatically named ‘experiment_All.properties’.

3.4.3 Running the Natural setting (co-training with a natural feature split)

The input form for experiment settings entry adjusted for performing the *Natural* setting for News2x2 experiment is shown in figure 17.



Experiment settings

Algorithm: Co-training

Mesures

Measure	For class
Accuracy	avg

Measure: Accuracy

For class: not specified

Feature split: Natural

Number of splits: 1

□ Load classifiers

File name:

□ Write training/test statistics

□ Write enlarged training set

Figure 17 The experiment settings input form adjusted for running the *Natural* setting on News2x2 dataset

For running the *Natural* setting, 'Algorithm' is set to 'Co-training' and 'Feature split' choice is set to 'Natural'. In this case the 'Number of splits' field is disabled and set to '1' as there is only one "natural" split. Running co-training with natural feature split in *RSSalg software* does not actually perform a feature split – it is assumed that the user has previously provided the two views (see chapter 3.1.1) and this separation is considered to be the „natural feature split" of the dataset.

The co-training algorithm does not rely on training or testing classifier statistics generated by other classifiers, thus the 'Load classifiers' choice is disabled.

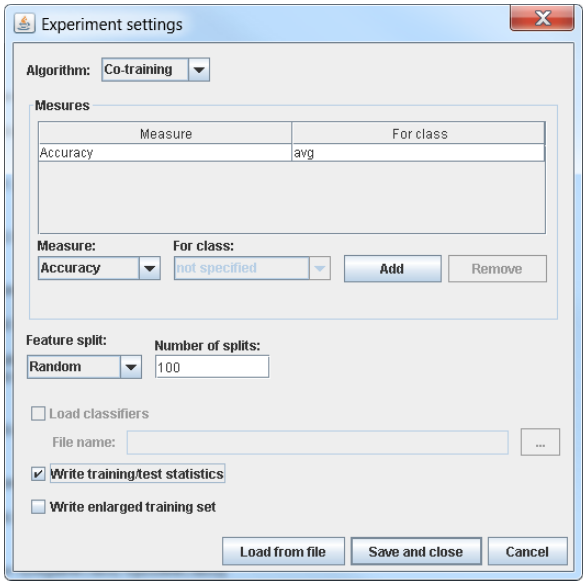
User can choose to save both *training* and *testing* classifier statistic by selecting the 'Write training/test statistic' choice. The resulting files are named 'classifiers_CoTraining.xml' and 'classifiers_test_CoTraining.xml', respectively.

User can also choose to save the enlarged training set obtained with co-training by selecting the 'Write enlarged training set' option. The resulting files will be named 'CT_enlargedTrainingSet_split0_view_0.arff' (the first view) and 'CT_enlargedTrainingSet_split0_view_1.arff' (the second view).

When the user choses to save the input properties for the *Natural* setting, a .*properties* file is automatically named 'experiment_Co-training_Natural.properties'.

3.4.4 Running the Random setting (co-training with a random feature split)

The input form for experiment settings entry adjusted for performing the *Random* setting for News2x2 experiment is shown in figure 18.



Experiment settings

Algorithm: Co-training

Mesures

Measure	For class
Accuracy	avg

Measure: Accuracy

For class: not specified

Feature split: Random

Number of splits: 100

☐ Load classifiers

File name:

☑ Write training/test statistics

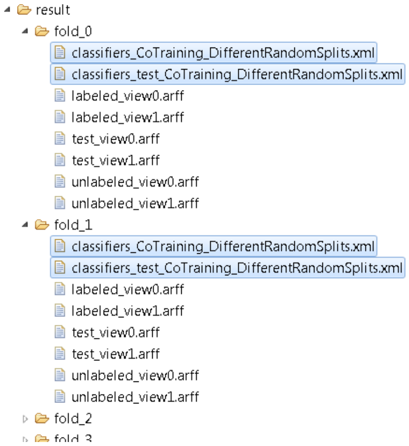
☐ Write enlarged training set

Figure 18 The experiment settings input form adjusted for running the *Random* setting on News2x2 dataset

For running the *Random* setting, ‘Algorithm’ is set to ‘Co-training’ and ‘Feature split’ choice is set to ‘Random’. In this case the ‘Number of splits’ field is enabled and user can enter an arbitrary number of random splits to perform. For News2x2 experiment, the number of splits is set to 100 which will create 100 different random splits (i.e. 100 different co-training classifiers). The resulting performance of *Random* will be the average performance of created classifiers.

The co-training algorithm does not rely on training or testing classifier statistics generated by other classifiers, thus the ‘Load classifiers’ choice is disabled.

