# *RSSalg software* user manual

The purpose of this document is to give an overview of *RSSalg software*. *RSSalg software* is a tool intended for training a classification model in the case where unlabeled data is abundant, but very little annotated training data is available.

In sections 1-4 we give an overview of *RSSalg software* functionalities. Section 1 states the problem. In section 2 we review the classification algorithms implemented in *RSSalg software* and their (user-controllable) parameters. Section 3 explains the built-in cross-validation procedure used for classifier comparison. Finally, section 4 explains all software outputs.

In section 5 we show the empirical experiments which demonstrate *RSSalg software* functionalities. These ready-to-run reproducible experiments can be found in *RSSalg software* code repository[1]. We discuss the obtained results and give guidelines on when to expect the best performance of each of the algorithms implemented in *RSSalg software*.

Section 6 is dedicated to explaining how to install and run *RSSalg software*. This step -by-step tutorial with screenshots will explain how to use *RSSalg software* GUI to create and run new experiments.

Finally, in section 7 we give an overview of *RSSalg software* architecture. This section also includes a detailed explanation how to expand *RSSalg software* code with custom implementations.

## 1 When to use *RSSalg software*

*RSSalg software* is a free and open-source tool for experimenting with semi-supervised learning (SSL) classification techniques. It is registered under GNU General Public License.

When training a prediction model we are often faced with a problem of lacking a sufficient amount of labeled training data. Data labeling may require manual human work which may be very expensive in terms of both time and money. For example, unlabeled clinical notes are abundant; however, labeling them with a goal of training a medical diagnostic expert system requires a group of trained medical experts that must manually inspect and carefully label each sample. The predictive power of the obtained prediction model is largely dependent on the size and quality of the training sample [1].

SSL techniques strive to incorporate both labeled and unlabeled data in the training process, thus minimizing the human effort in terms of hand-labeling data while keeping the desired level of quality of the prediction model [2]. *RSSalg software* can be used to train a classification model in a supervised fashion (using only available labeled data) or in the semi-supervised fashion (using both labeled and unlabeled data), thus enabling the user to assess how much did the unlabeled data augmented the learning process.

A major SSL technique is co-training [3]. Co-training is applicable on the datasets that have a natural separation of features in two views. *RSSalg software* enables the user to run co-training with user-defined feature split.

In practice, an adequate feature split for co-training might be unknown (these datasets are referred to as single-view datasets in the co-training literature). This may be addressed by creating an artificial feature split for co-training. In *RSSalg software* the user can chose to run co-training on a single-view dataset using the randomly generated feature split. Researchers can also easily extend *RSSalg software* by adding their own feature-splitting implementations (section 7.9.2).

Another way to address the lack of adequate feature split is to combine co-training with ensemble methods. One such technique is *Random Split Statistic Algorithm* (RSSalg) [4] which can be applied on single-view datasets). RSSalg method is implemented in *RSSalg software*.

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

In order to compare different algorithms, *RSSalg software* encompasses the fold-cross validation experimental design and a number of common metrics for evaluating algorithm performance, such as: accuracy, precision, recall and *f*1-measure (section 3).

*RSSalg software* is beneficial for the researchers who wish to experiment with their own SSL solutions. Implemented algorithms are highly-configurable in terms of parameter values, allowing the researchers to empirically test the parameter influence on algorithm performance. Experiments are fully reproducible as one of the user-controlled parameter is the random number generator seed. It is easy to extend *RSSalg software* by adding new SSL algorithms, feature splitting techniques, and evaluation measures (section 7.9).

In summary, *RSSalg software* can be used to train and evaluate different classification algorithms built on the dataset that:

- may contain both labeled and unlabeled instances;
- may or may not have a feature split that can be utilized in the learning process.

## 2 Algorithms implemented in *RSSalg software*

This section will first give a high-level overview of two main algorithms implemented in *RSSalg software*: co-training and RSSalg and provide a list all their parameters which are tunable in *RSSalg software* (sections 2.1 and 2.2, respectively). Section 2.3 gives a full list of settings implemented in *RSSalg software*.

### 2.1 Co-training

A high-level overview of co-training is illustrated in figure 1. In co-training two different views are used to train two different classifiers using the same available labeled data (*L*). The trained classifiers are applied on the set of unlabeled data (*U*) with the goal of selecting and labeling the most confident instances. These instances are added to the initial labeled set and both classifiers are retrained in the supervised fashion on the enlarged training set. This process is repeated for a predefined number of iterations.

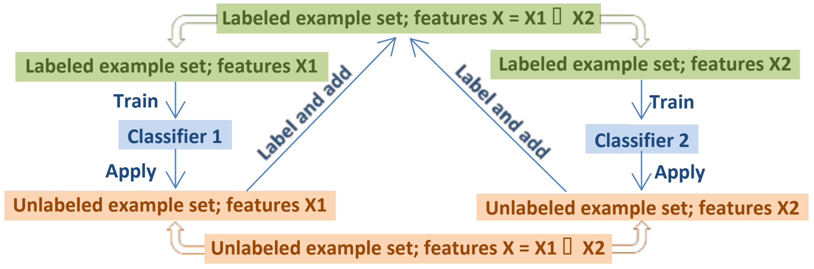

Labeled example set; features X = X1  X2
Labeled example set; features X1
Labeled example set; features X2
Train
Train
Classifier 1
Classifier 2
Apply
Apply
Label and add
Label and add
Unlabeled example set; features X1
Unlabeled example set; features X2
Unlabeled example set; features X = X1  X2

**Figure 1**. A high-level overview of co-training algorithm

#### 2.1.1 Co-training parameters

The performance of co-training, as introduced in [3], depends on the following algorithm parameters which are all tunable in *RSSalg software*:

- **Number of co-training iterations *k*.** In *RSSalg software* user can:
  (1) specify the desired number of iterations *k* after which co-training stops (regardless of how many unlabeled examples are left out of the training process) or
  (2) specify to stop co-training only when all unlabeled instances have been labeled.
- **The growth size (denoted *p*/*n* in [3]).** In *RSSalg software* user can specify $mcc_1$, $mcc_2$, …, $mcc_l$ where $mcc_i$ denotes the number of instances most confidently classified as category *i* which will be added to *L* in each iteration of co-training, and *l* denotes the number of categories. Authors in [3] suggest that, for the best performance of co-training, *p*/*n* should be representative of the underlying label distribution.
- **The usage (and size) of a small unlabeled pool *u'*.** In [3] authors use the small unlabeled pool *u'* for the selection of most confidently labeled instances that will be added to *L* in each iteration of co-training (rather than selecting the most confidently labeled instances directly from *U*). After each iteration, *u'* is replenished by randomly selected examples from *U* to keep its size constant. Authors in [3] have found

that using *u'* makes the underlying co-training classifiers choose the instances more representative of the underlying distribution that generated *U*. In *RSSalg software*:

(1) user can choose to use a small unlabeled pool, in which case he needs to specify the size of *u*', or
(2) he can choose to select examples directly from *U* (as some authors do [5]) by setting the size of *u'* to 0.

- **The classification model used for underlying view classifiers in co-training.** Underlying view classification models may significantly affect the algorithm performance [6]. In *RSSalg software* user can provide an arbitrary classifier from Weka libraries [7] to use for each view separately. User can also provide an implementation of a custom classifier, the only requirement being that it complies with Weka's *Classifier* interface[2]. The effect of using different underlying classification models is explored in section 5.3 of this document.

## 2.2 RSSalg

The first step in RSSalg is to create an ensemble of diverse co-training classifiers. The ensemble is formed by generating a predefined number of random feature splits. Each split is used for running a separate co-training process on the same data. Since each split is different, each run of co-training produces a different enlarged training set, consisting of initially labeled data and data labeled during the co-training process. Examples from all acquired enlarged training sets are integrated in one unique training set. If the same instance was labeled in multiple enlarged training sets, its label is determined by majority vote. The integrated set is then filtered in order to keep only the most confidently labeled instances. Instance is considered confidently labeled if: (1) it appears in most of the produced enlarged training sets and (2) it was assigned the same label in the majority of those sets. Two thresholds are defined for these two conditions: each instance that does not meet the thresholds is omitted from the dataset. After filtering, the integrated training set is used in order to train a supervised classification model which can be used to predict the label for previously unseen examples. The high-level overview of RSSalg is shown in figure 2.

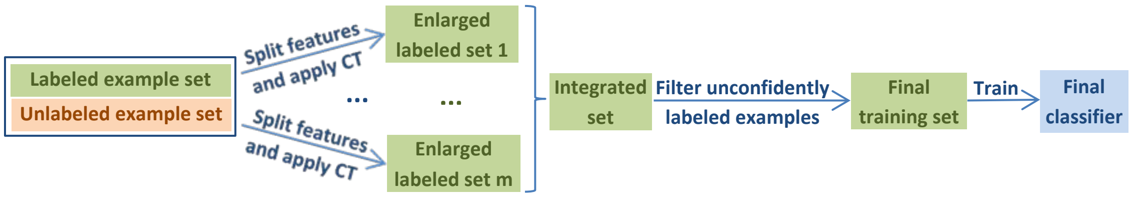


**Figure 2**. A High-level overview of RSSalg

### 2.2.1 Automatic determination of threshold values

For threshold optimization in RSSalg a genetic algorithm (GA) is used: each threshold pair is considered a candidate (an individual in GA generation). A threshold pair defines the instances that will be used for training the final model in RSSalg. Thus, the fitness of the candidate should reflect the quality of the final model. In RSSalg, the quality of the model is assessed using the labeled instances that were left out (filtered) from the final training set and used as candidate fitness.

### 2.2.2 RSSalg parameters

In addition to parameters used in underlying co-training classifiers, RSSalg introduces some additional parameters:

- **Number of random splits *m*** (used for creating *m* different co-training classifiers)**.** In *RSSalg software* user can specify any number of splits (larger than 0). The effect of the number of splits is explored in section 5.2 of this document.
- **Classification model used for training a final classifier in RSSalg.** In *RSSalg software* user can provide an arbitrary classifier from Weka libraries [7]. User can also provide an implementation of a custom classifier, the only requirement being that it complies with Weka's *Classifier* interface[2].
- **Parameters of the genetic algorithm used for automatic determination of threshold values**. In *RSSalg software* user should provide standard GA parameters: number of individuals in one generation

[2] http://weka.sourceforge.net/doc.dev/weka/classifiers/Classifier.html

(i.e. the generation size), maximal number of GA iterations, whether to stop if there was no improvement in a specified number of generations, crossover and mutation thresholds and elitism. There are two additional RSSalg-specific parameters used in GA: the performance measure for the final classifier in RSSalg which is being optimized (accuracy, precision, etc.) and the testing threshold [4] (minimal amount of left out instances needed for evaluating the model).

## 2.3 The list of implemented settings

This section gives a full list of the settings currently implemented in *RSSalg software*:

1. Supervised settings:
   - **L$_{acc}$**: Supervised learner trained on the labeled portion of the data
   - **All$_{acc}$**: Supervised learner trained on both labeled and unlabeled instances with correct label assigned
2. Semi-supervised settings:
   - **Natural:** co-training run with natural (user-defined) feature split. .
   - **Random:** co-training run with random feature split obtained by randomly assigning each feature to one of the two views of roughly equal size. *RSSalg software* reports the average performance of *m* runs of co-training with different random feature splits.
   - **MajorityVote (MV)**: a majority vote of *m* co-training classifiers trained using *m* different random splits (the same classifiers created in **Random** setting)
   - **RSSalg:** RSSalg as introduced in [4]. This is an ensemble technique which uses *m* co-training classifiers obtained using *m* different random splits. As in **MV**, user can choose to use the *m* co-training classifiers created by **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.

It is important to note that the Natural setting is applicable only on the datasets for which the feature split is defined[3]. All other settings are applicable on both single-view and multi-view datasets (the feature split information is not used in the learning process).

## 3 Experimental design implemented in *RSSalg software*

*RSSalg software* encompasses the *k*-fold-cross evaluation procedure designed in [6]: the dataset is divided in *k* stratified folds; in each iteration of the fold-cross validation procedure:

a. a different fold *f* is used for random selection of the initial labeled set (*L*). The size of *L* is $|L|=c_1 + c_2 + \ldots + c_l$, where $c_i$ denotes the number of instances per category *i* and *l* denotes the number of categories;
b. the remaining data from fold *f* as well as *j* ($j<k$) adjacent folds are used as unlabeled data *U* (the class-label information for these instances is disregarded);
c. the remaining *k*-*j*-1 folds are used as test data *T*.

With parameters $k$, $j$, $c_1,\ldots,c_l$ user can control the size and label distribution of the labeled set *L* and the size of the unlabeled set *U*. The fold-cross validation procedure for $k$=10 and $j$=5 is illustrated in figure 3.

[3] For example, WebKB course dataset in [3] is defined to have a feature split: the first view (i.e. feature set) contains the features extracted from the links pointing to the web page, while the second view contains the features extracted from the text of the web page

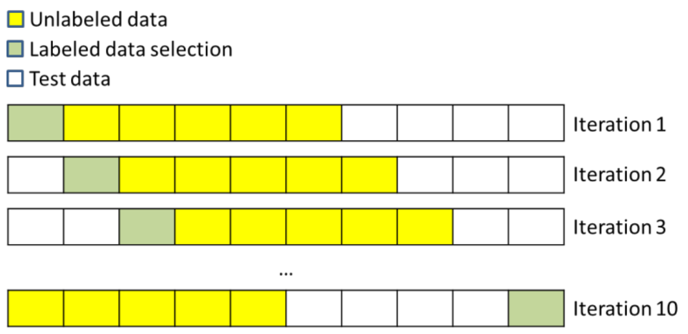


**Figure 3.** Illustration of the k-fold-cross evaluation procedure implemented in *RSSalg software* for k=10 where in each iteration five folds are used as unlabeled data

When running the software user is provided with two options (covered in details in section 6.3.1):

1) Load an existing experiment[4] (previously generated by our tool or a custom one)
2) Let *RSSalg software* generate a new experiment using user-provided $k$, $j$, $c_1,\ldots,c_l$.

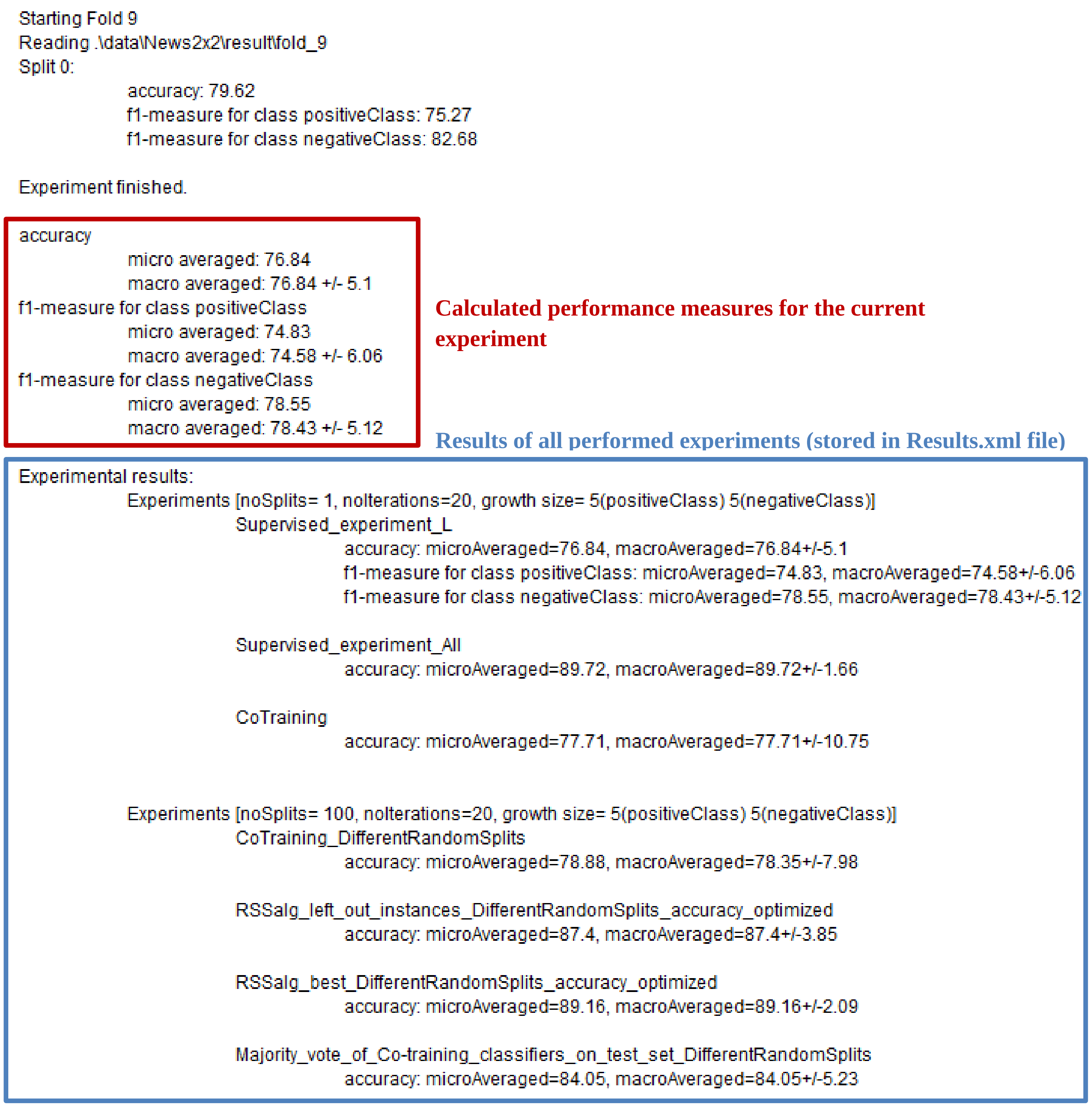
## 4 *RSSalg software* outputs

In *RSSalg software* user can specify a folder to which all outputs will be recorded (section 6.3.1). We will refer to this folder as the 'result folder'. Each time an experiment is ran *RSSalg software* will do the following:

a. **(Optional) Record the generated fold-cross validation experiment**. If the user has chosen to generate a new fold-cross validation experiment, *RSSalg software* will record the resulting data partition to the result folder. Please refer to section 6.3.1.2 for the structure of the recorded cross-validation experiment.
b. **Display micro and macro averaged performance measures**. User may specify one or more performance measures that will be displayed as shown in figure 4.
c. **Record the obtained performances on disk**. The obtained performances will be recorded in 'Results.xml' file located inside the result folder. An example of 'Results.xml' file is given in section 6.4.
d. **Display the results of all recorded experiments**. *RSSalg software* will list all of the results recorded in 'Results.xml' file (figure 4).
e. **(Optional) Record the enlarged training set**. During the execution of each SSL algorithm, some of the instances from the unlabeled set will be labeled and transferred to the labeled set. An enlarged training set may be recorded to the disk as an ARFF file (section 6.3.4 on *Recording results*).
f. **(Optional) Record the "training/test statistics"**. The "training statistics" of a single classifier refers to recording the details of training an SSL classifier: for each unlabeled instance that is labeled and transferred to $L$ during the learning process, the prediction and classifier confidence is recorded. The "testing statistics" of a single classifier refers to recording the details about classifier evaluation: for each instance from the test data, the prediction and classifier confidence is recorded. In *RSSalg software* each algorithm creates an ensemble of classifiers[5]. Thus, the recorded "training/test statistics" holds the training/test details for each classifier from the ensemble (section 6.3.4 on *Recording results*).
g. **(Optional) Log the performance of co-training**. As explained in section 6.3.3, we can log each iteration of co-training (the performance of individual views and the performance of their combination). If co-training is ran multiple times (e.g. in **Random** setting), each co-training process will be logged individually.
h. **(Optional) Log the threshold optimization process in RSSalg**. The threshold optimization properties will be recorded in the 'ThresholdOptimiserlog.txt' file (section 6.3.5).