User can choose to save both *training* and *testing* classifier statistic by selecting the ‘Write training/test statistic’ choice. The resulting files are named ‘classifiers_CoTraining_DifferentRandomSplits.xml’ and ‘classifiers_test_CoTraining_DifferentRandomSplits.xml’, respectively, and are located in the results folder. For News2x2 experiment the option ‘Write training/test statistics’ is chosen as we would later like to exploit the created *training* statistics for *RSSalg* and *RSSalg*$_{best}$ experiments and created *test* statistics for *MV* experiment. The name and the location of the resulting train and test statistic is shown in figure 19 .



```
result
  fold_0
    classifiers_CoTraining_DifferentRandomSplits.xml
    classifiers_test_CoTraining_DifferentRandomSplits.xml
    labeled_view0.arff
    labeled_view1.arff
    test_view0.arff
    test_view1.arff
    unlabeled_view0.arff
    unlabeled_view1.arff
  fold_1
    classifiers_CoTraining_DifferentRandomSplits.xml
    classifiers_test_CoTraining_DifferentRandomSplits.xml
    labeled_view0.arff
    labeled_view1.arff
    test_view0.arff
    test_view1.arff
    unlabeled_view0.arff
    unlabeled_view1.arff
  fold_2
  fold_3
```

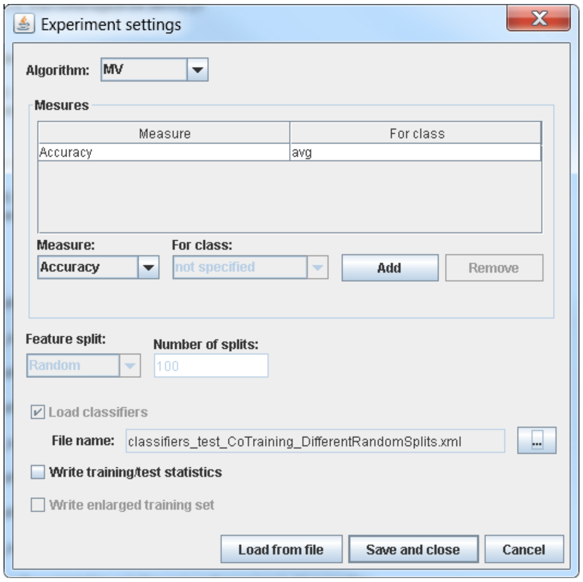
Figure 19 The result of saving the training and test statistics are the files named ‘classifiers_algorithm.xml’ and ‘classififiers_test_algorithm.xml’, respectively, where algorithm denotes the name of the applied algorithm. Figure shows the result of saving the training and test statistic for co-training ran with multiple random splits. These files are saved in the results folder.

User can also choose to save the enlarged training set obtained with co-training by selecting the 'Write enlarged training set' option. The resulting files will be named 'CT_enlargedTrainingSet_split*i*_view_0.arff' (the first view) and 'CT_enlargedTrainingSet_split*i*_view_1.arff' (the second view) where *i* is the number of the performed random split. In News2x2 *Random* experiment *i* ϵ {0,..,99}.

When the user choses to save the input properties for the *Random* setting, a *.properties* file is automatically named 'experiment_Co-training_Random.properties'.

3.4.5 Running the *MV* setting (majority vote of co-training classifiers)

The input form for experiment settings entry adjusted for performing the *MV* setting for News2x2 experiment is shown in figure 20.



Experiment settings

Algorithm: MV

Mesures

Measure	For class
Accuracy	avg

Measure: Accuracy

For class: not specified

Add | Remove

Feature split: Random

Number of splits: 100

☑ Load classifiers

File name: classifiers_test_CoTraining_DifferentRandomSplits.xml ...

☐ Write training/test statistics

☐ Write enlarged training set

Load from file | Save and close | Cancel

Figure 20 The experiment settings input form adjusted for running the *MV* setting on News2x2 dataset

In *MV* setting, an instance from the test set is classified by a simple majority vote of an ensemble of trained co-training classifiers. As the *test* statistic is needed (information on how each classifier from the ensemble classifies instances from the test set), it is necessary to load it. Thus the choice 'Load classifiers' is checked and disabled and by clicking on "…" button user is prompted to select a file containing the test statistic. *RSSalg software* automatically saves the *test* statistic of the resulting classifiers in the results folder in XML files with names starting with 'classifiers_test' (see figure 19). Thus, user is able to select only XML files whose names start with 'classifiers_test' and are located in fold_0 subdirectory in the results folder[10]. In this experiment (figure 20), we assume that the Random setting was already ran and that its test statistic has been recorded in 'classifiers_test_CoTraining_DifferentRandomSplit.xml' file.

When test statistic is selected, *RSSalg software* will automatically read the test statistics from fold_0 and populate the 'Number of splits' field with the number of classifiers from the loaded ensemble.

User can choose to save *testing* classifier statistic by selecting the 'Write training/test statistic' choice[11]. The resulting files are named 'classifiers_test_Majority_vote_of_Co-training_classifiers_on_test_set_DifferentRandomSplits.xml' and are located in the results folder. As MV does not label and add unlabeled instances to the training set, 'Write enlarged training set' option is disabled.

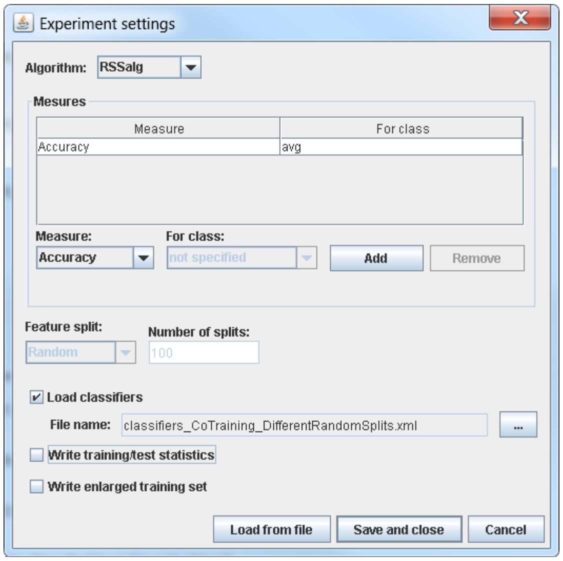
When the user choses to save the input properties for the *MV* setting, a *.properties* file is automatically named 'experiment_MV.properties'.

[10] It is assumed that the appropriate file names are the same in other subdirectories which correspond to other folds in cross-fold-validation. Also, it is assumed that there is a minimum one fold (which corresponds to subdirectory fold_0)

[11] Training statistics is not saved as it would be exactly the same statistic which was loaded for *MV* setting

3.4.6 Running *RSSalg* and *RSSalg*$_{best}$ settings

User can choose to run *RSSalg* or *RSSalg*$_{best}$ setting by using ‘RSSalg’ or ‘RSSalg_best’ as the algorithm choice, respectively. The input form for experiment settings entry adjusted for performing the *RSSalg* setting for News2x2 experiment is shown in figure 21.



Experiment settings

Algorithm: RSSalg

Mesures

Measure	For class
Accuracy	avg

Measure: Accuracy

For class: not specified

Feature split: Random

Number of splits: 100

☑ Load classifiers

File name: classifiers_CoTraining_DifferentRandomSplits.xml

☐ Write training/test statistics

☐ Write enlarged training set

Figure 21 The experiment settings input form adjusted for running the *Rsalg* setting on News2x2 dataset

RSSalg relies on using the train statistic of multiple co-training classifiers. User can choose to load a previously saved statistic (e.g. if user has previously ran the *Random* setting and would like to use the same classifiers for RSSalg) by choosing the ‘Load classifiers’ option. Since *RSSalg software* automatically saves the *train* statistic in the results folder in XML files with names starting with ‘classifiers_’ (see figure 19), the user is able to select only XML files whose names start with ‘classifiers_’ and are located in fold_0 subdirectory in the results folder[12]. When train statistic is selected, *RSSalg software* will automatically read the test statistics from fold_0 and populate the ‘Number of splits’ field with the number of classifiers from the loaded ensemble. In this experiment (figure 21), we assume that the Random setting was already ran and that its train statistic has been recorded in ‘classifiers_test_CoTraining_DifferentRandomSplit.xml’ file. We will use this statistics in order to form the final training set in *RSSalg*.s

If the ‘Load classifiers’ option is not selected, *RSSalg* will create a new training statistics by running co-training with different random splits for the number of times user specified in the ‘Number of splits’ field. Assuming that the random seed is unchanged and that user chose the same number of splits as in *Random* experiment, the newly created statistic will be exactly the same as the same as the same sequence of random numbers is generated.

User can choose to save *train* and *test* classifier statistic by selecting the ‘Write training/test statistic’ choice. The resulting files are named ‘classifiers_RSSalg_left_out_instances_DifferentRandomSplits.xml’ and ‘classifiers_test_RSSalg_left_out_instances_DifferentRandomSplits.xml’, respectively, for *RSSalg* setting and ‘classifiers_RSSalg_best_DifferentRandomSplits.xml’ and ‘classifiers_test_RSSalg_best_DifferentRandomSplits.xml’ for *RSSalg*$_{best}$ setting. The saved files are located in the results folder.

The selection of ‘Write enlarged training set’ option will result in writing the final classifier of *RSSalg* or *RSSalg*$_{best}$ in the files named ‘RSSalg_left_out_instances_enlargedTrainingSet.arff’ and ‘RSSalg_best_enlargedTrainingSet.arff’, respectively.

When the user choses to save the input properties for the *RSSalg* or *RSSalg*$_{best}$ setting, a *.properties* file is automatically named ‘experiment_RSSalg.properties’ and experiment_RSSalg_best.properties, respectively.

[12] It is assumed that the appropriate file names are the same in other subdirectories which correspond to other folds in cross-fold-validation. Also, it is assumed that there is a minimum one fold (which corresponds to subdirectory fold_0)

3.4.7 The resulting experiment property file

Table 4 lists all available experiment properties, their meaning and possible values.