An example of the output (displayed in console or GUI) is given in figure 4. An example of the result folder structure is given in figure 5.

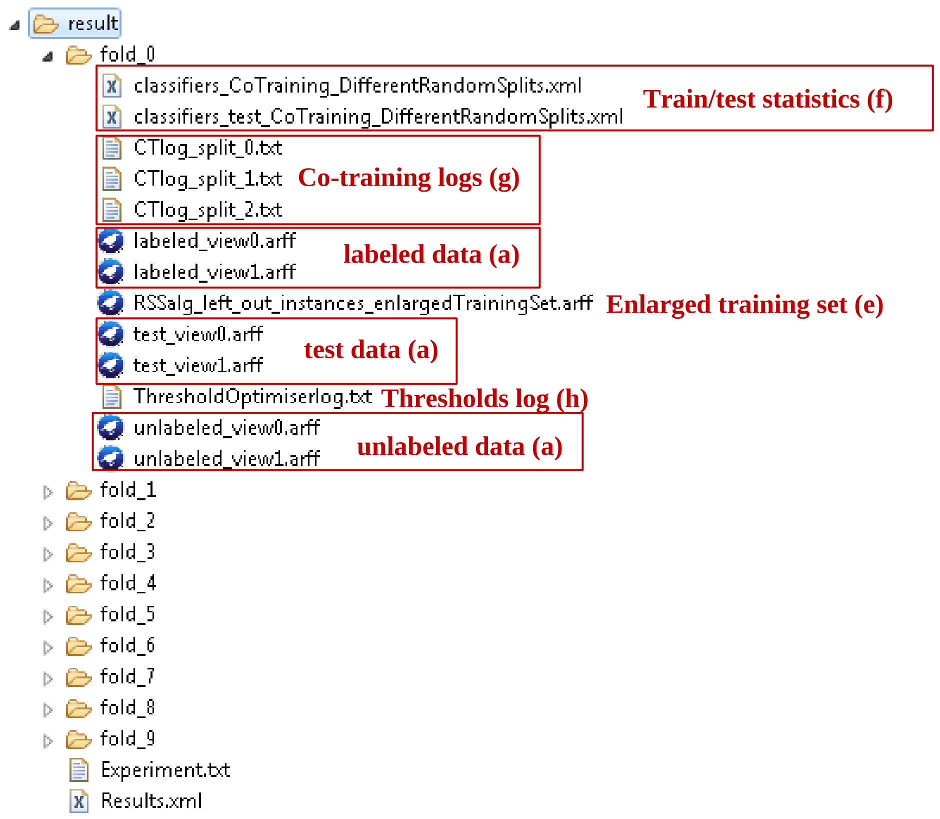
[4] By experiment we mean the dataset divided in folds (the number of folds may be one) where the data in each fold is divided in $L$, $U$ and $T$.
[5] Though an ensemble may contain only one classifier, e.g., in **Lacc** setting



Calculated performance measures for the current experiment
Results of all performed experiments (stored in Results.xml file)

```
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

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 4.** An example of the displayed output (**L**$_{\mathbf{acc}}$ experiment on News2x2 dataset)