Property name	Description	Value
algorithm	The performed algorithm	String value: a full-package name of algorithm implementation. Possible values: • `algorithms.SupervisedAlgorithm_L` (L_{acc}) • `algorithms.SupervisedAlgorithm_All` *(All_{acc})* • `algorithms.co_training.CoTraining` *(Random, Natural)* • `algorithms.RSSalg.MajorityVote` *(MV)* • `algorithms.RSSalg.RSSalg` *(RSSalg, $RSSalg_{best}$)* • Arbitrary class which extends the `algorithms.Algorithm` class

measures	The list of performance measures that should be calculated	Quoted Strings separated with whitespace characters. Each String should be a full-package name of a performance measure. Possible values:
		• `classificationResult.measures.AccuracyMeasure`
		• `classificationResult.measures.F1Measure`
		`classificationResult.measures.Precision`
		`classificationResult.measures.Recall`
		• Arbitrary implementation of `classificationResult.measures.MeasureIF`

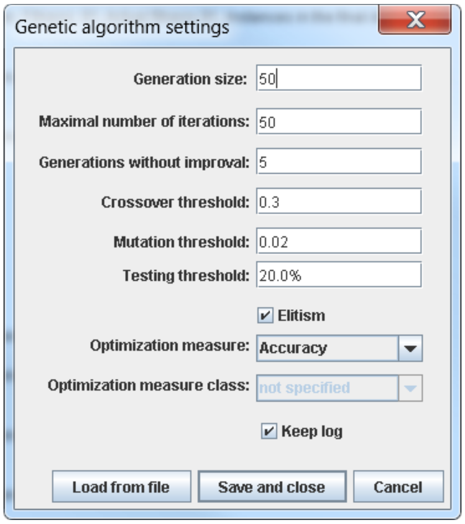
measuresForClass	Categories for category-specific measures such as *f1-measure*	Quoted Strings separated with whitespace characters. Each String denotes the category for which the performance measure is calculated (the *i*th String corresponds to the *i*th String in *measures* property). If the measure is category-independent (e.g. accuracy) or if an average for all categories should be calculated, specify the value as “avg”
featureSpliter	Algorithm for feature splitting	String value: a full-package name of a feature splitting algorithm. Possible values: • `featureSplit.RandomSplit` (random split) • `featureSplit.DifferentRandomSplitsSplitter` (unique random splits) • Arbitrary implementation of `featureSplit.SplitterIF`
noSplits	Number of feature splits	Integer value
loadClassifiers	If true, load a *training* classifier statistic from *ClassifiersFileName*	Boolean value (true/false)
ClassifiersFilename	A classifier statistics to be loaded if *loadClassifier=true.*	String value representing the XML file containg the classifier statistics. Note: For *MV* use *loadClassifier=false* and specify the *test* classifier statistic in this property
writeClassifiers	If true, record *train* and *test* classifier statistics obtained during algorithm execution	Boolean value (true/false)
writeEnlargedTrainingSet	If true, record the enlarged training set obtained by labeling unlabeled instances during algorithm execution	Boolean value (true/false)
voter	Algorithm that aggregates votes assigned by multiple classifiers	String value: a full-package name of the voting algorithm. Possible values: • `algorithms.RSSalg.resultStatistic.Label.MajorityVotes` • Arbitrary implementation of `algorithms.RSSalg.resultStatistic.Label.VoterIF`
candidateEvaluator	Algorithm that evaluates candidates for GA optimization of thresholds in *RSSalg*. This property value is only used for *RSSalg* setting.	String value: a full-package name of the algorithm. Possible values: • `algorithms.RSSalg.GA.RSSalgCandidateEvaluator` (*RSSalg* setting) • `algorithms.RSSalg.GA.TestSetAccuracyCandidateEvaluator` (*$RSSalg_{best}$* setting) • Arbitrary implementation of `algorithms.RSSalg.GA.CandidateEvaluatorIF`

Table 4 The meaning of properties saved in experiment *.properties* file

3.5 Specifying genetic algorithm settings

The genetic algorithm (GA) settings are needed for the execution of *RSSalg* and *RSSalg$_{best}$* experiments as the genetic algorithm is used as the optimization algorithm for threshold selection in these algorithms. Figure 22 shows the dialogue for genetic algorithm settings entry populated with the settings needed for News2x2 experiment: each generation will encompass 50 individuals; GA will be run for maximally 50 iterations; If there is no improvement in 5 generations, GA will stop; The crossover and mutation thresholds are 0.3 and 0.02, respectively; the testing threshold [4] used in *RSSalg* is 20%; Elitism will be used[13]; when evaluating a model resulting from the threshold pair candidate, accuracy is used as the optimization measure; accuracy measure does not depend on a specific category, thus 'Optimization measure for class' is set to 'not specified'.

The user can also select the option of keeping the log about the GA execution. The result log will be written to the file named 'ThresholdOptimiserlog.txt' in the result folder[14].



Generation size: 50

Maximal number of iterations: 50

Generations without improval: 5

Crossover threshold: 0.3

Mutation threshold: 0.02

Testing threshold: 20.0%

☑ Elitism

Optimization measure: Accuracy

Optimization measure class: not specified

☑ Keep log

Figure 22 The GA settings input form with values entered for the News2x2 experiment

User can save the specified GA settings by clicking on the „Save and close" button on the bottom right side of GA settings input form (figure 22). User is prompted to specify the directory the settings will be saved to in the file automatically named 'GA.properties'.

User can load the existing *.properties* file by clicking on the „Load from file" button on the bottom right side of GA settings input form (figure 22). User will be prompted to select a directory which must contain the file named 'GA.properties' from which the properties will be loaded.

[13] The best individual from the generation will be copied to the next generation

[14] The full path to each file is{results_folder}/fold_{currentFold}/ThresholdOptimiserlog.txt

3.5.1 The resulting GA.properties file

Table 5 lists all available GA properties as written in 'GA.propertes' file, their meaning and possible values.

Property name	Description	Value
generationSize	Number of candidates in each GA generation	Integer value
iterations	Maximal number of GA iterations	Integer value
noImprovalGenerations	If the solution does not improve in last *noImprovalGenerations*, stop GA. If set to -1 GA will iterate for the exact number of iterations specified in 'iterations' property	Integer value (-1 or greater than 0)
crossoverTS	Crossover threshold in GA	Double value in the range (0,1)
mutationTS	Mutation threshold in GA	Double value in the range (0,1)
testingTS	Testing threshold in RSSalg	Double value in the range (0,1)
elitism	If set to true use elitism (copy the best individual from current generation to the next generation)	Boolean value (true/false)

optimizationMeasure	Each candidate in *RSSalg* represents a threshold pair which defines the training set used the final model in *RSSalg*. This property defines the evaluation measure for this model.	String value: full-package name of the implementation of a performance measure. Possible values:
		• `classificationResult.measures.AccuracyMeasure`
		• `classificationResult.measures.F1Measure`
		`classificationResult.measures.Precision`
		`classificationResult.measures.Recall`
		• Arbitrary implementation of `classificationResult.measures.MeasureIF`

optimizationMeasureClass	Categories for category-specific measures such as *f1*-measure	String value: the category for which the performance measure is calculated. If the measure is category-independent (e.g. accuracy) or if an average of the measure value for all categories should be calculated, specify the value as "avg"
logGA	If true, log the process of GA optimization	Boolean value (true/false)

Table 5 The meaning of properties saved in 'GA.properties' file

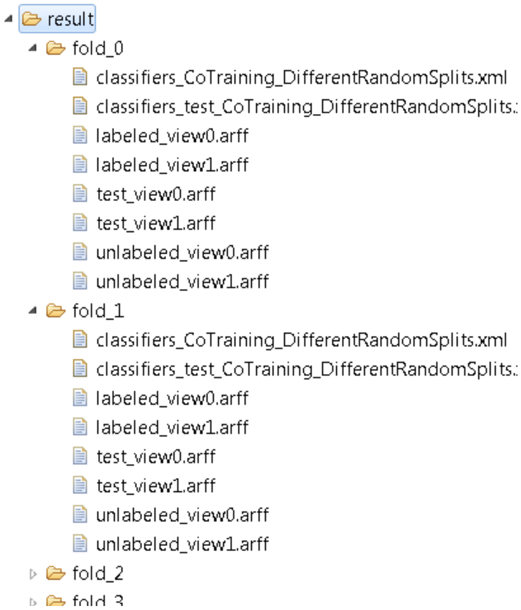
4 Running the experiment in RSSalg software

In order to run the desired experiment the user must firstly specify the desired settings. Dataset settings and co-training settings are obligatory for all experiments. Cross-validation experiment settings are obligatory if the user has chosen to create a new cross-validation experiment (chapter 3.2). The experiment that will be run is determined by the currently selected experiment settings. Genetic algorithm settings are only required in case of running *RSSalg* or *RSSalg*$_{best}$ experiment.

Please refer to chapter 3 for detailed description of entering the desired settings. By clicking on 'Load all' button (figure 1) the user may load all previously created settings: user is prompted to select the directory to re-load the saved properties from. *RSSalg* software will load the following *.properties* files from the selected directory: data settings from the file 'data.properties', the cross-validation settings from the file 'cv.properties', the co-training settings from the file 'co-training.properties', the settings of the *L* experiment from the file 'experiment_L.properties' and the genetic algorithm settings from the file 'GA.properties'. User can clear all selecting settings by clicking on 'Clear settings' button (figure 1).

The experiment execution is started by clicking on the 'Start experiment' button. Experiment results will be shown in the results window, figure 1. If the user wishes to clear the results window, he can do so by clicking on the „Clear" button on the bottom right side of the main window of *RSSalg software* application (figure 1).