```
result
  fold_0
    classifiers_CoTraining_DifferentRandomSplits.xml
    classifiers_test_CoTraining_DifferentRandomSplits.xml
    CTlog_split_0.txt
    CTlog_split_1.txt
    CTlog_split_2.txt
    labeled_view0.arff
    labeled_view1.arff
    RSSalg_left_out_instances_enlargedTrainingSet.arff
    test_view0.arff
    test_view1.arff
    ThresholdOptimiserlog.txt
    unlabeled_view0.arff
    unlabeled_view1.arff
  fold_1
  fold_2
  fold_3
  fold_4
  fold_5
  fold_6
  fold_7
  fold_8
  fold_9
  Experiment.txt
  Results.xml
```

**Figure 5.** An example of the structure of result folder after running several experiments

## 5 Illustrative examples of *RSSalg software* usage

In this section, we will demonstrate how *RSSalg software* can be used for experimenting with semi-supervised classification algorithms. To test an SSL algorithm, we will treat only the portion of a dataset as labeled data and treat the remainder of the dataset as unlabeled data (by disregarding the label information). *RSSalg software* allows for testing the following traits an SSL algorithm should have:

**T1. Is the algorithm able to improve the performance of the supervised baseline by using (only) unlabeled data?**

We will evaluate the performance of a supervised learner trained only on the labeled portion of the data *L* by running **L**$_{\mathbf{acc}}$ setting implemented in *RSSalg software*. The goal of an SSL algorithm is to surpass this performance by incorporating (only) unlabeled data in the process of learning.

**T2. Does the algorithm reduce the number of labeled instances needed for learning while maintaining model quality?**

We will evaluate the performance of a supervised learner trained on both labeled examples and unlabeled examples with the correct label assigned (i.e. while training this supervised learner we do not disregard the label of the instances chosen to be unlabeled) by running **All**$_{\mathbf{acc}}$ setting. We can look at this performance as our goal performance, i.e. we would like to minimize the number of labeled instances needed for learning while still holding the performance of the classifier on this level.

**T3. When is the good performance of the algorithm guaranteed?**

SSL techniques usually rely on imposing certain assumptions about the data and can yield poor performance if those assumptions are violated [2]. For example, co-training imposes assumptions about the properties of the used feature split [3] and violating these assumptions can cause it to degrade the performance of the initial learner instead of improving it [8]. Furthermore, in an SSL setting we lack the sufficient amount of labeled test data to evaluate algorithms. Thus, it is important to characterize which traits should a dataset have so we can guarantee the good performance of our algorithm.

To test the robustness of the methods implemented in *RSSalg software*, we perform the experiments on multiple datasets of various properties that may affect the performance of different algorithms: size, dimensionality, and redundancy. Here we will explain how these dataset properties can affect the algorithms implemented in *RSSalg software*.

The size of the dataset might influence the performance of the algorithms. In order to comply with the original co-training settings where only a few labeled training examples are used [3], in our experiments we use only a few examples as our initial labeled data, regardless of the size of the dataset. Thus, the larger the dataset is, we dispose of more unlabeled data which might augment the training process.

The other factor that might influence the algorithm performance is the dimensionality of the dataset. For low-dimensional datasets we only have a low number of different roughly equal random splits of features. Furthermore, in such case both views will have only a small number of features which might affect their discriminative power, thus violating the sufficiency condition in co-training[6] [3]. However, co-training suffers little from the view insufficiency when the two views have a large diversity [9].

Based on the results presented in [5], we expect that the performance of the settings based on co-training ran with random split (**Random**, **MV** and **RSSalg**) will be best on datasets characterized with high feature redundancy.

**T4. Is there a principal way of choosing optimal parameter values for the algorithm?**

Chosen parameter values might greatly affect algorithm performance. There is a lot of research that shows that co-training greatly depends on the chosen parameter values [5][8][10] and the lack of principal way of selecting co-training parameters was noted as one of its greater disadvantages [10]. This is certainly an interesting open research questions in co-training and, with that in mind, *RSSalg software* was designed to be maximally configurable in terms of user-supplied parameters. Exploring the influence of all parameter combinations for co-training is an exhaustive task and out of the scope of this manual. However, in the experiments performed in this section we will show that *RSSalg software* can be used to empirically address this question. In section 5.2 we test the influence of the number of random splits (*m*) on **MV**, **RSSalg** and **RSSalg$_{best}$**. In section 5.3 we test the influence of different classification models used for individual views in co-training on the performance of algorithms implemented in *RSSalg software*.

For an ensemble technique such as RSSalg it is important to test the following traits:

**T5. Does an ensemble perform better than its individual classifiers on average?**

In *RSSalg software* we are able to run **MV** and **RSSalg** on the same *m* classifiers generated in **Random** setting. We expect these settings to perform better than **Random** which represents the average performance of the individual classifiers in the ensemble.

**T6. Is aggregating the knowledge of individual classifiers better than simple majority vote?**

To test this we will apply **RSSalg** and **MV** on the same set of classifiers. We hypothesize that **RSSalg** will perform better than **MV**.

**T7. Is the threshold procedure used in RSSalg optimal?**

The vote aggregation procedure in RSSalg relies on thresholds which are automatically tuned using unlabeled data. We would like to test whether the obtained thresholds are truly optimal, i.e. if we would get the same values if we had actual labels to optimize on. If the threshold procedure is optimal, performance of **RSSalg** should be close to performance of **RSSalg$_{best}$**.

In section 5.1 we test whether the algorithms implemented in *RSSalg software* possess all listed traits except for T4. Trait T4 is partially explored in section 5.2 where we test the sensitivity of the algorithms to the

[6] For example, Diabetes dataset (section 5.1.1) has only 8 features, thus there are 35 combinations of delimiting the features in two groups of 4. Thus, the maximal number of classifiers used in RSSalg is 35, and in each case underlying co-training classifiers use only 4 features which might make them very weak.

number of performed random splits *m* and in section 5.3 where we test the sensitivity of the algorithms to the classification models used for individual views in co-training.

## 5.1 Comparing the performance of different semi-supervised algorithms

In this section we test whether the SSL algorithms implemented in *RSSalg software* possess all of the listed traits (except T4 which is explored independently in the next two sections).

### 5.1.1 Choosing the datasets

In order to test T3, we perform the experiments on multiple publicly available datasets of various size, dimensionality and feature redundancy: three natural language datasets (WebKB-Course [3], LingSpam [11] and News2x2 [5]) and 12 UCI datasets. The datasets and their basic properties are listed in table 1. First four rows of table 1 are grouped as they belong to the group of natural language datasets which are characterized by the high level of feature redundancy [12]. On the other hand, the rest of the UCI datasets are manually constructed by carefully selecting features, thus we expect little redundancy in these feature sets.

The natural language datasets are preprocessed as proposed in [6]: an English stop-word filter is applied in order to remove 319 frequent words; stemming is performed using Porter's stemmer [13]; for each of the views separately, 200 most frequent words are used. After preprocessing, the text is represented using the bag of words model in combination with *tfidf* measurement [14]. The used text preprocessing and text representation techniques do not depend on the class labels and therefore do not violate the co-training setting (in which the class labels are available for only the few labeled instances). The described preprocessed versions of WebKB-Course, LingSpam and News2x2 datasets are available for download on the *RSSalg software* code repository[7].

**Table 1**. A summary of the datasets used in experiments. Notation: Dataset: datasets marked with * can be downloaded from *RSSalg software* repository **https://github.com/slivkaje/RSSalg-software/tree/master/dist/data**. Datasets marked with ł are part of the Weka distribution of 37 classification problems originally obtained from the UCI repository **http://www.cs.waikato.ac.nz/ml/weka/datasets.html**; Datasets marked with × can be downloaded from UCI machine learning repository **http://archive.ics.uci.edu/ml/**; Dim: dimensionality (number of features); Pos class: the class considered to be "positive" in the experiment[8]; |L|, |All|: size of the labeled training set *L* and the total number of instances used in training (both labeled and unlabeled instances), respectively. The number of instances is given as number of positive instances/number of negative instances; Baseline: accuracy achieved by classifying all instances to the majority class; p/n: the dataset specific growth size parameter used in co-training: the number of instances most confidently labeled as positive and negative, respectively, that will be added to labeled set *L* in each iteration of co-training

| **Dataset** | **Dim** | **Pos class** | **\|L\|** | **\|All\|** | **Baseline** | **p/n** |
|---|---|---|---|---|---|---|
| WebKB ***** | 400 | non-course | 5/5 | 492/138 | 78.1 | 1/3 |
| LingSpam ***** | 400 | spam | 5/5 | 288/1447 | 83.4 | 2/10 |
| News2x2 ***** | 400 | positiveClass | 5/5 | 600/600 | 50.0 | 5/5 |
| Spambase × | 57 | 0 | 2/1 | 1672/1087 | 60.6 | 1/1 |
| Hepatitis **ł** | 19 | LIVE | 1/1 | 73/19 | 79.4 | 11/3 |
| Kr-vs-kp **ł** | 36 | won | 6/5 | 1001/916 | 52.2 | 1/1 |
| Credit-g **ł** | 20 | good | 1/1 | 420/180 | 70.0 | 5/2 |
| Heart-statlog **ł** | 13 | present | 2/3 | 72/90 | 55.6 | 5/6 |
| Sonar **ł** | 60 | Rock | 1/1 | 58/66 | 53.4 | 6/7 |
| Ionosphere **ł** | 34 | g | 5/3 | 135/75 | 64.1 | 2/1 |
| Breast-cancer **ł** | 9 | no-recurrence-events | 1/1 | 120/51 | 70.3 | 5/2 |
| Credit-a **ł** | 15 | + | 4/5 | 184/229 | 55.5 | 1/1 |
| Breast-w **ł** | 9 | malignant | 1/2 | 144/274 | 65.5 | 1/2 |
| Mushroom **ł** | 22 | e | 3/3 | 2524/2349 | 51.8 | 1/1 |
| Diabetes **ł** | 8 | tested_positive | 1/2 | 160/300 | 65.1 | 1/2 |

### 5.1.2 Evaluation procedure

In order to compare the performance of the algorithms, we use the *k*-fold-cross validation procedure described in section 3. Here, we divide the data in 10 folds (*k*=10), use five folds as unlabeled data (*j*=5) and the remaining four folds as test data. Authors in [6] also use *k*=10 and *j=5* as illustrated in figure 3. If we examine the co-training algorithm we can see that it uses only a small number of labelled and unlabeled instances for training. If we would apply a standard 10-fold-cross validation procedure where 90% of the data is used for training and 10% for testing, many instances would be left unused. Using *k*=10 and *j*=5 enables us better utilization of the available

[7] The preprocessed versions of the WebKB-Course, News2x2 and LingSpam datasets are available at https://github.com/slivkaje/RSSalg-software/tree/master/dist/data

[8] In *RSSalg software*, if all class probabilities for the instance are equal, tie is broken by assigning the example to the class specified as "positive"

data: each time 40% of the data is used for testing and the remaining 60% for training. In *RSSalg software*, user is able to set *j* and *k* parameters through the file named 'cv.properties', table 2.

When choosing the size and the distribution of the initial labeled set *L* we are governed by two criteria:

(1) the distribution of labels in *L* is roughly the same as the distribution of labels in the original dataset as in [3] (with the exception of LingSpam and WebKB datasets where we kept the settings recommended in [6]).

(2) When choosing the size of *L*, we derive the initial accuracy by applying $\mathbf{L_{acc}}$ setting and the goal accuracy by applying $\mathbf{All_{acc}}$ setting. *L* is chosen so that the difference $\mathbf{All_{acc}}$ - $\mathbf{L_{acc}}$ is large enough in order to give space for possible improvement by SSL techniques (10-20 %).

The exact size and the distribution of the initial set *L* for each dataset is listed in table 1. User can define the initial set *L* through className and noLabeled properties in 'cv.properties'. For example, in order to form an initial labeled set *L* from Spambase dataset consisting of two randomly chosen instances belonging to class 0 and one randomly chosen instance belonging to the class 1, the following property values should be set: className=”0” “1” and noLabeled=2 1.

As the measure of performance in this experiment we will use accuracy (table 2, common experiment properties).

Since there is randomness involved in the creation of labeled/unlabeled/test splits, and co-training and RSSalg algorithms as well, for the exact replicability of these experiments it is important to maintain the same sequence of random numbers. In *RSSalg software* this is done by supplying the random number generator seed. In this experiment we use 2016 as the seed (table 2, 'data.properties').

### 5.1.3 Choosing algorithm parameters

For better mutual comparison, all algorithms use the same underlying co-training parameters and the same number of random splits for multiple-split settings. In our experiments we use the following underlying co-training parameters (table 2, 'co-training.properties'):

- the number of co-training iterations *k* is 20 as in [6];
- a small unlabeled pool *u*' of size 50 is used for the selection of most confidently labeled examples in each iteration as in [6];
- the number of instances most confidently labeled as 'positive' (*p*) and the number of instances most confidently labeled as 'negative' (*n*) that are added to the initial labeled set *L* in each co-training iteration for each dataset are listed in table 1. The numbers *p* and *n* are chosen to reflect the original dataset distribution as recommended in [3];
- the classification model used for underlying view classifiers in co-training and for the final model in RSSalg is Naïve Bayes (table 2, 'data.properties'). This is the most common choice of an underlying classifier used in co-training studied. In section 5.3 we will perform the experiments using other classifiers.

The number of different random splits (*m*) used in **Random**, **MV**, **RSSalg** and $\mathbf{RSSalg_{best}}$ settings is 100 with the exception of Diabetes dataset where the maximal number of balanced random feature splits is 35 (table 2, common experiment properties).

In **RSSalg** and $\mathbf{RSSalg_{best}}$ settings, a genetic algorithm (GA) is used for automatic threshold optimization. We have used the following GA settings (table 2, 'GA.properties'): the generation size was 50; the maximal number of iterations was 50, but the process was stopped if there was no improvement in 5 successive generations; the crossover threshold was 0.3; the probability of bit mutation was 0.02; elitism was used; the testing threshold in RSSalg was set to 20%; All these parameters were empirically chosen.

### 5.1.4 Summary of the parameter values used in the experiment

Table 2 summarizes the parameter values common for all experiments performed in this section. All experiments presented here are completely reproducible by using the same parameter values. These parameters are supplied to *RSSalg software* through several property files. Pre-created property files needed for running the experiments presented here are available on *RSSalg software* code repository[9]. Detailed instructions on creating these property files are listed in section 6.

**Table 2.** A summary of the property values common for all presented experiments

| **Property file** | **Property name in *RSSalg software*** | **Used value** |
|---|---|---|
| data.properties | randomGeneratorSeed | 2016 |
| | classifiers | weka.classifiers.bayes.NaiveBayes |
| | combinedClassifier | weka.classifiers.bayes.NaiveBayes |
| cv.properties | noFolds | 10 |
| | noUnlabeled | 6 |
| | noTest | 4 |
| co-training.properties | labelAllUnlabeledData | false |
| | coTrainingIterations | 20 |
| | poolSize | 50 |
| GA.properties | generationSize | 50 |
| | Iterations | 50 |
| | noImprovalGenerations | 5 |
| | crossoverTS | 0.3 |
| | mutationTS | 0.02 |
| | elitism | true |
| | testingTS | 0.2 |
| | optimizationMeasure | classificationResult.measures.AccuracyMeasure |
| | optimizationMeasureClass | avg |
| Commom experiment properties | measures | classificationResult.measures.AccuracyMeasure |
| | measuresForClass | avg |
| | noSplits (This property is used in all settings that require multiple random splits for co-training: **Random**, **MV**, **RSSalg** and **RSSalg_best**) | 100 (with the exception of Diabetes dataset where the maximal number of different balanced random splits of 35 is used) |

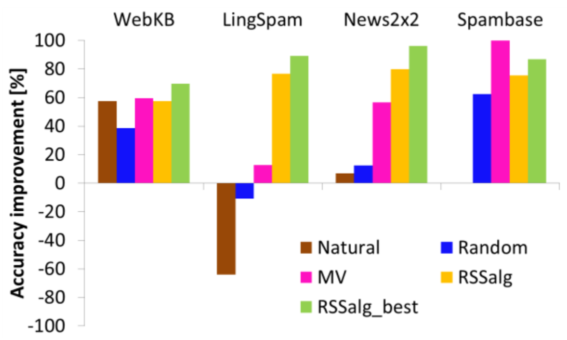
### 5.1.5 Results

Table 3 presents the accuracy and standard deviation obtained by each tested method on all considered datasets. For easier analysis the results are graphically summarized in figure 6. Note that the experiments presented here are the same set of experiments used in [4] to demonstrate the advantages of RSSalg over considered alternatives. However, the results presented here slightly differ from those presented in [4] (in terms of reported accuracy) due to latter software refactoring which affected the sequence of numbers generated by Java random number generator and the usage of different random number generator seed.

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

**Table 3.** Algorithm performance (accuracy ± standard deviation) of all considered algorithms. The performance of *Natural* setting is reported only for WebKB, LingSpam and News2x2 datasets where it was applicable

| **Datasets** | **L$_{acc}$** | **All$_{acc}$** | **Natural** | **Random** | **MV** | **RSSalg** | **RSSalg best** |
|---|---|---|---|---|---|---|---|
| WebKB | 76.1±8.2 | 96.3±1.1 | 87.7±4.7 | 83.9±5.6 | 88.1±5.1 | 87.7±3.7 | 90.2±4.0 |
| LingSpam | 78.2±9.4 | 89.3±3.1 | 71.1±14.7 | 77.0±9.1 | 79.6±7.2 | 86.7±3.5 | 88.1±3.8 |
| News2x2 | 76.8±5.1 | 89.7±1.7 | 77.7±10.8 | 78.4±8.0 | 84.1±5.2 | 87.1±4.0 | 89.2±2.0 |
| Spambase | 67.4±3.8 | 79.6±1.2 | | 75.0±4.4 | 79.6±6.0 | 76.6±7.1 | 78.0±5.7 |
| Hepatitis | 59.6±17.9 | 84.2±2.8 | | 80.6±8.3 | 83.0±4.5 | 82.6±2.8 | 85.7±3.6 |
| Kr-vs-kp | 65.0±4.8 | 87.7±0.6 | | 53.2±3.2 | 55.7±4.1 | 56.6±3.6 | 65.4±5.2 |
| Credit-g | 55.8±10.0 | 73.2±2.8 | | 60.3±5.4 | 65.8±4.8 | 64.4±5.0 | 70.2±1.7 |