In case the user has chosen to create a new cross-validation experiment (chapter 3.1.1), when the experiment starts, the created cross-validation experiment will be written to the specified result folder (chapter 3.1.3) as shown in figure 23. The file structure of the written data is explained in chapter 3.1.2. For the later experiments, user may want to select the option of loading the prepared experiment from the result folder as explained in chapter 3.1.2. Assuming that the user does not change the random generator seed or other data and cross-validation settings, choosing to load a prepared experiment or generating a new one for each experiment does not affect the results. Each time the experiment is created using the same sequence of random numbers, so each time the instances layout across different folds and train/unlabeled/test data is the same. However, it is recommended to create the experiment only once and later load the prepared experiment as the experiment creating procedure may be slow for the larger datasets.



```
result
  fold_0
    classifiers_CoTraining_DifferentRandomSplits.xml
    classifiers_test_CoTraining_DifferentRandomSplits.
    labeled_view0.arff
    labeled_view1.arff
    test_view0.arff
    test_view1.arff
    unlabeled_view0.arff
    unlabeled_view1.arff
  fold_1
    classifiers_CoTraining_DifferentRandomSplits.xml
    classifiers_test_CoTraining_DifferentRandomSplits.
    labeled_view0.arff
    labeled_view1.arff
    test_view0.arff
    test_view1.arff
    unlabeled_view0.arff
    unlabeled_view1.arff
  fold_2
  fold_3
```

Figure 23 The result of creating a new cross-validation experiment

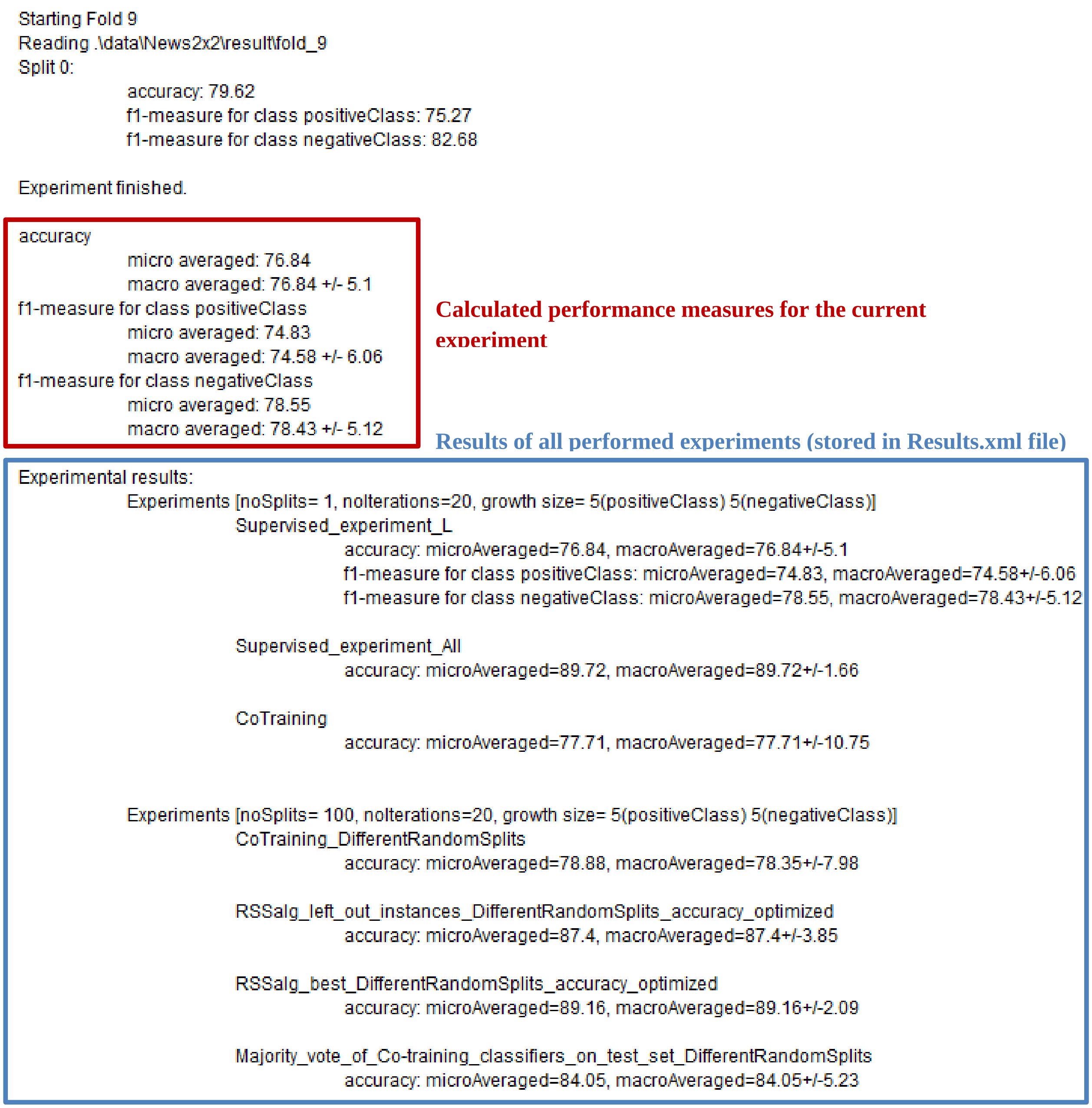
Each time an experiment is ran, the resulting micro and macro averaged measures are shown in the results window. The example output (obtained by executing L_{acc} experiment of News2x2 dataset) is shown in figure 24. Each time an experiment is run, its resulting performance is recorded in 'Results.xml' file located inside the result folder. At the end of performing the experiment, RSSalg software will list all of the results recorded in 'Results.xml' (figure 24).