| Heart-statlog | 66.2±7.5 | 85.0±2.0 | | 80.0±4.4 | 82.1±4.5 | 80.4±4.2 | 83.1±4.4 |
| Sonar | 54.9±6.3 | 67.9±4.7 | | 51.1±5.2 | 54.9±4.6 | 54.7±4.2 | 58.4±3.2 |
| Ionosphere | 70.2±6.8 | 83.3±5.3 | | 73.9±4.7 | 71.8±5.7 | 72.0±7.3 | 77.8±4.1 |
| Breast-cancer | 56.4±11.6 | 74.1±1.9 | | 68.0±5.8 | 69.7±3.9 | 68.8±4.4 | 72.2±3.3 |
| Credit-a | 68.0±3.6 | 78.1±2.5 | | 71.3±14.2 | 78.7±3.9 | 74.6±4.5 | 79.3±3.5 |
| Breast-w | 80.5±11.8 | 96.0±1.2 | | 93.8±3.1 | 95.1±1.6 | 94.3±2.6 | 96.1±1.6 |
| Mushroom | 80.5±10.0 | 95.4±0.4 | | 82.6±12.2 | 89.0±0.8 | 88.3±1.4 | 89.2±1.5 |
| Diabetes | 64.6±5.4 | 75.2±1.4 | | 54.1±9.4 | 64.7±4.9 | 64.5±4.8 | 68.2±2.2 |



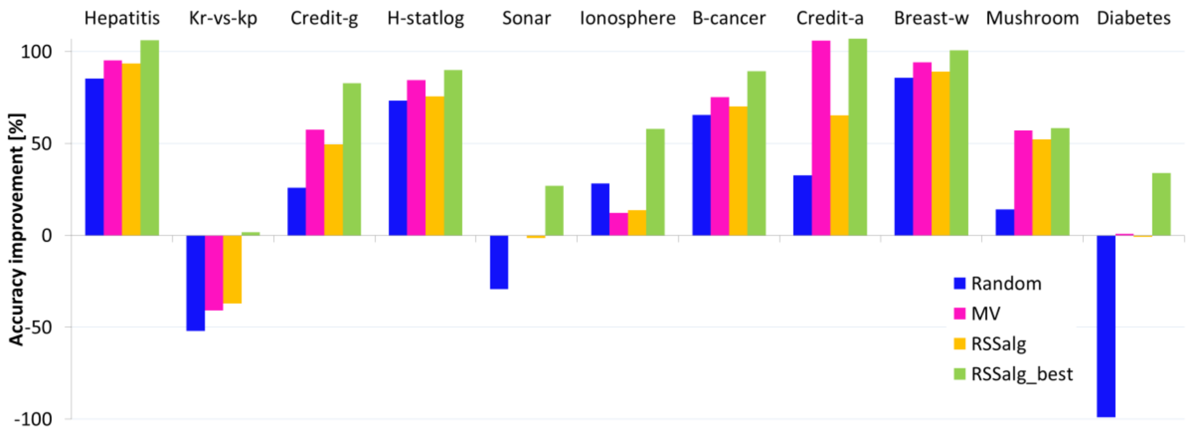


**Figure 6.** Percentage improvement of accuracy calculated as (**Alg$_{acc}$**– **L$_{acc}$**)/(**All$_{acc}$** – **L$_{acc}$**)·100% where **Alg$_{acc}$** denotes the algorithm accuracy. Negative values show degradation of accuracy compared to initial classifier **L$_{acc}$**. **Natural** was only applicable on WebKB, LingSpam and News2x2 datasets which have a defined feature split. **Top:** natural language datasets characterized by high level of feature redundancy; **Bottom:** UCI datasets.

The results presented in table 3 and figure 6 indicate that:

- A dataset trait that influences the performance of **Random**, **MV**, **RSSalg** and **RSSalg$_{best}$** is the redundancy of its features (T3). **Random**, **MV**, **RSSalg** and **RSSalg$_{best}$** perform better on more redundant datasets. This is expected, as all of these settings are based on co-training ran with random feature split which is shown to be beneficial in case of high feature redundancy [5].
- **RSSalg$_{best}$** is the best performing setting. It always shows an improvement of accuracy compared to **L$_{acc}$** (T1 is satisfied). On more redundant datasets the perfomance of **RSSalg$_{best}$** is close to the goal performance **All$_{acc}$** (T2 is satisfied on more redundant datasets).
- On more redundant datasets **MV** and **RSSalg** always perform better than **L$_{acc}$** – we can say that they satisfy T1 on more redundant datasets. However, they are less reliable on low redundancy dataset were there are cases when they even degrade the initial accuracy.
- **RSSalg** and **MV** generally perform better than **Random** setting (the only exception is the Ionosphere dataset). This is expected since **MV** and **RSSalg** belong to the group of ensemble techniques that tend to reduce the variance compared to single classifiers. We can say that **RSSalg** and **MV** generally satisfy T5.
- We observe that in some cases Random performs better than Natural. The same effect was observed in [15] where authors contribute this result to the big difference in quality between the individual views in co-training (in terms of accuracy).
- In most cases **RSSalg** did not beat **MV** (T6). Since **RSSalg$_{best}$** generally performs better than **MV** we can conclude that the threshold optimization procedure needs to be improved.
- On more redundant datasets **RSSalg** is close to **RSSalg$_{best}$** (T7 is satisfied), however this is not always the case on the less redundant datasets. Improving the threshold optimization procedure would make RSSalg more generally applicable (thus satisfying T3).

## 5.2 Testing the influence of the number of random splits used in RSSalg

In this section we will analyze how does the number of random splits (*m*) influence the performance of multi-split settings: **Random**, **MV**, **RSSalg** and **RSSalg$_{best}$** (trait T4). We perform this experiment by keeping the values of all parameters same as in section 5.1 and only varying *m* from 5 to 200 in steps of 5. We record the accuracy achieved by each algorithm for each tested value of *m*. The obtained results are graphically summarized in figures 7 - 10[10]. The results indicate that **MV**, **RSSalg** and **RSSalg$_{best}$** are relatively robust to the value of *m* even though performance of **Random** varies significantly. The performance of **MV**, **RSSalg** and **RSSalg$_{best}$** is expectedly lower for lower values of *m* and grows as *m* grows, however, after a certain point adding more co-training classifiers makes little difference. Another result we were able to observe from this experiment is that the performances of **MV**, **RSSalg** and **RSSalg$_{best}$** are correlated to performance of **Random**.

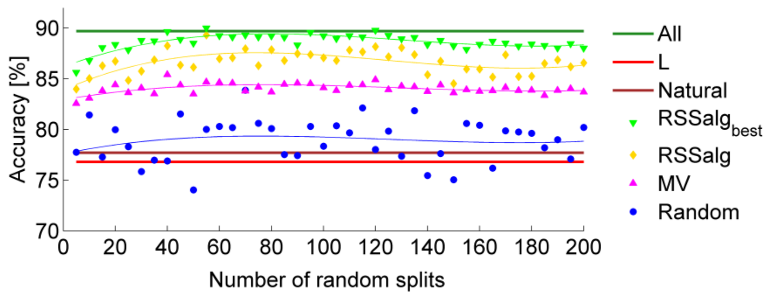


**Figure 7.** Exploring the influence of the number of performed random splits on **Random**, **MV** and **RSSalg** on News2x2 dataset

[10] Due to the magnitude of the task we have selected a representative subset of the datasets presented in section 4.1.1

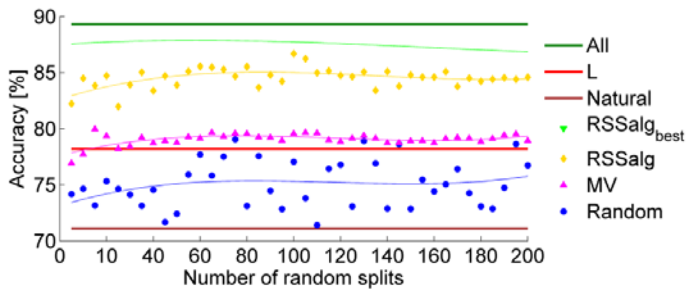


**Figure 8.** Exploring the influence of the number of performed random splits on **Random**, **MV** and **RSSalg** on LingSpam dataset

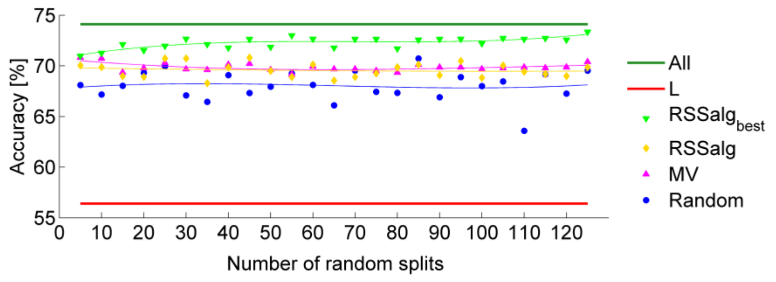


**Figure 9.** Exploring the influence of the number of performed random splits on **Random**, **MV** and **RSSalg** on Breast-cancer dataset. Here the maximal number of splits was 125 due to small dimensionality (only 9 features)

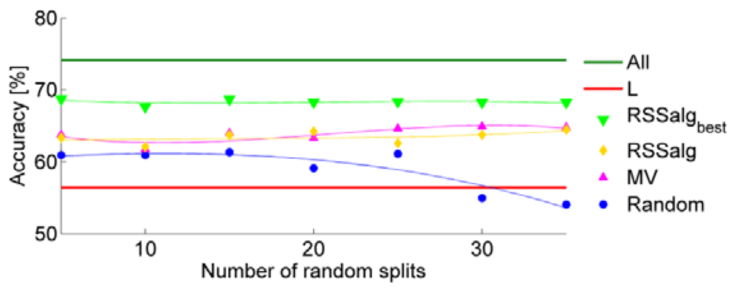


**Figure 10.** Exploring the influence of the number of performed random splits on **Random**, **MV** and **RSSalg** on Diabetes dataset. Here the maximal number of splits was 35 due to small dimensionality (only 8 features)

## 5.3 Testing the influence of different underlying classification models

In co-training studies, the most commonly used classification model used for training individual views in co-training is Naïve Bayes (NB) [3][5][6][11]. Authors in [6] have experimented with NB, SVM [6][12] and RBF Nets [6][13] and found that co-training performance depends on the choice of the underlying classifiers. In this section we will analyze the influence of the underlying classification models on the performance of algorithms implemented in *RSSalg software*. Similarly to [6] we will consider NB, SVM and RBF Nets. In this experiment we keep the values of all parameters same as in section 5.1 and only vary the view classification models[14]. Experiments with NB classifier are presented in section 5.1.5, the experiments performed using SVM are presented in table 4 and graphically summarized in figure 11, the experiments performed using RBF Nets are presented in table 5 and graphically summarized in figure 12.

[11] The implementation of Naïve Bayes can be found in Weka's class weka.classifiers.bayes.NaiveBayes http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayes.html

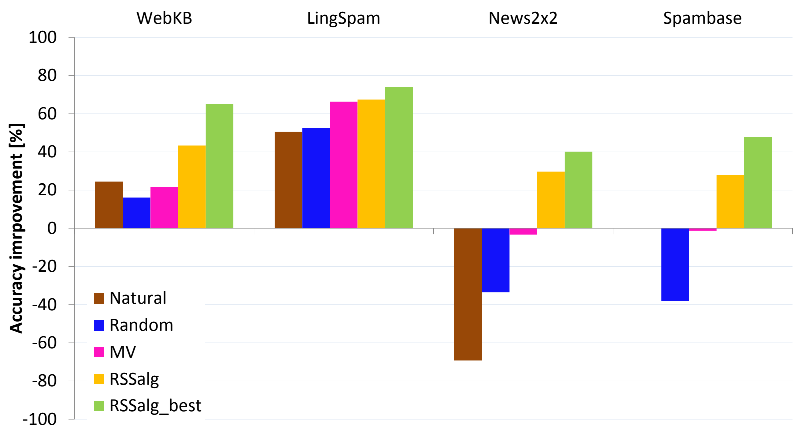
[12] The implementation of SVM can be found in Weka's class weka.classifiers.functions.SMO http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SMO.html

[13] The implementation of RBF Nets can be found in Weka's class weka.classifiers.functions.RBFNetwork http://weka.sourceforge.net/doc.stable/weka/classifiers/functions/RBFNetwork.html

[14] The created experiment (division in folds and L, U and T in each fold) is exactly the same as used in experiments with NB

**Table 4.** The performance of the algorithms (accuracy ± standard deviation) obtained using **SVM** for underlying view classifiers in co-training. The performance of **Natural** setting is reported only for WebKB, LingSpam and News2x2 datasets where it was applicable

| **Datasets** | $\mathbf{L_{acc}}$ | $\mathbf{All_{acc}}$ | **Natural** | **Random** | **MV** | **RSSalg** | **RSSalg best** |
|---|---|---|---|---|---|---|---|
| WebKB | 82.7±12.0 | 97.0±0.4 | 86.2±4.1 | 85.0±5.0 | 85.8±4.3 | 88.9±5.1 | 92.0±2.9 |
| LingSpam | 67.5±8.0 | 94.8±1.0 | 81.3±2.6 | 81.8±3.8 | 85.6±2.4 | 85.9±0.9 | 87.7±2.0 |
| News2x2 | 78.5±5.4 | 96.7±0.5 | 65.9±6.8 | 72.4±4.9 | 77.9±2.5 | 83.9±2.0 | 85.8±1.9 |
| Spambase | 74.2±6.7 | 89.9±1.0 | | 68.2±13.5 | 74.0±4.5 | 78.6±5.7 | 81.7±3.5 |
| Hepatitis | 68.8±17.5 | 85.3±3.3 | | 78.7±2.2 | 79.8±1.0 | 80.3±3.1 | 84.3±2.8 |
| Kr-vs-kp | 65.9±4.4 | 95.5±0.9 | | 56.2±4.3 | 61.5±3.8 | 65.5±4.2 | 66.2±4.1 |
| Credit-g | 55.0±12.8 | 73.5±1.3 | | 68.2±2.1 | 70.1±0.2 | 69.2±1.4 | 71.3±1.3 |
| Heart-statlog | 75.2±5.2 | 83.5±3.6 | | 74.6±2.6 | 76.8±3.3 | 79.1±3.8 | 81.9±3.8 |
| Sonar | 54.6±7.1 | 78.2±3.9 | | 50.5±7.2 | 58.2±10.6 | 56.4±8.9 | 61.7±8.5 |
| Ionosphere | 70.9±7.3 | 87.7±3.0 | | 65.5±6.1 | 67.9±7.7 | 77.3±7.6 | 81.3±5.9 |
| Breast-cancer | 56.6±9.7 | 69.3±3.9 | | 68.4±3.1 | 70.4±1.9 | 67.7±4.8 | 72.1±4.7 |
| Credit-a | 75.6±7.6 | 84.6±2.8 | | 66.6±10 | 70.3±4.2 | 79.3±4.9 | 81.1±6.0 |
| Breast-w | 90.2±7.8 | 96.5±0.7 | | 95.9±0.8 | 96.6±0.8 | 96.1±1.0 | 96.6±0.8 |
| Mushroom | 77.1±10.3 | 100.0±0.0 | | 76.7±11.4 | 86.9±6.3 | 84.1±9.4 | 86.4±8.1 |
| Diabetes | 59.1±9.4 | 76.8±1.6 | | 65.8±1.7 | 65.8±1.6 | 66.0±2.2 | 68.1±2.4 |



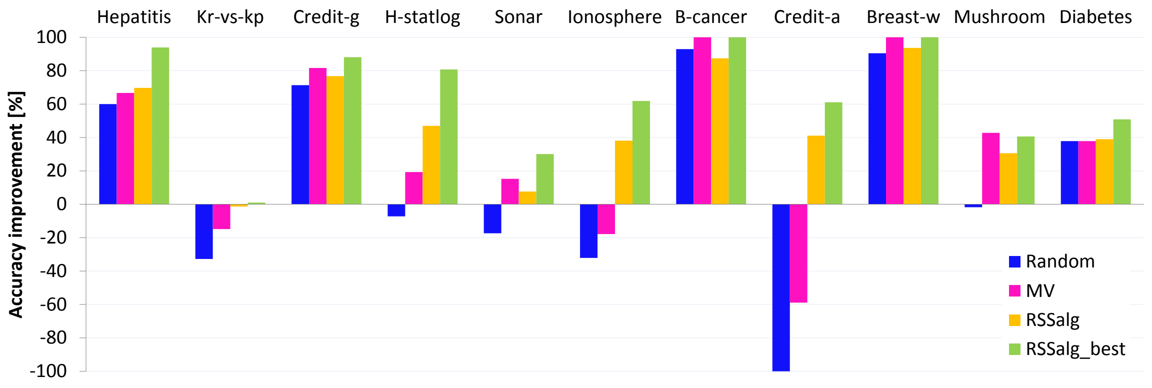


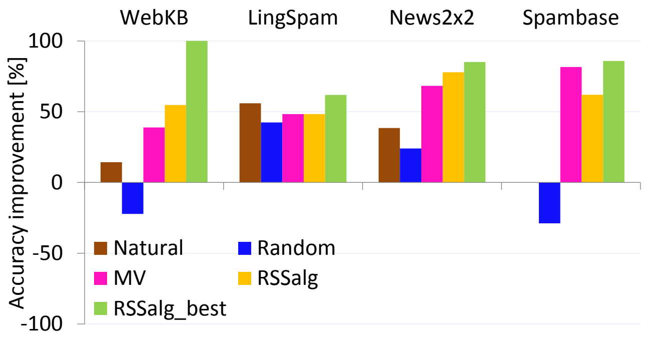
**Figure 11. SVM classifier:** percentage improvement of accuracy calculated as ($\mathbf{Alg_{acc}}$– $\mathbf{L_{acc}}$)/($\mathbf{All_{acc}}$ – $\mathbf{L_{acc}}$)·100% where $\mathbf{Alg_{acc}}$ denotes the algorithm accuracy. Negative values show degradation of accuracy compared to initial classifier $\mathbf{L_{acc}}$. **Natural** was only applicable on WebKB, LingSpam and News2x2 datasets which have a defined feature split. **Top:** natural language datasets characterized by high level of feature redundancy; **Bottom:** UCI datasets.

**SVM classifier**: the results presented in table 4 and figure 11 indicate that:

- When using SVM classifier, the redundancy of the dataset features does not seem to have the influence on algorithm performance (contrary to results obtained using NB)
- The best performing setting is again **RSSalg$_{best}$**. It is always able to improve the accuracy of the initial classifier (T1 is satisfied), but it is not always close to the goal performance (**All$_{acc}$**) as it mostly was when using NB.
- **RSSalg** is more stable when using SVM – except for the small degradation of accuracy (0.4) on Kr-vs-kp dataset, it was always able to improve the initial accuracy, thus we can say that it satisfies T1. Its performance is generally close to **RSSalg$_{best}$** (T7 is satisfied). **RSSalg** generally performs better than **MV** (T6 is satisfied) considering that MV is less reliable in terms of the degradation of initial accuracy.
- **MV** is less reliable and sometimes results in the degradation of initial accuracy (T1 is not always satisfied). This is also the case for **Random** setting. As expected, **MV** and **RSSalg** always outperform **Random** (T5 is satisfied).
- As in experiments with NB, we can observe that **Random** sometimes outperforms **Natural.**

**Table 5.** The performance of the algorithms (accuracy ± standard deviation) obtained using **RBF Nets** for underlying view classifiers in co-training. The performance of *Natural* setting is reported only for WebKB, LingSpam and News2x2 datasets where it was applicable. Weka's RBFNetwork was unable to build a classifier on a small amount of labeled data on Hepatitis, Credit-g and Sonar datasets, thus we have excluded these datasets from the evaluation.

| **Datasets** | **L$_{acc}$** | **All$_{acc}$** | **Natural** | **Random** | **MV** | **RSSalg** | **RSSalg best** |
|---|---|---|---|---|---|---|---|
| WebKB | 73.0±10.4 | 85.6±1.6 | 74.8±4.9 | 70.2±5.7 | 77.9±3.5 | 79.9±4.6 | 86.7±2.7 |
| LingSpam | 77.8±7.9 | 89.6±0.7 | 84.4±1.8 | 82.8±1.4 | 83.5±0.3 | 83.5±1.2 | 85.1±0.7 |
| News2x2 | 76.9±5.3 | 97.7±0.8 | 84.9±8.3 | 81.9±9.0 | 91.1±2.8 | 93.1±2.3 | 94.6±1.4 |
| Spambase | 64.8±12.1 | 81.1±1.9 | | 60.1±14.5 | 78.1±7.8 | 74.9±6.0 | 78.8±4.19 |
| Kr-vs-kp | 62.2±4.3 | 82.4±2.6 | | 54.8±7.4 | 61.5±5.0 | 62.1±5.3 | 65.4±4.2 |
| Heart-statlog | 73.5±6.9 | 83.4±2.8 | | 74.4±7.4 | 81.5±4.5 | 75.6±5.2 | 82.6±4.2 |
| Ionosphere | 71.1±7.3 | 91.6±1.8 | | 73.6±15 | 80.8±9.3 | 75.1±7.6 | 84.8±6.0 |
| Breast-cancer | 52.2±11.5 | 73.3±2.8 | | 67.1±6.3 | 69.9±4.1 | 68.7±4.6 | 72.8±3.2 |
| Credit-a | 62.4±6.6 | 80.9±3.8 | | 62.0±15.6 | 77.5±3.3 | 73.9±4.2 | 78.2±3.5 |
| Breast-w | 84.7±12.2 | 95.3±1.9 | | 83.0±20.6 | 96.1±1.5 | 94.4±2.5 | 96.3±1.4 |
| Mushroom | 79.7±9.6 | 98.3±0.2 | | 76.6±8.7 | 83.3±5.5 | 83.1±8.1 | 84.8±7.1 |
| Diabetes | 57.7±7.2 | 73.3±1.5 | | 59.2±8.2 | 67.6±3.6 | 65.0±5.8 | 70.5±3.1 |



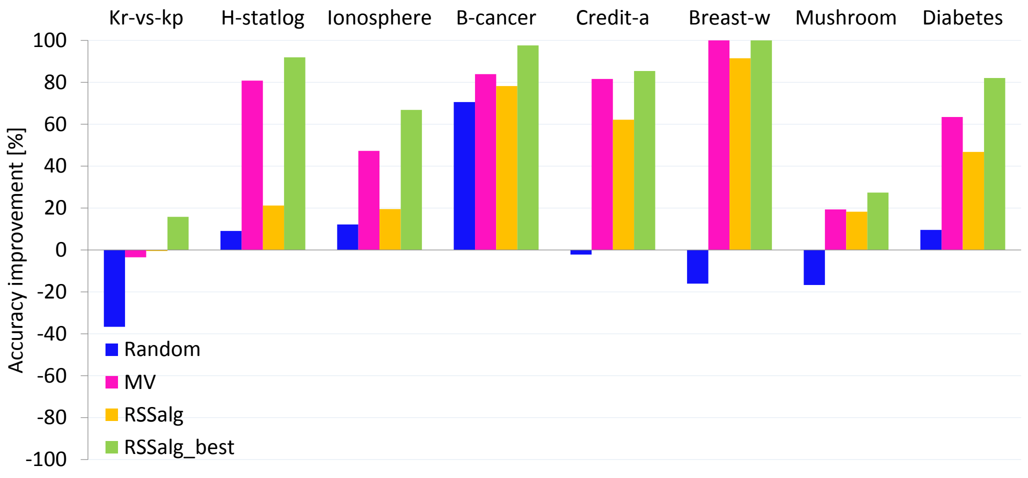


**Figure 12. RBF classifier:** percentage improvement of accuracy calculated as (**Alg**$_{acc}$– **L**$_{acc}$)/(**All**$_{acc}$ – **L**$_{acc}$)·100% where **Alg**$_{acc}$ denotes the algorithm accuracy. Negative values show degradation of accuracy compared to initial classifier **L**$_{acc}$. **Natural** was only applicable on WebKB, LingSpam and News2x2 datasets which have a defined feature split. **Top:** natural language datasets characterized by high level of feature redundancy; **Bottom:** UCI datasets

**RBF classifier**: the results presented in table 5 and figure 12 indicate the following:

- Similarly to SVM and contrary to NB, the redundancy of the dataset features does not seem to have the influence on algorithm performance
- The best performing setting is again **RSSalg**$_{best}$. It is always able to improve the accuracy of the initial classifier (T1 is satisfied), but it is not always close to the goal performance (**All**$_{acc}$).
- Except for the small degradation of accuracy (0.1) on Kr-vs-kp dataset, **RSSalg** was always able to improve the initial accuracy, thus we can say that it satisfies T1. Its performance is generally close to **RSSalg**$_{best}$ (T7 is satisfied). These conclusions about **RSSalg** are similar to those obtained using SVM.
- With the exception of kr-vs-kp dataset, **MV** has always improved the initial accuracy and thus satisfied T1. When using RBF classifier **MV** is generally close to or better than **RSSalg**, thus T6 is not satisfied. This is similar to the results obtained using NB.
- When using a RBF classifier, **Natural** always outperformed **Random** and was always able to improve the accuracy (T1 is satisfied)**.** This confirms the result obtained in [6] where authors show that RBF outperforms SVM and NB in a co-training setting with a natural feature split.

To summarize, our experiments with different underlying classifiers indicate that out of considered algorithms:

- **RSSalg** is the best choice when using SVM
- **MV** is the best choice when using RBF nets

- When using NB, **RSSalg** yields somewhat better performance on more redundant datasets and it is comparable to **MV** on less redundant datasets.
- **Natural** generally performs worse than **RSSalg** and **MV**. It has displayed best results when applied with RBF nets.
- **Random** displayed the best performance when applied with NB on more redundant datasets.
- In all experiments $\textbf{RSSalg}_{\textbf{best}}$ was the best performing setting. While it is not a realistic setting (it utilizes test data labels for determination of threshold values) it shows the potential of **RSSalg**. An improved threshold determination procedure would boost the performance of **RSSalg** and make it more widely applicable.

# 6 Creating the property files and running the experiment: a tutorial on News 2x2 dataset

In this section we will give a step-by-step tutorial on creating and running a new experiment with *RSSalg software*. We will demonstrate this on the example of running News2x2 experiment. Section 6.1 describes the problem we are trying to solve and the experimental setting for evaluating the solution. Section 6.2 describes how to install and start *RSSalg software*. Section 6.3 describes how to set the experiment and parameter values in *RSSalg software*. Finally, section 6.4 describes how to run the set experiment and interpret the software output.

## 6.1 Problem setting

News 2x2 dataset is a semi-artificial dataset introduced in [5]. It was artificially created to be a binary classification problem with class-conditional independence, ideal for testing the performance of co-training. 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.

### 6.1.1 Experimental setting

We will test our solution using a 10-fold-cross validation procedure introduced in [6]. 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.

### 6.1.2 Co-training parameters

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

- the number of iterations of co-training algorithm *k* is 20;
- a small unlabeled pool *u’* of 50 instances is used;
- growth size *p/n*: 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**, **MV** 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 [6] when applying co-training on News2x2 dataset.

## 6.2 Installing and starting *RSSalg software*

*RSSalg software* requires Java Runtime Environment (JRE) 1.7 or above[15]. Once Java is installed, you can build the project by using Apache Ant or by loading the source code into your favorite IDE (section 6.2.1). After building you can run the project by using the packaged JAR executable (section 6.2.2) or by using the provided Apache Ant script (section 6.2.3).

### 6.2.1 Building the project

The easiest way to build and run the project is by using Apache Ant[16]. You will firstly need to install Ant. After installing Ant, open the terminal and navigate to the root of the project. There you will find the Ant script named "build.xml"[17]. This script assumes there is a folder named 'lib' next to the source folder 'src' containing `weka.jar` as in figure 13 (version 3.7.0). In order to compile and package the software in an executable JAR file one should run *jar* target defined in 'build.xml' script. In the terminal type the command:

`ant`.

This will compile and package the code in an executable `RSSalg.jar` located inside the folder `<project_root>/dist/jar/`.

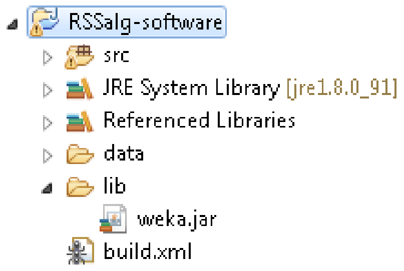


**Figure 13.** *RSSalg software* folder structure: 'src' – source folder; 'data' – folder contacting the datasets, property files and algorithm outputs; 'lib' – folder contacting external libraries needed for code compilation (currently only weka.jar version 3.7.0)

Optionally, you load the source code from the `src` folder[18] into your favorite IDE. Note that *RSSalg software* implementation depends on Weka (version 3.7.0). Thus, you will need to add the weka.jar library (located inside the folder `<project_root>/lib`[19]) to your project.

### 6.2.2 Running the project by using the built executable

To run *RSSalg software* you can use the packaged `RSSalg.jar` executable obtained from the software release or by building the project (section 6.2.1).

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 6.3 and commented examples of these files for running News2x2 experiment described in this document can be found in `dist/data/News2x2/experiment` folder[20].

*RSSalg software* can be run in two modes: GUI (Graphical User Interface) mode (section 6.2.2.1) or console mode (section 6.2.2.2). 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.

#### *6.2.2.1 GUI mode*

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

[15] To install Java go to http://www.oracle.com/technetwork/indexes/downloads/index.html#java
[16] Apache Ant can be obtained from http://ant.apache.org/
[17] https://github.com/slivkaje/RSSalg-software/blob/master/build.xml
[18] https://github.com/slivkaje/RSSalg-software/tree/master/src
[19] https://github.com/slivkaje/RSSalg-software/blob/master/lib/weka.jar
[20] https://github.com/slivkaje/RSSalg-software/tree/master/dist/data/News2x2/experiment