Figure 25 shows an example of 'Results.xml' file obtained after running several different experiments. The experiments recorded in recorded in the 'Results.xml' are organized according to some basic settings: the number of co-training iterations, the growth size and number of performed splits. The name of the applied algorithm is also saved. Thus, only running the experiment with exactly the same name and recorded properties will rewrite the results for that experiment, otherwise the experiment results will be appended to the list. Note that for each experiment different measures may be calculated – running the experiment with exactly the same settings will only modify the measure which is being calculated.

Table 6 lists the results obtained by running the experiments described in this document on the News2x2 dataset. One of the advantages of *RSSalg software* is the exact reliability of the obtained results. The random generator seed parameter ensures that the experiments are always executed using the same sequence of random numbers. Thus, specifying the same settings as the ones given in this document should always yield the same results.

	L_{acc}	**All_{acc}**	**Natural**	**Random**	**MV**	**RSSalg**	**$RSSalg_{best}$**
News2x2	76.8±5.1	89.7±1.7	77.7±10.8	78.4±8.0	84.1±5.2	84.7±3.8	89.2±2.0

Table 6 Results of presented experiments on News2x2 dataset



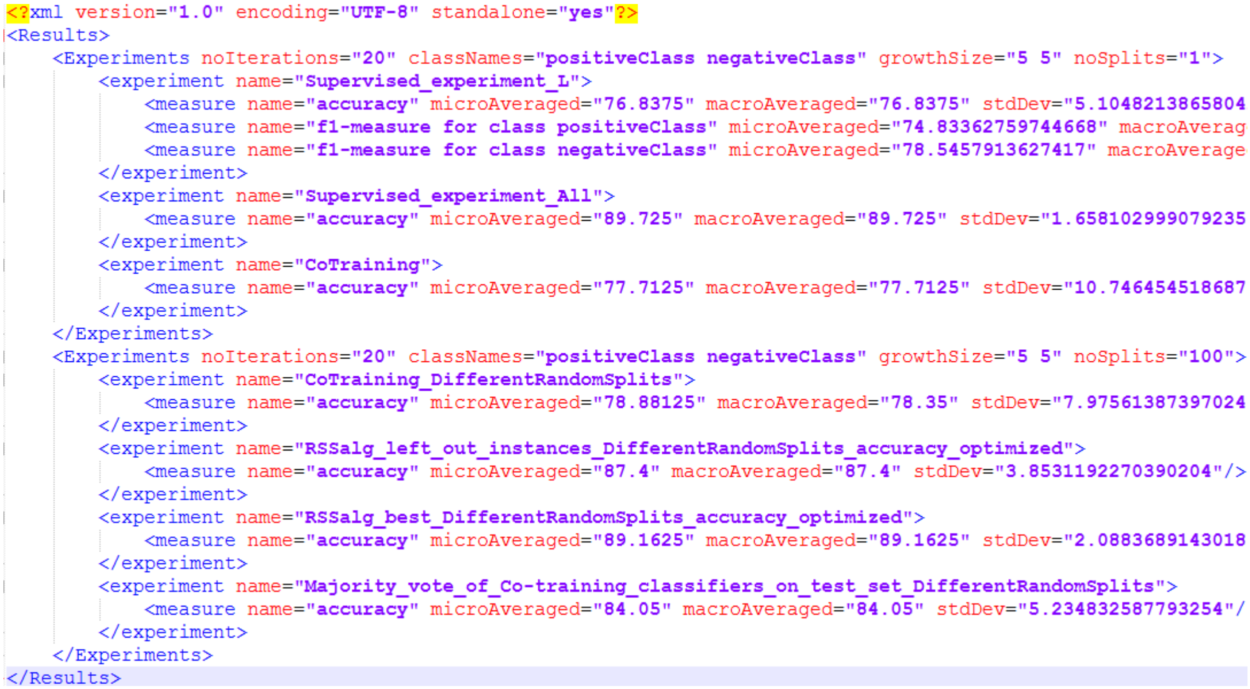
```
Starting Fold 9
Reading .\data\News2x2\result\fold_9
Split 0:
            accuracy: 79.62
            f1-measure for class positiveClass: 75.27
            f1-measure for class negativeClass: 82.68

Experiment finished.

accuracy
            micro averaged: 76.84
            macro averaged: 76.84 +/- 5.1
f1-measure for class positiveClass
            micro averaged: 74.83
            macro averaged: 74.58 +/- 6.06
f1-measure for class negativeClass
            micro averaged: 78.55
            macro averaged: 78.43 +/- 5.12
```