```
java –jar RSSalg.jar
```

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

#### 6.2.2.2 *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 6.3.

`<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 6.3.4.

For example, if a user wishes to execute the $L_{acc}$ setting on News2x2 dataset he should download the `/data` folder[21] and place it in the same folder as `RSSalg.jar` and type the following command:

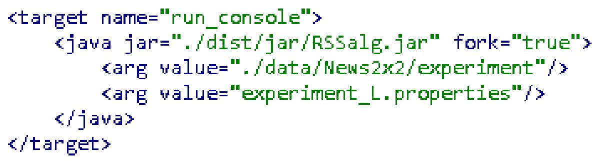
```
java -jar RSSalg.jar ./data/News2x2/experiment experiment_L.properties
```

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

### 6.2.3 Running the project by using Apache Ant

Open the terminal and navigate to the root of the project (containing build.xml):

- You can run *RSSalg software* in GUI mode by typing the command:
  `ant run_GUI`
- You can run *RSSalg software* in console mode by typing the command
  `ant run_console`.
  In this case you need to modify the arguments of target run_console located in build.xml to correspond to the adequate *properties folder* and *experiment* properties (figure 14).



```
<target name="run_console">
    <java jar="./dist/jar/RSSalg.jar" fork="true">
        <arg value="./data/News2x2/experiment"/>
        <arg value="experiment_L.properties"/>
    </java>
</target>
```

**Figure 14.** An example of the `run_console` target (build.xml): the experiment that will be run when executing this target is $L_{acc}$ on the News2x2 dataset. The the experiment that will be run is determined by the two arguments (<arg> elements): *properties folder* and *experiment properties* (see section 6.2.2)

## 6.3 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 6.3.1.6
- **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 6.3.2.1

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

- **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 6.3.3.1
- **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 6.3.4.7
- **genetic algorithm settings** are only needed for **RSSalg** setting 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 6.3.5.1.

The applications GUI allows the easy creation of the needed propety files. Figure 15 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 15).

### 6.3.1 Specifying dataset settings

The dialog for dataset properties entry is illustrated in Figure 16. User can choose between the two options: (1) prepare a new cross-validation experiment (explained in section 6.3.1.1) or (2) load previously prepared cross-validation experiment from files (explained in section 6.3.1.2). 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 6.3.1.3. Section 6.3.1.4 describes how the user can specify a classification model used for each of the individual views. Finally, section 6.3.1.5 describes how to save the selected settings and describes the resulting property files. Section 6.3.1.5 also describes how to load a previously created property file.

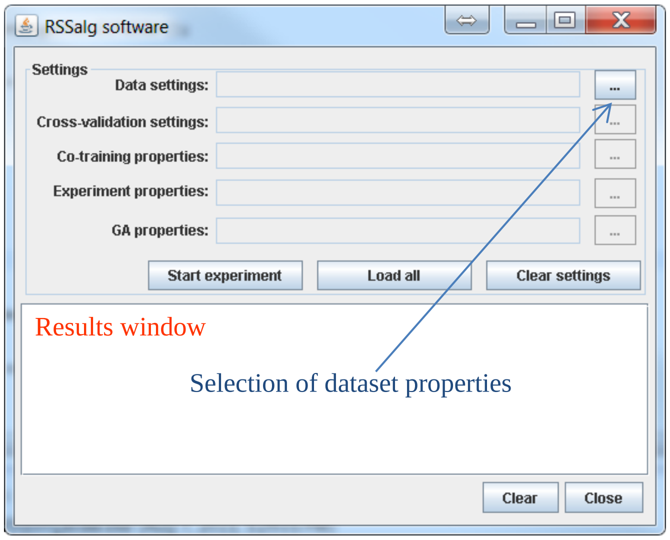
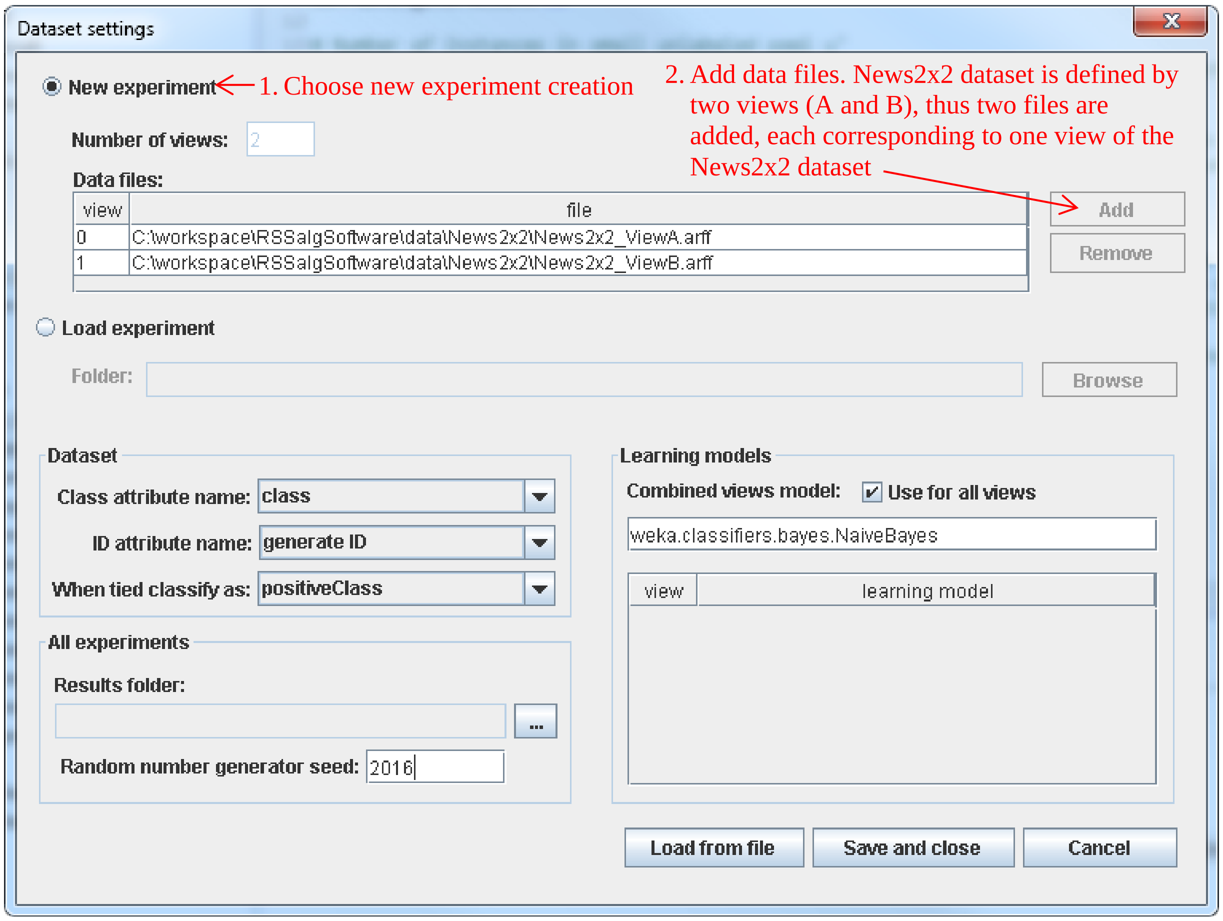


**Figure 15.** The main window of *RSSalg software* application



Dataset settings

● 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 |

Add | Remove

○ Load experiment

Folder: | Browse

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 |
|---|---|

Load from file | Save and close | Cancel

**Figure 16.** Dataset settings input form: creation of the new cross-validation experiment for News2x2 dataset

#### 6.3.1.1 Creating a new cross-validation experiment

Figure 16 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[22]. 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[23].

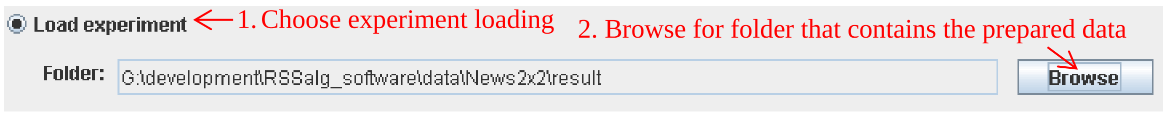
Currently, *RSSalg software* only allows two views of the data. If the user provides only one dataset file, *RSSalg software* will automatically create the second view by using only class and id attribute. Later, during experiment execution, other features might be added to the second view if needed (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 6.3.1.2 for the structure of the recorded cross-validation experiment and to section 6.3.1.3 for the explanation on specifying the *result folder*.

[22] 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.
[23] 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

### 6.3.1.2 *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 17).


● Load experiment
1. Choose experiment loading
2. Browse for folder that contains the prepared data
Folder:
Browse

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

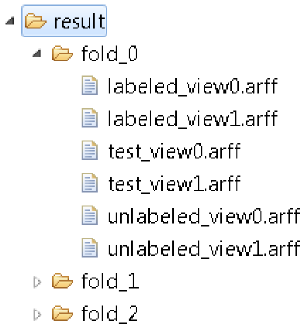
**Figure 17.** Loading an already prepared experiment (part of the dataset settings input form, figure 16)

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 18): 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, $i \in \{0,..,n\text{-}1\}$. 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.



```
◢ 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 18.** File structure of the directory *result* that contains an already prepared 3-fold-cross-validation experiment for the dataset that has two views

### 6.3.1.3 *Defining general data properties*

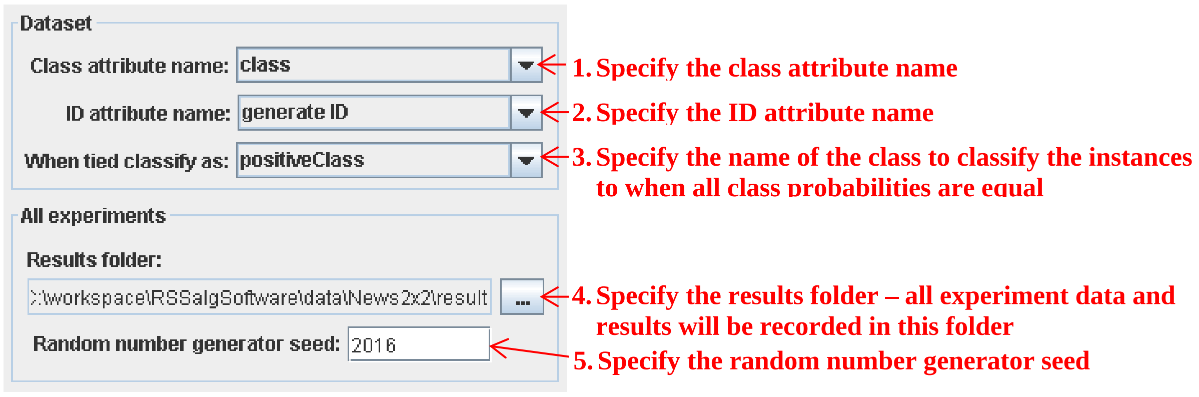
After the selection of whether to create a new or load an existing cross-validation experiment, the general data properties listed in figure 19 should be provided:

- 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 19), 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 6.3.1.3) 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

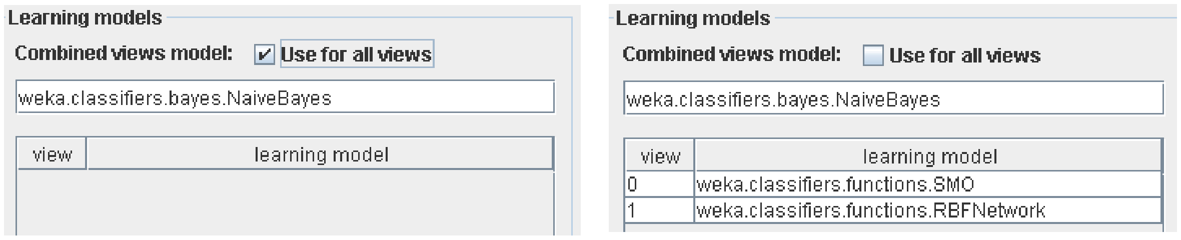
Random number generator seed: 2016

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 19.** Entering the general data properties (part of the dataset settings input form, figure 16)

### *6.3.1.4 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 20, left) or to use different classification models (as shown in figure 20, right). User can provide arbitrary classifiers from Weka libraries [7] 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.SMO |
| 1 | weka.classifiers.functions.RBFNetwork |

**Figure 20.** Choosing classification models (part of the dataset settings input form, figure 16). 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 $\mathbf{L}_{acc}$ and $\mathbf{All}_{acc}$, as well as the model used for the final classifier in **RSSalg**.

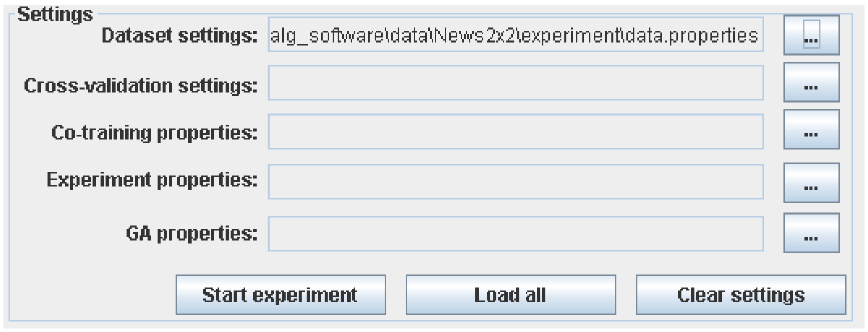
For News2x2 experiment, a Naïve Bayes classifier is specified to be used for both individual views as well as for the final classifier, as shown in in figure 20, left.

### *6.3.1.5 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 16). 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 16). 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 21.


Settings

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

Cross-validation settings:

Co-training properties:

Experiment properties:

GA properties:

**Figure 21.** The main window of the application after the successful save of data properties

#### *6.3.1.6 The resulting data.properties file*

Table 6 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.

**Table 6.** The meaning of properties saved in 'data.properties' file

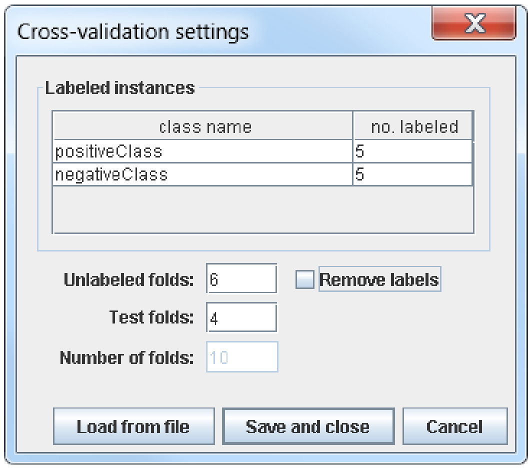
| Property name | Description | Value |
|---|---|---|
| resultFolder | Folder in which the performed cross-validation experiment and results will be recorded | String representing a folder location |
| classNames | Names of the categories | Quoted Strings separated with whitespace character |
| combinedClassifier | The combined classifier: supervised model trained on full attribute set acquired by merging all views. It is used in $\mathbf{L}_{acc}$ and $\mathbf{All}_{acc}$ 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 |

### 6.3.2 Specifying cross-validation experiment settings

The cross-validation experiment settings are used if user chose the "New experiment" option (section 6.3.1.1). Otherwise (if the user chose the "Load experiment" option, section 6.3.1.2), user will still be able to access these settings in order to review the loaded experiment, but will be unable to change them.

Figure 22 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 22), 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 22.** 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 22). User is prompted to specify the directory the settings will be saved to in the file automatically named 'cv.properties'.

#### 6.3.2.1 The resulting cv.properties file

Table 7 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.

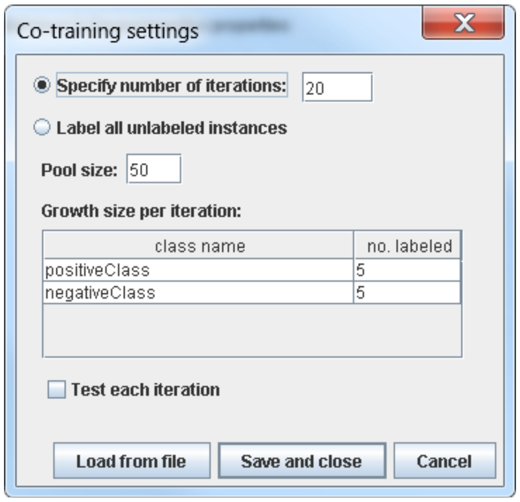
**Table 7.** The meaning of properties saved in 'cv.properties' file

| 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) |

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 22). User will be prompted to select a directory which must contain the file named 'cv.properties' from which the properties will be loaded.

### 6.3.3 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 23.


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 23.** 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 23), 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, section 6.3.1.3). For example, if the **RSSalg** setting 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 24.

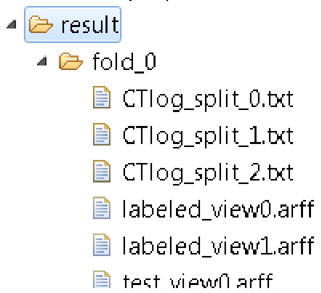
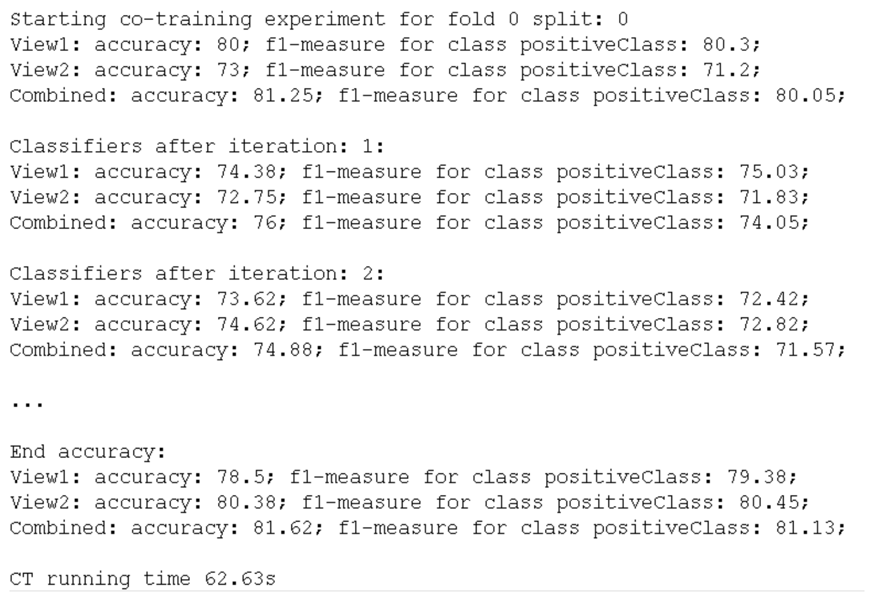

```
◢ result
   ◢ fold_0
        CTlog_split_0.txt
        CTlog_split_1.txt
        CTlog_split_2.txt
        labeled_view0.arff
        labeled_view1.arff
        test_view0.arff
```