Calculated performance measures for the current experiment

Results of all performed experiments (stored in Results.xml file)

```
Experimental results:
            Experiments [noSplits= 1, noIterations=20, growth size= 5(positiveClass) 5(negativeClass)]
                        Supervised_experiment_L
                                    accuracy: microAveraged=76.84, macroAveraged=76.84+/-5.1
                                    f1-measure for class positiveClass: microAveraged=74.83, macroAveraged=74.58+/-6.06
                                    f1-measure for class negativeClass: microAveraged=78.55, macroAveraged=78.43+/-5.12

                        Supervised_experiment_All
                                    accuracy: microAveraged=89.72, macroAveraged=89.72+/-1.66

                        CoTraining
                                    accuracy: microAveraged=77.71, macroAveraged=77.71+/-10.75


            Experiments [noSplits= 100, noIterations=20, growth size= 5(positiveClass) 5(negativeClass)]
                        CoTraining_DifferentRandomSplits
                                    accuracy: microAveraged=78.88, macroAveraged=78.35+/-7.98

                        RSSalg_left_out_instances_DifferentRandomSplits_accuracy_optimized
                                    accuracy: microAveraged=87.4, macroAveraged=87.4+/-3.85

                        RSSalg_best_DifferentRandomSplits_accuracy_optimized
                                    accuracy: microAveraged=89.16, macroAveraged=89.16+/-2.09

                        Majority_vote_of_Co-training_classifiers_on_test_set_DifferentRandomSplits
                                    accuracy: microAveraged=84.05, macroAveraged=84.05+/-5.23
```

Figure 24 The example output (*MV* experiment on News2x2 dataset)



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Results>
    <Experiments noIterations="20" classNames="positiveClass negativeClass" growthSize="5 5" noSplits="1">
        <experiment name="Supervised_experiment_L">
            <measure name="accuracy" microAveraged="76.8375" macroAveraged="76.8375" stdDev="5.1048213865804
            <measure name="f1-measure for class positiveClass" microAveraged="74.83362759744668" macroAverag
            <measure name="f1-measure for class negativeClass" microAveraged="78.5457913627417" macroAverage
        </experiment>
        <experiment name="Supervised_experiment_All">
            <measure name="accuracy" microAveraged="89.725" macroAveraged="89.725" stdDev="1.658102999079235
        </experiment>
        <experiment name="CoTraining">
            <measure name="accuracy" microAveraged="77.7125" macroAveraged="77.7125" stdDev="10.746454518687
        </experiment>
    </Experiments>
    <Experiments noIterations="20" classNames="positiveClass negativeClass" growthSize="5 5" noSplits="100">
        <experiment name="CoTraining_DifferentRandomSplits">
            <measure name="accuracy" microAveraged="78.88125" macroAveraged="78.35" stdDev="7.97561387397024
        </experiment>
        <experiment name="RSSalg_left_out_instances_DifferentRandomSplits_accuracy_optimized">
            <measure name="accuracy" microAveraged="87.4" macroAveraged="87.4" stdDev="3.8531192270390204"/>
        </experiment>
        <experiment name="RSSalg_best_DifferentRandomSplits_accuracy_optimized">
            <measure name="accuracy" microAveraged="89.1625" macroAveraged="89.1625" stdDev="2.0883689143018
        </experiment>
        <experiment name="Majority_vote_of_Co-training_classifiers_on_test_set_DifferentRandomSplits">
            <measure name="accuracy" microAveraged="84.05" macroAveraged="84.05" stdDev="5.234832587793254"/
        </experiment>
    </Experiments>
</Results>
```

Figure 25 The 'Results.xml' file located in the results folder, obtained after running the experiments on News2x2 dataset described in this document

5 References

[1] Nigam, K. and Ghani, R., 2000. Understanding the behavior of co-training. In Proceedings of KDD-2000 workshop on text mining (pp. 15-17)

[2] Blum, A. and Mitchell, T., 1998, July. Combining labeled and unlabeled data with co-training. In Proceedings of the eleventh annual conference on Computational learning theory (pp. 92-100). ACM.

[3] Feger, F. and Koprinska, I., 2006, July. Co-Training using RBF nets and different feature splits. In Neural Networks, 2006. IJCNN'06. International Joint Conference on (pp. 1878-1885). IEEE.

[4] Slivka, J., Kovačević, A. and Konjović, Z., 2013. Combining Co-Training with Ensemble Learning for Application on Single-View Natural Language Datasets. Acta Polytechnica Hungarica, 10(2). http://www.uni-obuda.hu/journal/Slivka_Kovacevic_Konjovic_40.pdf

[5] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I.H., 2009. The WEKA data mining software: an update. ACM SIGKDD explorations newsletter, 11(1), pp.10-18.