**Figure 24.** 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', $i \in \{0,..,2\}$

The example of one of these log files is shown in figure 25. Individual co-training classifiers, as well as the combined co-training style classifier[24] 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 (section 6.3.4).



```
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 25.** 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 23). 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 23). User will be prompted to select a directory which must contain the file named 'co-training.properties' from which the properties will be loaded.

#### 6.3.3.1 The resulting co-training.properties file

Table 8 lists all available co-training properties as written in 'co-training.propertes' file, their meaning and possible values.

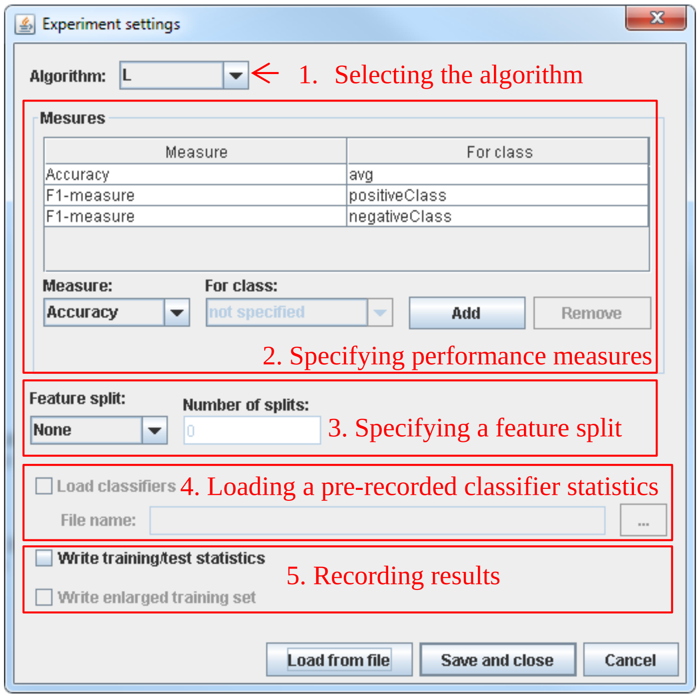
**Table 8.** The meaning of properties saved in 'co-training.properties' file

| 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) |

[24] 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

### 6.3.4 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 26.


Experiment settings
Algorithm: L
1. Selecting the algorithm
Mesures
2. Specifying performance measures
Feature split: None
Number of splits: 0
3. Specifying a feature split
Load classifiers
4. Loading a pre-recorded classifier statistics
File name:
Write training/test statistics
Write enlarged training set
5. Recording results
Load from file
Save and close
Cancel

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

Measure: Accuracy  For class: not specified  Add  Remove

**Figure 26.** 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 2.3. 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**$_{\mathbf{best}}$ 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 27 user chooses to obtain an average of *f*1-measure for ‘positiveClass’ and *f*1-measure for ‘negativeClass’).

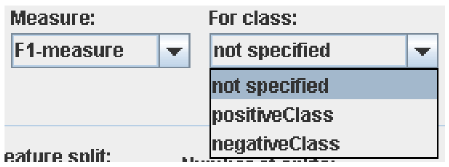


**Figure 27.** Choosing to calculate an average of *f*1-measure for ‘positiveClass’ and *f*1-measure for ‘negativeClass’ by selecting “not specified”. The category list is automatically obtained from the data settings by examining the provided class attribute

User may provide several measures which will be calculated. For example, in figure 26, 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

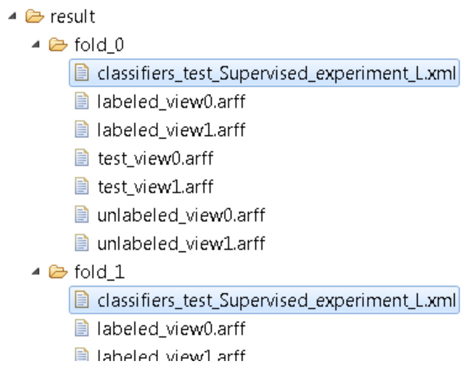
“Load classifiers” check box denotes whether to load a previously recorded classifier statistic or not. Selecting to use the classifier statistic will enable the user to specify the XML file that contains the recorded statistic.

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

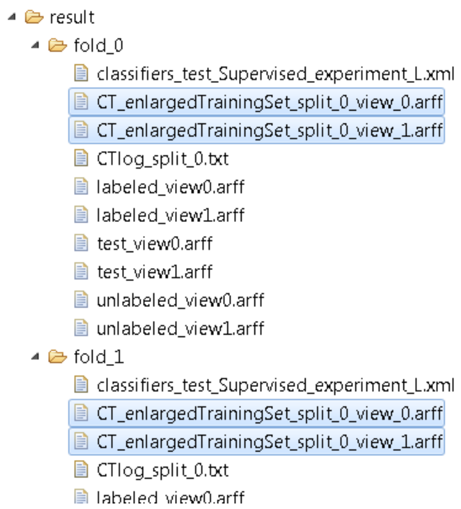
User can choose to record both training and testing classifier statistic for the executed algorithm by selecting the “Write training/test statistics” option. The statistic will be saved in the directory defined by the result folder (property resultFolder, table 6) to the file whose name will be automatically determined by the executed algorithm (please refer to later subsections for the file names in which the statistics of the algorithms will be recorded). Figure 28 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 28.** 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 6) to the file whose name will be automatically determined by the executed algorithm (see later subsections for the file names for specific algorithms). Figure 29 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 29.** 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 26). 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 26). User will be prompted to select a *.properties* file containing the desired experiment.

### 6.3.4.1 Running the $L_{acc}$ setting

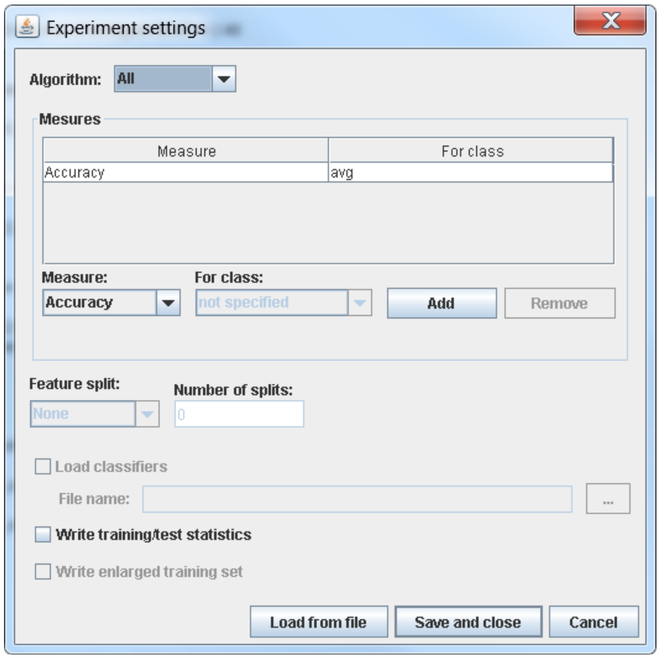
The input form for experiment settings entry with the selection of *L* setting is shown in figure 26. In $L_{acc}$ setting, all features are merged in a unique attribute set and the classifier is trained on the labeled portion of the data (i.e. feature values from all other views are copied to the first 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 28.

Figure 26 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’.

### 6.3.4.2 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 30.


Experiment settings

Algorithm: All

Mesures

| Measure | For class |
|---|---|
| Accuracy | avg |

Measure: Accuracy

For class: not specified

Add Remove

Feature split: None

Number of splits: 0

☐ Load classifiers

File name:

☐ Write training/test statistics

☐ Write enlarged training set

Load from file Save and close Cancel

**Figure 30.** 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 (section 6.3.2), $All_{acc}$ setting will reduce to the $L_{acc}$ setting as the labels for unlabeled data are unavailable.

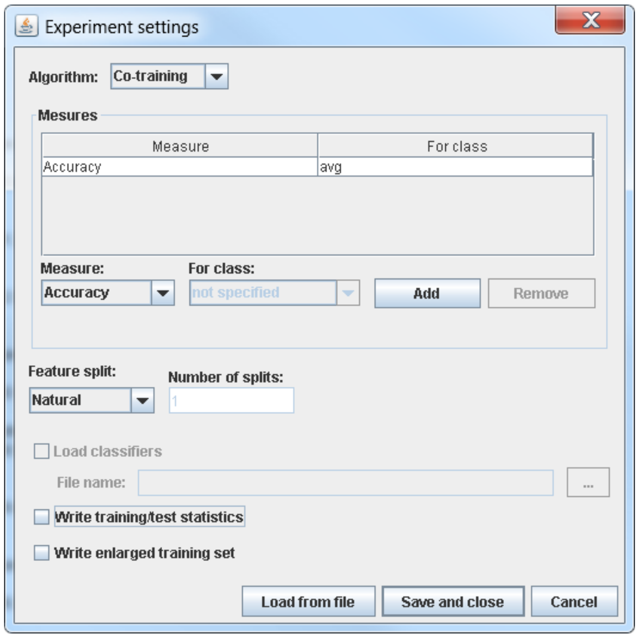
The 'Feature split' choice list and the 'Number of splits' field are disabled and set to 'None' and 0, respectively, as the supervised experiment $All_{acc}$ does not rely on a feature split. As the $All_{acc}$ setting does not rely on training or testing classifier statistics, 'Load classifiers' choice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

#### 6.3.4.3 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 31.


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 31.** 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 section 6.3.1.1) 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.

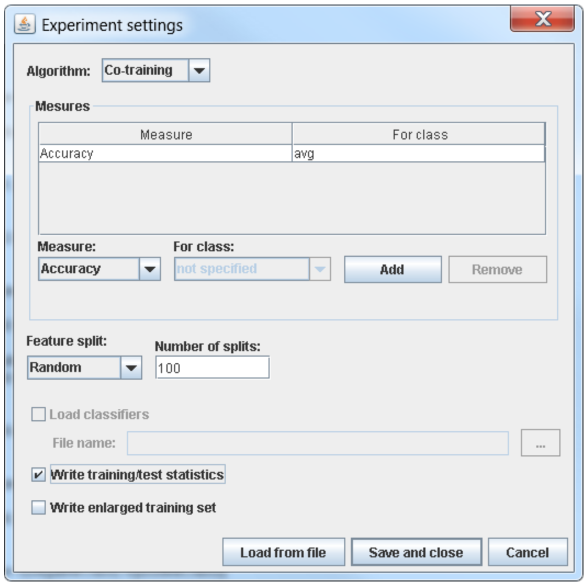
User can choose to save both *training* and *testing* classifier statistic by selecting the ‘Write training/test statistic’ choice. The resulting files are named ‘classifiers_CoTraining.xml’ and ‘classifiers_test_CoTraining.xml’, respectively.

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’.

#### 6.3.4.4 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 32.


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 32.** 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$_{\text{best}}$** experiments and created *test* statistics for **MV** experiment. The name and the location of the resulting train and test statistic is shown in figure 33 .

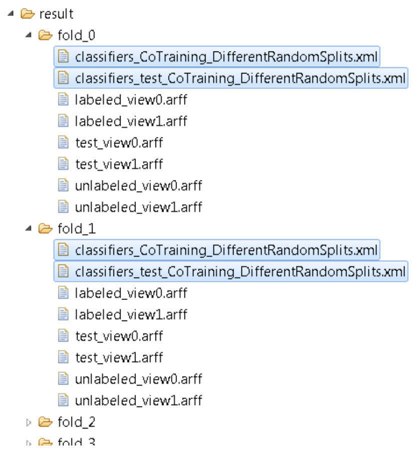

```
result
  fold_0
    classifiers_CoTraining_DifferentRandomSplits.xml
    classifiers_test_CoTraining_DifferentRandomSplits.xml
    labeled_view0.arff
    labeled_view1.arff
    test_view0.arff
    test_view1.arff
    unlabeled_view0.arff
    unlabeled_view1.arff
  fold_1
    classifiers_CoTraining_DifferentRandomSplits.xml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

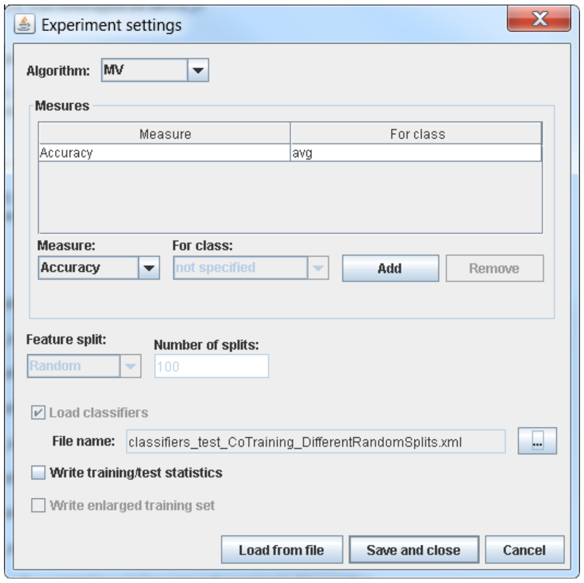
**Figure 33.** The result of saving the training and test statistics are the files named 'classifiers_algorithm.xml' and 'classifiers_test_algorithm.xml', respectively, where algorithm denotes the name of the applied algorithm. Figure shows the result of saving the training and test statistic for co-training 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'.

#### 6.3.4.5 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 34.



Experiment settings

Algorithm: MV

Mesures

| Measure | For class |
|---|---|
| Accuracy | avg |

Measure: Accuracy

For class: not specified

Feature split: Random

Number of splits: 100

[✓] Load classifiers

File name: classifiers_test_CoTraining_DifferentRandomSplits.xml

[ ] Write training/test statistics

[ ] Write enlarged training set

**Figure 34.** 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 33). 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[25]. In this experiment (figure 34), 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[26]. 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.

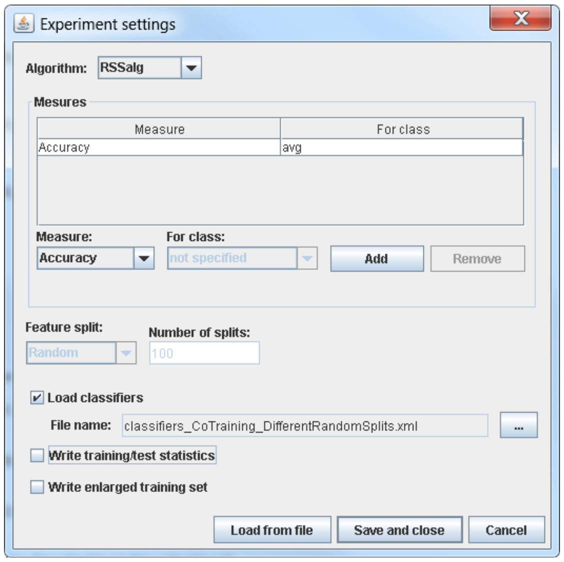
When the user choses to save the input properties for the *MV* setting, a *.properties* file is automatically named 'experiment_MV.properties'.

[25] 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)

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

### 6.3.4.6 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 35.



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 35.** The experiment settings input form adjusted for running the *Rsalg* setting on News2x2 dataset

**RSSalg** relies on using the training 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 *training* statistic in the results folder in XML files with names starting with 'classifiers_' (see figure 33), 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[27]. When training 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 35), we assume that the Random setting was already ran and that its training 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.

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.

[27] 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)

#### 6.3.4.7 The resulting experiment property file

Table 9 lists all available experiment properties, their meaning and possible values.

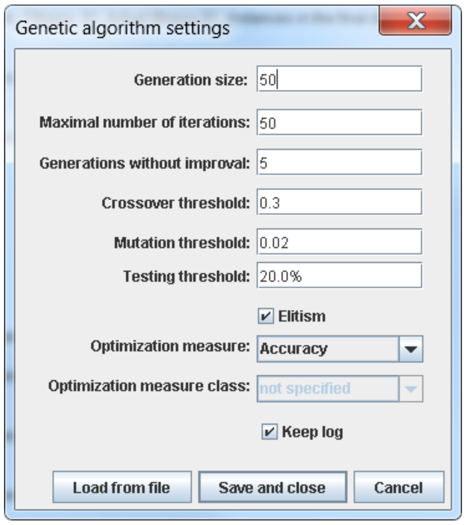
**Table 9.** The meaning of properties saved in experiment *.properties* file

| Property name | Description | Value |
|---|---|---|
| algorithm | The performed algorithm | String value: a full-package name of algorithm implementation. Possible values:<br>• `algorithms.SupervisedAlgorithm_L` (*$L_{acc}$*)<br>• `algorithms.SupervisedAlgorithm_All` *($All_{acc}$)*<br>• `algorithms.co_training.CoTraining` *(Random, Natural)*<br>• `algorithms.RSSalg.MajorityVote` *(MV)*<br>• `algorithms.RSSalg.RSSalg` *(***RSSalg**, **$\mathbf{RSSalg_{best}}$***)*<br>• 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:<br>• `classificationResult.measures.AccuracyMeasure`<br>• `classificationResult.measures.F1Measure` `classificationResult.measures.Precision` `classificationResult.measures.Recall`<br>• 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:<br>• `featureSplit.RandomSplit` (random split)<br>• `featureSplit.DifferentRandomSplitsSplitter` (unique random splits)<br>• 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:<br>• `algorithms.RSSalg.resultStatistic.Label.MajorityVotes`<br>• 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:<br>• `algorithms.RSSalg.GA.RSSalgCandidateEvaluator` (**RSSalg** setting)<br>• `algorithms.RSSalg.GA.TestSetAccuracyCandidateEvaluator` (**$\mathbf{RSSalg_{best}}$** setting)<br>• Arbitrary implementation of `algorithms.RSSalg.GA.CandidateEvaluatorIF` |

### 6.3.5 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 36 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[28]; 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[29].


Genetic algorithm settings

| | |
|---|---|
| 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 36.** 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 36). 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 36). User will be prompted to select a directory which must contain the file named 'GA.properties' from which the properties will be loaded.

[28] The best individual from the generation will be copied to the next generation
[29] The full path to each file is `<results_folder>/fold_<currentFold>/ThresholdOptimiserlog.txt`

#### 6.3.5.1 *The resulting GA.properties file*

Table 10 lists all available GA properties as written in 'GA.propertes' file, their meaning and possible values.

**Table 10.** The meaning of properties saved in 'GA.properties' file

| 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) |

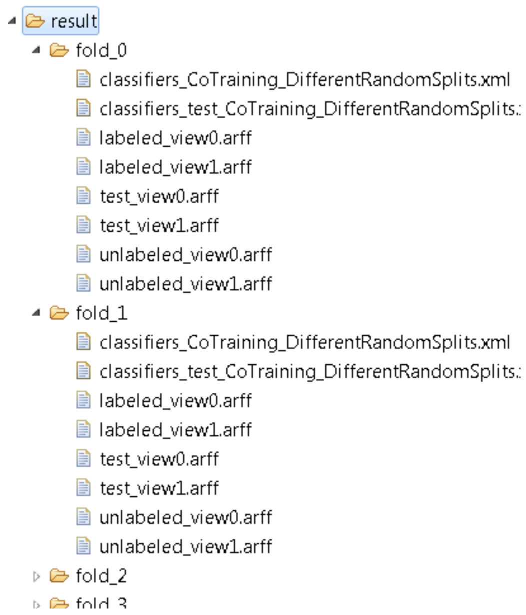
## 6.4 Running the experiment in *RSSalg software*

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 (section 6.3.2). 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 section 6.3 for detailed description of entering the desired settings. By clicking on 'Load all' button (figure 15) 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 15).

The experiment execution is started by clicking on the 'Start experiment' button. Experiment results will be shown in the results window, figure 15. 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 15).

In case the user has chosen to create a new cross-validation experiment (section 6.3.1.1), when the experiment starts, the created cross-validation experiment will be written to the specified result folder (section 6.3.1.3) as shown in figure 37. The file structure of the written data is explained in section 6.3.1.2. For the later experiments, user may want to select the option of loading the prepared experiment from the result folder as explained in section 6.3.1.2. 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 37.** The result of creating a new cross-validation experiment

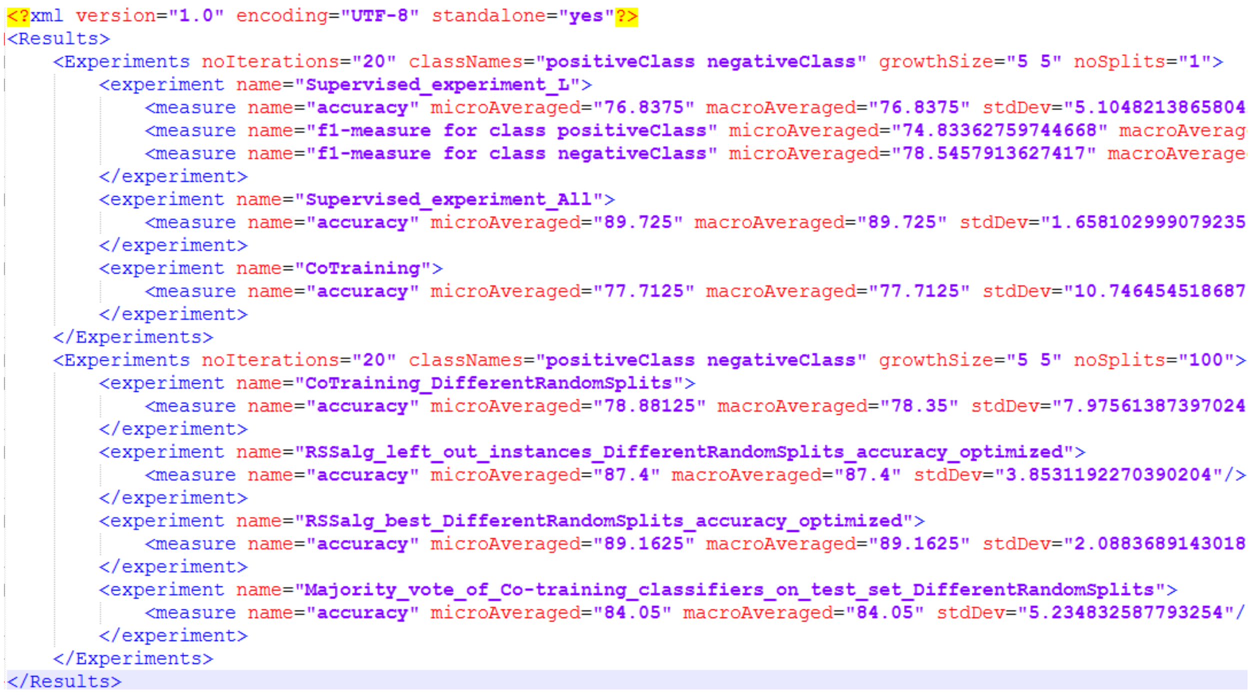
Each time an experiment is ran, the resulting micro and macro averaged measures are shown in the results window. The example output (obtained by executing $L_{acc}$ experiment of News2x2 dataset) is shown in figure 4. Each time an experiment is run, its resulting performance is recorded in 'Results.xml' 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 4).

Figure 38 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 11 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.

**Table 11.** Results of presented experiments on News2x2 dataset

| | **$L_{acc}$** | **$All_{acc}$** | **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 |



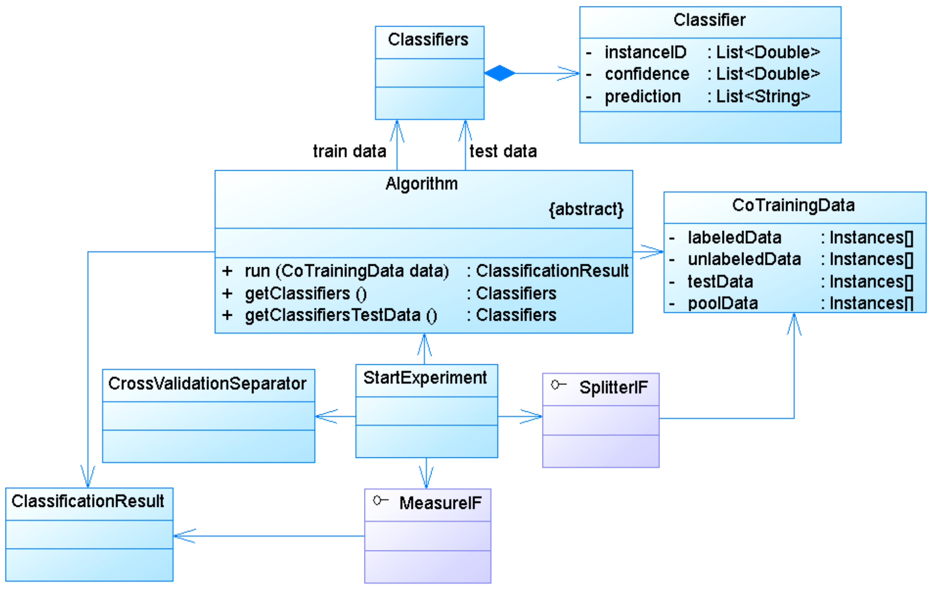
```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Results>
    <Experiments noIterations="20" classNames="positiveClass negativeClass" growthSize="5 5" noSplits="1">
        <experiment name="Supervised_experiment_L">
            <measure name="accuracy" microAveraged="76.8375" macroAveraged="76.8375" 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 38.** The 'Results.xml' file located in the results folder, obtained after running the experiments on News2x2 dataset described in this document

## 7 *RSSalg software* architecture

In this section we will give an overview of the *RSSalg software* architecture (sections 7.1-7.8) and implementation details (section 7.8). In section 7.9 we explain how *RSSalg software* can be extended programatically. Main classes of *RSSalg software* are presented in figure 39. The starting entry of the program is the `StartExperiment` class (section 0) which orchestrates training and evaluation of the selected algorithm on the selected dataset:

- The dataset is represented by `CoTrainingData` class (section 7.1).
- Experiment preparation (dividing the loaded dataset in labeled, unlabeled and test data) is implemented in `CrossValidationSeparator` class.
- Model training:
    - Before running certain algorithms, we want to make sure that the appropriate feature split is performed (e.g. if we want to run co-training with a random feature split). Different feature splitting methods are represented by classes that implement `SplitterIF` interface (setion 7.5).
    - The applied algorithm itself is represented by `Algorithm` class (section 7.2).
    - During algorithm execution, a classifier (or an ensemble of classifiers) is trained. The details of the training process can be stored in `Classifiers` object for later analysis (section 7.3). This is represented by *train data* association in figure 39.
- Model testing:
    - The result of applying the trained model on test data (predicted labels and classifier confidence) is represented by `ClassificationResult` class.
    - *RSSalg software* supports a number of common metrics for calculating algorithm performance (accuracy, *f*1-measure, precision and recall). These metrics are represented by classes implementing the `MeasureIF` interface (section 7.4).
    - The traine classifier (or ensemble) is evaluated on the test data. The details of the testing process (e.g. confidences of each classifier from the ensemble) can be stored in `Classifiers` object for later analysis (section 7.3). This is represented by *test data* association in figure 39.



Classifiers
Classifier
- instanceID : List<Double>
- confidence : List<Double>
- prediction : List<String>
train data
test data
Algorithm
{abstract}
+ run (CoTrainingData data) : ClassificationResult
+ getClassifiers () : Classifiers
+ getClassifiersTestData () : Classifiers
CoTrainingData
- labeledData : Instances[]
- unlabeledData : Instances[]
- testData : Instances[]
- poolData : Instances[]
CrossValidationSeparator
StartExperiment
SplitterIF
ClassificationResult
MeasureIF

**Figure 39.** Main classes in *RSSalg software*. Due to space restrictions and clarity some details (less important classes and methods) have been omitted from the diagram

## 7.1 `CoTraningData` class

`CoTrainingData` class presents the dataset ready for the execution of a co-training style algorithm. The dataset is divided into several parts:

- labeled data;
- unlabeled data;
- (optionally used) small unlabeled pool data *u*';
- test data

Each dataset part is represented by an array of Weka's [7] `Instances` objects (figure 39). Elements of this array correspond to different views of the data. For example, if the feature set is split into two views, labeled, unlabeled, unlabeled pool and test datasets each consist of an array containing two `Instances` objects. Both `Instances` objects contain the same instances described by different attributes[30].

The `CoTrainingData` class encompasses a number of useful methods for co-training data manipulation. Some examples are:

- assigning a label to an unlabeled instance and simultaneously transferring that instance to the labeled set;
- testing the performance of the current classifier (a supervised classifier trained using currently available labeled data);
- testing the performance of individual views (the performance of a single view is the performance of a supervised classifier trained using only attributes from that view and currently available labeled data);
- moving an attribute from one view to another; etc.

## 7.2 `Algorithm` class

The algorithm that is being trained and evaluated during *RSSalg software* execution is represented by abstract `Algorithm` class. All concrete implementations of the settings (**L**$_{acc}$, **All**$_{acc}$, **Random**, **Natural**, **MV**, **RSSalg** or **RSSalg**$_{best}$) inherit the `Algorithm` class.

The most important method of `Algorithm` class is the `run` method. This method executes the algorithm on the dataset supplied via `CoTrainingData` object. During the execution of the `run` method, an ensemble of

[30] With the exception of the id and label attributes

classifiers will be trained in an algorithm-specific way using the supplied labeled and unlabeled data and applied on the test data. The result of the run method is the `ClassificationResult` object.

The classes inheriting the abstract `Algorithm` class (concrete strategy classes) form the obtained classifier ensemble in the following way:

- `SupervisedAlgorithm_L`: a single supervised classifier trained using only labeled instances (**L$_{acc}$** setting);
- `SupervisedAlgorithm_All`: a single supervised classifier trained using both labeled and unlabeled instances with correct label assigned (**All$_{acc}$** setting);
- `CoTraining`: a single classifier trained in co-training fashion using labeled and unlabeled instances (**Random** and **Natural** settings both use the `CoTraining` class for the implementation of the underlying algorithm. They differ in the pre-applied feature splitting method);
- `RSSalg`: the implementation of RSSalg (**RSSalg** and **RSSalg$_{best}$** settings both use this ckass for the underlying algorithm implementation. The difference between their implementation is explained in section 7.7);

`MajorityVote`: multiple classifiers trained using the `CoTraining` class. The ensemble prediction is obtained by a majority vote (**MV** setting).A user might choose to record the details about the ensemble training process. Those details can be obtained after execution of the `run` method by calling the `getClassifiers` method. `getClassifiers` method results with `Classifiers` object (*train data* association in figure 39). In this classifiers object we record the details about each instance transferred from the unlabeled to the labeled set during the training process, For example, in RSSalg this “classifier statistics” is used to derive the final training set.

Similarly, we are able to record the details about the testing process. The details about classifying test instances can be accessed via the `getClassifiersTestData` method (*test data* association in figure 39) which returns the `Classifiers` object. For example, this “classifier test statistics” can be used by the `MajorityVote` algorithm to derive the label of each test instance in a majority vote fashion.

## 7.3 `Classifiers` class

`Classifiers` class represents an ensemble of individual classifiers. An individual classifier from the ensemble is represented by the `Classifier` class (figure 39).

Each setting implemented in *RSSalg software* produces an ensemble of classifiers. For example, the **Random** setting produces *m* co-training classifiers (obtained using *m* random splits). Some settings will produce the ensemble consisting of only one classifier, for example, in **Natural** setting we train only one co-training classifier using the natural feature split.

While training an individual semi-supervised classifier we can store the “training statistics” in the `Classifier` class: for each unlabeled instance that was labeled and transferred to the labeled set we store the predicted label and classifier confidence of assigning that label. Similarly, when evaluating an individual classifier on a set of test instances, we use `Classifier` object to store the “testing statistics”: how an individual classifier of the ensemble classified each test instance (prediction and confidence).

`Classifiers` object may be stored on disk as an XML file for the later in-depth analysis. This object may also be used as input for other algorithms, For example, **RSSalg** can use the “training statistics” produced by **Random** setting, while **MV** can use the “testing statistics” produced by **Random** setting.

## 7.4 `MeasureIF` interface

`MeasureIF` represents a metrics for calculating algorithm performance. Concrete implementations of `MeasureIF` are:

- `AccuracyMeasure`,
- `F1Measure`,
- `Precision`
- `Recall.`

`MeasureIF` interface exposes the method `getMeasure` which takes the `ClassificationResult` object as an input parameter and returns a single real number that represents algorithm performance.

## 7.5 `SplitterIF` interface

`SplitterIF` interface represents a feature splitting method. Concrete implementations of `SplitterIF` are:

- `RandomSplit` (randomly divides the feature set in two views of roughly equal size) and
- `DifferentRandomSplitsSplitter` (using the `RandomSplit` class it generates multiple unique random feature splits).

`SplitterIF` interface exposes the method `splitDataset` which accepts the `CoTrainingData` object as one of its input parameters and applies the algorithm-specific feature split on the supplied data.

## 7.6 StartExperiment class

`StartExperiment` class is the starting point that orchestrates the execution of the program. The executon of the program is represented by an activity diagram in figure 40. It involves the following steps:

1. **Reading user-specified properties**. Values of the algorithm properties are loaded in several classes that follow the singleton design pattern in order to ensure that all parts of the system use the same parameter settings during the system life cycle. These classes are:

   - `DatasetSettings`: provides the dataset properties such as file locations, name of the class attribute, etc.;
   - `CVSettings`: provides settings needed for the creation of a new cross-validation experiment. For example, the number of folds, the number of randomly chosen labeled instances, etc.;
   - `CoTrainingSettings`: provides the settings needed for running the co-training algorithm. For example, the number of co-training iterations, the number of most confident instances labeled in each iteration, etc.;
   - `ExperimentSettings`: provides the specifics of the applied algorithm such as the concrete strategy classes that will be used during the execution (algorithm, splitting algorithm and performance measures);
   - `GASettings`: provides the parameters for the GA used for threshold optimization in RSSalg.

2. **(Optional) Setting the cross-validation experiment.** If the user specified that a new cross-validation experiment should be created, the `CrossValidationSeparator` class will be used to generate a new experiment.

3. **Running the experiment.** In each round of cross-validation the following steps are executed:
   (1) An appropriate `CoTrainingData` object (i.e. fold data) is loaded;
   (2) (Optional) a feature split is performed by the chosen feature splitter algorithm on `CoTraningData` object;
   (3) The `run` method of the chosen algorithm is executed using the `CoTraningData` object. The result is the `ClassificationResult` object and `Classifier` objects that store details about the training and testing process;
   (4) Performance measures are obtained based on the obtained `ClassificationResult` object;
   (5) (Optional) Train and test “classifier statistics” are saved on disk

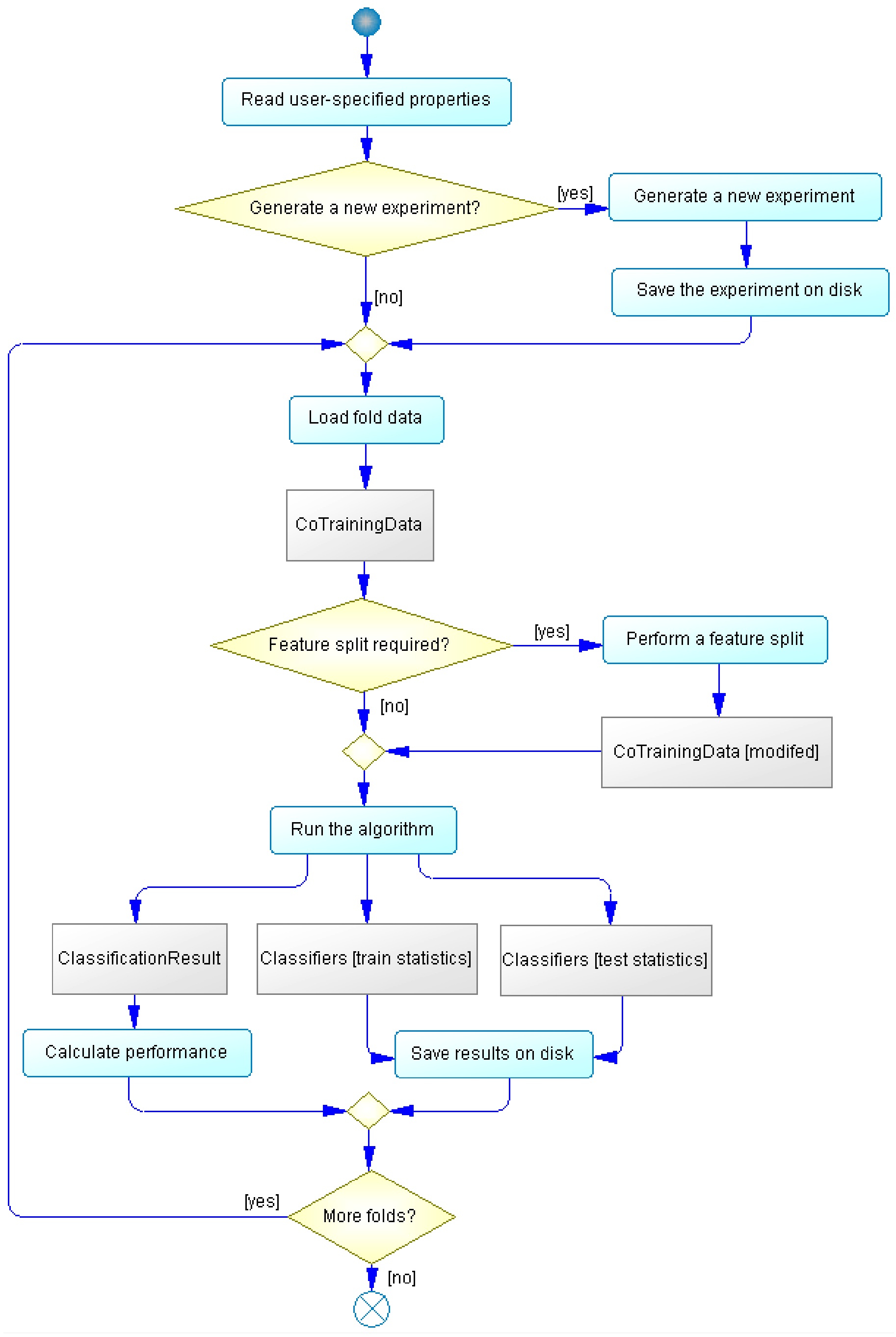


**Figure 40**. Activity diagram: running the experiment

## 7.7 RSSalg implementation

In this section we will display the details of RSSalg implementation. RSSalg is implemented in `RSSalg` class which inherits the abstract `Algorithm` class (figure 39). The execution flow of the `run` method implemented in `RSSalg` is represented by the activity diagram on figure 41.

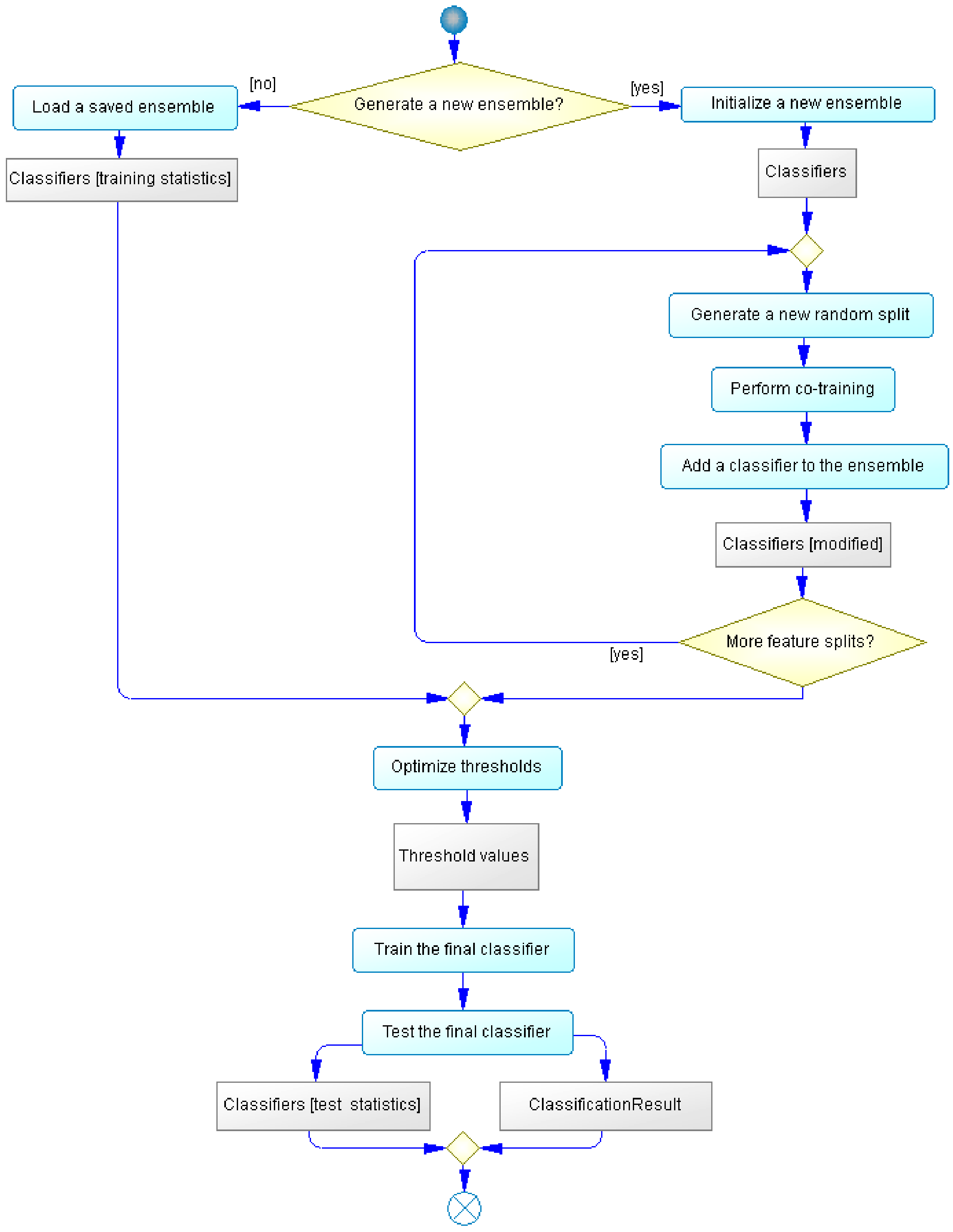

Generate a new ensemble?
[no]
[yes]
Load a saved ensemble
Initialize a new ensemble
Classifiers [training statistics]
Classifiers
Generate a new random split
Perform co-training
Add a classifier to the ensemble
Classifiers [modified]
More feature splits?
[yes]
Optimize thresholds
Threshold values
Train the final classifier
Test the final classifier
Classifiers [test statistics]
ClassificationResult

**Figure 41.** Main classes in RSSalg threshold optimization

When running the **RSSalg** setting, an already created ensemble can be loaded (for example, an ensemble produced earlier when running the **Random** setting). Otherwise, multiple unique random feature splits are created by applying the `DifferentRandomSplitsSplitter` class on the data supplied through `CoTrainingData` object. Each generated split is used to perform co-training (using `CoTraining` class implementation). The resulting training classifier statistics is recorded in the `Classifiers` object. Alternatively, user can choose to load a previously generated classifier statistics from a file.

After creating (or loading) an ensemble, the thresholds which determine the training set of the final model are determined by the GA optimization process. Using the labeled and unlabeled set supplied through the `CoTrainigData` object input parameter and the determined threshold values, training set for the final model is created. The final model is trained and evaluated (using the test data supplied through the `CoTrainigData` object parameter). The output of the algorithm is the `CoTrainingResult` object which can be used for performance measure evaluation and the `Classifier` object which stores the details of evaluating all classifiers from the ensemble.

The main classes involved in the threshold optimization process are presented in the class diagram on figure 42. The class `GAThresholdOptimizer` implements the GA threshold optimization process in RSSalg. Each threshold pair represents a candidate in GA optimization process (class `Candidate`). Each candidate is evaluated using the implementation of `CandidateEvaluatorIF` interface. The `evaluateCandidate` method in this interface takes as input parameters the `CoTrainingData`, `Classifiers` and `Candidate` objects as well as the implementation of `MeasureIF`: the concrete implementation of `CandidateEvaluatorIF` should, according to the candidate threshold pair and the recorded statistics adjust the labeled and test portion of `CoTrainingData` for final model training and testing; the final model will be evaluated using the provided implementation of `MeasureIF`.

There are two concrete implementations of `CandidateEvaluatorIF`:

- `RSSalgCandidatEvaluator`: used for running the threshold process described in section 2.2.1;
- `TestSetAccuracyCandidateEvaluator`: which uses the hand-labeled test data provided through `CoTrainingData` object. It is used for running the RSSalg$_{best}$ setting in [4].

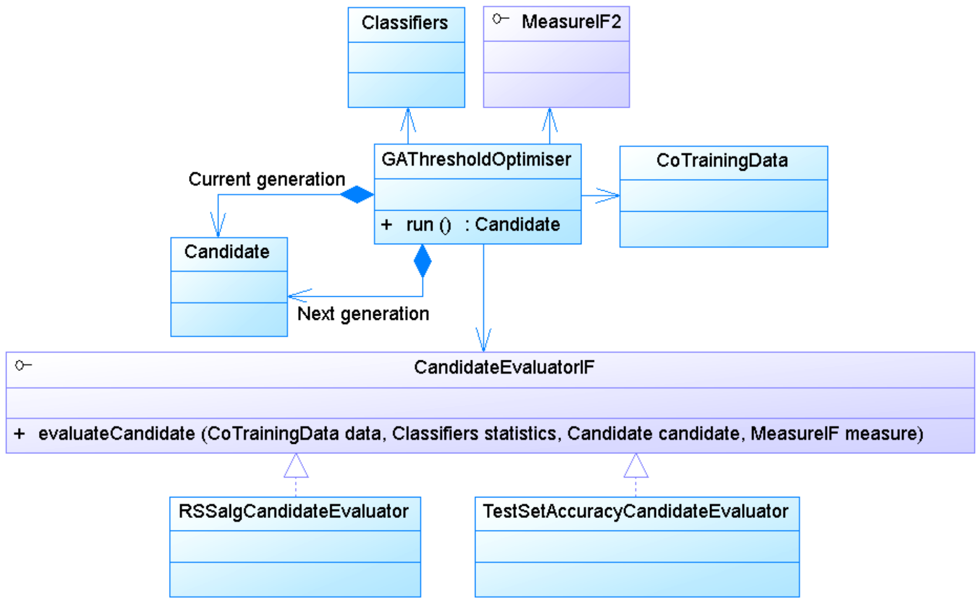


**Figure 42.** Main classes in RSSalg threshold optimization

## 7.8 *RSSalg software* implementation details

*RSSalg software* is implemented in Java and requires Java Runtime Environment 1.7 or higher in order to work. The implementation relies on Weka library version 3.7.0[31]:

(1) The datasets are internally represented using .ARFF file format and corresponding Weka's `Instances` class[32]. This is the commonly used representation widely accepted in the data mining community;

[31] In order to run the distributed executable (https://github.com/slivkaje/RSSalg-software/blob/v1.0/dist/RSSalg.jar), Weka does not have to be installed

[32] http://weka.sourceforge.net/doc.dev/weka/core/Instances.html

(2) Weka library provides implementations of individual view classifiers used in co-training. In *RSSalg software* any class implementing Weka's `Classifier` interface[33] can be used for underlying classifiers.

A disadvantage of *RSSalg software* is its rather slow performance due to relying on Weka data representation. A bottleneck of this representation is transferring instances between different feature sets. In the future we plan to address this issue by implementing a custom optimized version of the internal data representation.

## 7.9 Extending *RSSalg software*

One of the goals of *RSSalg software* design was to make it easily extendable in order to allow researchers easy experimenting with their own co-training based solutions. We strived to achieve this goal by using the strategy design pattern and exposing the interfaces that allow adding new implementations without changing the existing code. You can extend *RSSalg software* by adding a new algorithm (section 7.9.1), feature-splitting method (section 7.9.2), performance measure (section 7.9.3) or a new threshold optimization procedure for RSSalg (section 7.9.4).

### 7.9.1 Adding a new algorithm

We will explain how to add a new algorithm on the example of $\mathbf{L_{acc}}$ setting implementation given in figure 43. Each algorithm in *RSSalg software* inherits an abstract `Algorithm` class. In the inherited `run` method we implement all the steps needed for algorithm training and evaluation:

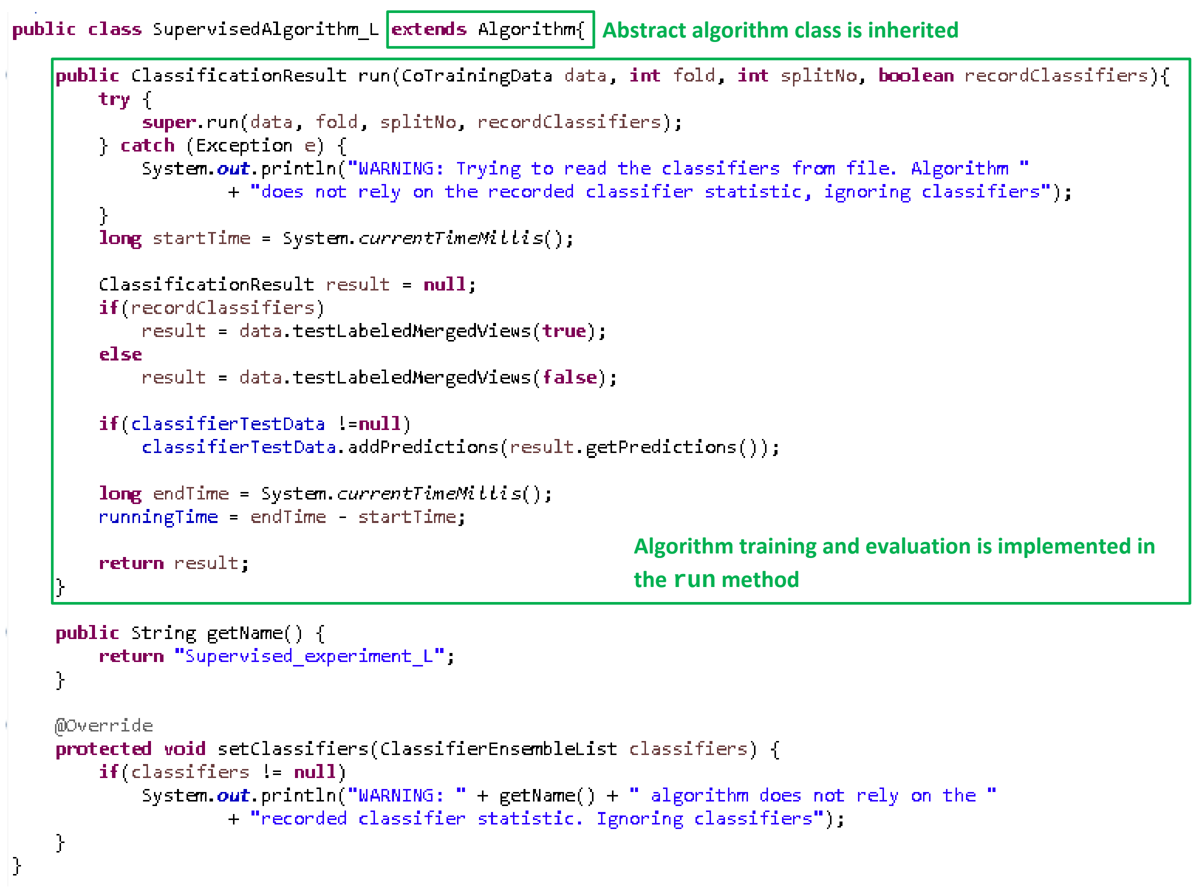
1. In the first try/catch block we make a call to the `run` method inherited from the `Algorithm` class. This method contains some implementation steps common for all algorithms, for example, (depending on user settings) it will initialize a new ensemble of classifiers or load an existing one.
2. After that we record the starting time. At the end of `run` method we also record the execution end time and determine the time needed for running the algorithm as the difference between these two values. The running time recorded in `runningTime` property of the `Algorithm` class is accessible via the call to `getRunningTimeMillis` method of `Algorithm` class.
3. The `run` method receives the dataset (divided in labeled, unlabeled and test data) as one of its input parameters: `data` object that belongs to `CoTrainingData` class. In $\mathbf{L_{acc}}$ setting our goal is to train a supervised classifier on the labeled portion of the data and evaluate it on the test portion of the data. This is already implemented in the `testLabeledMergedViews` method of the passed `data` object which returns the `ClassificationResult` object as a result.
4. The obtained `ClassificationResult` object is returned as a result of the `run` method. This object contains the information needed for calculating various performance measures.
5. As we have explained in section 4, we can record the details about training ("training statistics") and testing ("test statistics") of an algorithm. Since the "training statistics" refers to the training process of an SSL algorithm – labeling and transferring unlabeled instances to labeled data, we do not record this for $\mathbf{L_{acc}}$ which is a supervised setting. The "test statistic" contains the details about evaluation of each classifier from the ensemble and is represented by the `classifierTestData` object in figure 43.

Another method that should be implemented is the abstract `getName` method which should return the algorithm name as a `String`. The returned `String` is used for automatic naming of the XML files in which the obtained "training/test statistics" will be recorded.

Some algorithms may use previously recorded "training/test statistics", for example, **MV** uses the "test statistics" previously created by **Random** setting. The `Algorithm` class provides `SetClassifiers` method for setting the value of the used "statistics" and it may be overridden for algorithm-specific purposes. In figure 43, since $\mathbf{L_{acc}}$ setting does not use train or test "statistics", we have overridden the method to display the warning if the user tries to provide it.

Once an algorithm is implemented in a certain class, it can be used by simply specifying the full-package name of that class in the *algorithm* property located in experiment properties (section 6.3.4.7).

[33] http://weka.sourceforge.net/doc.dev/weka/classifiers/Classifier.html



```
public class SupervisedAlgorithm_L extends Algorithm{
    public ClassificationResult run(CoTrainingData data, int fold, int splitNo, boolean recordClassifiers){
        try {
            super.run(data, fold, splitNo, recordClassifiers);
        } catch (Exception e) {
            System.out.println("WARNING: Trying to read the classifiers from file. Algorithm "
                    + "does not rely on the recorded classifier statistic, ignoring classifiers");
        }
        long startTime = System.currentTimeMillis();

        ClassificationResult result = null;
        if(recordClassifiers)
            result = data.testLabeledMergedViews(true);
        else
            result = data.testLabeledMergedViews(false);

        if(classifierTestData !=null)
            classifierTestData.addPredictions(result.getPredictions());

        long endTime = System.currentTimeMillis();
        runningTime = endTime - startTime;

        return result;
    }

    public String getName() {
        return "Supervised_experiment_L";
    }

    @Override
    protected void setClassifiers(ClassifierEnsembleList classifiers) {
        if(classifiers != null)
            System.out.println("WARNING: " + getName() + " algorithm does not rely on the "
                    + "recorded classifier statistic. Ignoring classifiers");
    }
}
```

Abstract algorithm class is inherited
Algorithm training and evaluation is implemented in the run method

**Figure 43.** Implementation of $\mathbf{L_{acc}}$ (supervised learner trained on labeled portion of the data)

### 7.9.2 Adding a new feature-splitting method

An implementation of the feature-splitting method should be contained in the class implementing `SplitterIF` (section 7.5). This interface exposes two abstract methods: `getName` and `splitDatasets.`

The `getName` method returns a name of the feature-splitting algorithm as a `String`. The returned `String` is used for automatic naming of the XML files in which the obtained "training/test statistics" will be recorded.

The `splitDatasets` method should contain the implementation of the feature-splitting methodology. Its two most important input parameters are:

- `CotrainingData` object (section 7.1) that contains the dataset to perform the feature splitting method on. The `splitDatasets` method should modify this object so that the views correspond to the created feature split.
- `FeatureGraph` object represents the undirected weighted graph in which features represent vertices and the weight of an edge connecting two features represents some measure of dependence between those two features. For example, the `maxInd` algorithm proposed in [6] is a feature splitting methodology for co-training based on the assumption that the two views in co-training should be maximally independent given the class label. As the measure of class-conditional dependence of two features, authors propose to use the Conditional Mutual Information (*CondMI*) [16]. In this case we would construct the feature graph in which the weight of an edge connecting two features would be the *CondMI* measure calculated for these two features. In the implementation of random feature split this parameter is ignored as it does not depend on any inter-feature measure.

Once a feature-splitting methodology is implemented in a certain class, it can be used by simply specifying the full-package name of that class in the *featureSpliter* property located in experiment properties (section 6.3.4.7).

### 7.9.3 Adding a new performance measure

A class representing a certain performance measure should implement `MeasureIF` interface (section 7.4). `MeasureIF` exposes the following methods that a performance measure class should implement:

- `getName` – returns the performance measure name as Java `String`.
- `dependsOnClass` – This method should return false if the performance measure is not category-dependent (e.g. accuracy) and true if it is category-dependent (e.g. *f*1-measure).
- `setClassName` – This method is used to specify the category name for the calculation of category-dependent measures. The passed value may be the name of the certain category (for which we want to calculate the measure) or the `String` value "avg" (meaning that we want to calculate the measure averaged over all categories).
- `getClassName` – Returns the category value for which the measure is being calculated or the `String` "avg" if an averaged measure over all categories is being calculated.
- `getMeasure` – This measure accepts the `ClassificationResult` object as an input parameter. This object summarizes the results of evaluating a trained classifier on the set of test instances (the number of correctly and incorrectly labeled instances per each class and prediction confidences). Based on this object, in the `getMeasure` method, a double value representing the algorithm performance is calculated and returned as a result.

Once a performance measure is implemented in a certain class, it can be used by simply specifying the full-package name of that class in the *measures* property located in experiment properties (section 6.3.4.7).

### 7.9.4 Adding a new threshold optimization technique

As explained in section 2.2.1, RSSalg uses a GA for threshold optimization. A threshold pair is considered a candidate in the GA evaluation process. Setting certain threshold values will determine which portion of the most confidently labeled instances will be kept in the final training set. The open question is how can we evaluate the model trained on this final set. The classes implementing a certain model-evaluation methodology should implement the `CandidateEvaluatorIF` interface.

`CandidateEvaluatorIF` interface exposes two methods:

- `getName` – returns the name of the method as Java `String`.
- `evaluateCandidate` – the method in which the final model evaluation is performed. The input parameters of this method are: a `CoTrainingData` object which contains the instances of the dataset; `Classifiers` object which contains details about training the classifier ensemble; the implementation of the performance measure (`MeasureIF`) which defines whether a model should be evaluated by its accuracy, precision or some other measure; and the `Candidate` object which represents the candidate threshold pair which defines the instances that will be used for model training.

Once a final model evaluation procedure is implemented in a certain class, it can be used by simply specifying the full-package name of that class in the *candidateEvaluator* property located in experiment properties (section 6.3.4.7).

# 8 References

[1] Figueroa, R.L., Zeng-Treitler, Q., Kandula, S. and Ngo, L.H., 2012. Predicting sample size required for classification performance. BMC medical informatics and decision making, 12(1), p.1.

[2] Zhu, X., 2005. Semi-supervised learning literature survey.

[3] 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.

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

[6] 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.

[7] 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.

[8] D. Pierce and C. Cardie. Limitations of Co–Training for Natural Language Learning from Large Datasets. In Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP–2001), 2001.

[9] Wang, W. and Zhou, Z.H., 2013. Co-Training with Insufficient Views. In *ACML* (pp. 467-482).

[10] V. Ng and C. Cardie. Weakly supervised natural language learning without redundant views. *Proceedings of Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 94–101, Edmonton, USA, 2003

[11] Androutsopoulos, I., Koutsias, J., Chandrinos, K.V., Paliouras, G. and Spyropoulos, C.D., 2000. An evaluation of naive bayesian anti-spam filtering.arXiv preprint cs/0006013.

[12] T. Joachims: “A Statistical Learning Model of Text Classification for Support Vector Machines” Proc. ACM-SIGIR Int’l Conf. Research & Development in Information Retrieval, pp. 128-136, 2001

[13] Porter, M.F., 1980. An algorithm for suffix stripping. Program, 14(3), pp.130-137.

[14] Salton, G. and Buckley, C., 1988. Term-weighting approaches in automatic text retrieval. Information processing & management, 24(5), pp.513-523.

[15] J. Chan, I. Koprinska and J. Poon. Co–training with a single natural feature set applied to email classification. In Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence (WI’04), pages 586–589, Beijing, China 2004.

[16] MacKay, D.J., 2003. *Information theory, inference and learning algorithms*. Cambridge university press.