ssc: An R Package for Semi-Supervised Classification

by Mabel González, Osmani Rosado, José D. Rodríguez, Christoph Bergmeir, Isaac Triguero and José M. Benítez

Abstract Semi-supervised classification has become a popular area of machine learning, where both labeled and unlabeled data are used to train a classifier. This learning paradigm has obtained promising results, specifically in the presence of a reduced set of labeled examples. We present the R package ssc that implements a collection of self-labeled techniques to construct a classification model. This family of techniques enlarges the original labeled set using the most confident predictions to classify unlabeled data. The techniques implemented in the ssc package can be applied to classification problems in several domains by the specification of a suitable learning scheme. At low ratios of labeled data, it can be shown to perform better than classical supervised classifiers.

Introduction

Nowadays, the increasing amount of stored data from the most diverse domains makes data mining (Witten et al., 2011) a powerful tool to discover underlying knowledge on such data. A popular area of data mining focuses on the prediction of the label or class of new examples from data that describe what happened in the past. In those applications, an additional effort is required to obtain the labeled examples needed during the training process. Often, obtaining the labels is an expensive and time consuming process that requires the attention of the experts from a particular domain. Semi-supervised learning (SSL, Chapelle et al., 2006) can relieve this situation since it performs the training from a reduced number of labeled data in conjunction with abundant unlabeled data.

Semi-supervised classification (SSC) is in the middle ground between supervised and unsupervised classification. Several approaches have been proposed that follow the semi-supervised classification paradigm. The principal approaches are: generative models (Fujino et al., 2008), graph-based models (Blum and Chawla, 2001), and the semi-supervised support vector machines (Joachims, 1999). Every SSC approach makes its own assumptions (Zhu and Goldberg, 2009) about the link between the distribution of unlabeled and labeled data.

The taxonomy proposed in Triguero et al. (2015) describes another family of methods, denoted self-labeled techniques. The main target of this family is to enlarge the original labeled set using the most confident predictions to classify unlabeled data. In the specialized literature, there are reported several self-labeled methods (Li and Zhou, 2005; Wang et al., 2010; Zhou and Li, 2005; Zhou and Goldman, 2004). Some of the most popular self-labeled techniques are self-training (Yarowsky, 1995) and co-training (Blum and Mitchell, 1998). In general, those methods do not make any special assumptions about the distribution of the input data, but they accept that their most confident predictions tend to be correct.

This paper presents the ssc R package which implements successful self-labeled methods selected from the experimental analysis presented in Triguero et al. (2015). The methods start learning with a partially labeled dataset. A classification model is obtained as a result from the semi-supervised learning performed. The hypothesis learned by the model can be used to classify either the unlabeled instances provided during the training process or new instances. The R package implementing the self-labeled methods described in this paper is available from the Comprehensive R Archive Network (CRAN) at http://CRAN.R-project.org/package=ssc. Additionally, a web page with a usage tutorial and more usage examples is available at http://sci2s.ugr.es/dicits/software/ssc.

Semi-supervised Classification

The main idea of SSL is to learn from unlabeled as much as from labeled data to obtain more accurate models (Chapelle et al., 2006). In SSL, data can be separated into two sets: $L = \{x_1, \ldots, x_l\}$ with its known labels $Y_l = \{y_1, \ldots, y_l\}$, and $U = \{x_{l+1}, \ldots, x_{l+u}\}$ for which labels are unknown.

Depending on the main goal of these methods, SSL can be divided into *semi-supervised classification* (SSC) and *semi-supervised clustering*. The first focuses on improving the results obtained with supervised classification and the latter in finding better defined clusters (Zhu and Goldberg, 2009). This paper is focused on SSC.

SSC can be categorized into two slightly different settings, denoted *transductive* and *inductive learning*. Transductive learning concerns the problem of predicting the labels of the unlabeled

examples provided during the training phase. On the other hand, inductive learning considers the labeled and unlabeled data provided as the training examples, and its objective is to predict the label of unseen data (Chapelle et al., 2006).

Self-labeled methods

Self-labeled techniques (Triguero et al., 2015) obtain an enlarged labeled set by the iterative classification of unlabeled examples under the assumption that their most accurate predictions tend to be correct. Self-labeled techniques are typically divided into *self-training* and *co-training* methods.

Self-training is a simple and effective SSL methodology. During the self-training process the classifier is initially trained with a reduced set of labeled examples, aiming to classify unlabeled examples. Then, it is retrained with its own most confident predictions, enlarging its labeled training set. This process is repeated until a stopping criterion is reached. The major advantages of self-training are its simplicity and the fact that it is a wrapper method (Zhu and Goldberg, 2009).

The standard co-training (Blum and Mitchell, 1998) methodology assumes that the feature space can be split into two different conditionally independent views and that each view is able to predict the classes on its own. It trains one classifier in each specific view, and then the classifiers teach each other the most confidently predicted examples from the unlabeled pool. The process continues until a predefined number of iterations is reached.

The wrapper methodology used in both methods makes the selection of the learning scheme for the classifiers flexible. The only requirement is that the classifiers can assign a confidence score to their predictions, which could be used to select which unlabeled instances to turn into additional training data.

The **ssc** package implements six self-labeled methods. Table 1 describes these methods according to the properties given by Triguero et al. (2015):

Addition mechanism Describes the way in which the enlarged labeled set (EL) is formed. In incremental scheme, the algorithm starts with $EL = L$ and adds, step by step, the most confident instances of U. Another scheme is amending, which differs from incremental in that it can iteratively add or remove any instance that meets a certain criterion; this mechanism has been designed to avoid the introduction of noisy instances into EL at each iteration.

Classifiers This refers to whether it uses one or several classifiers during the enlarging phase of the labeled set. All of these methods follow a wrapper methodology using one or more classifiers to establish the possible class of unlabeled instances. In a single-classifier model, each unlabeled instance belongs to the most probable class assigned by the uniquely used classifier. Multi-classifier methods combine the learned hypotheses with several classifiers to predict the class of unlabeled instances.

Learning It specifies whether the models are constituted by the same (single) or different (multiple) learning algorithms. Multi-learning approaches are closely linked with multi-classifier models; a multi-learning method is itself a multi-classifier method in which the different classifiers come from different learning methods. On the other hand, a single-learning approach can be linked to both single and multi-classifiers.

Teaching In a mutual-teaching approach, the classifiers teach each other their most confident predicted examples. Each C_i classifier has its own EL_i which it uses for training at each stage. EL_i is increased with the most confident labeled examples obtained as the hypotheses combination of the remaining classifiers. By contrast, the self-teaching property refers to those classifiers that maintain a single EL.

Stopping criteria This is related to the mechanism used to stop the self-labeling process. It is an important factor due to the fact that it influences the size of EL and therefore the learned hypothesis. Some of the approaches for this are: (i) repeat the self-labeling process until a portion of U has been exhausted, (ii) establish a limited number of iterations, and (iii) the learned hypothesis remains stable between two consecutive stages.

Method	Reference	Addition mechanism	Classifiers	Learning paradigm	Teaching	Stopping criteria
Self-training	(Yarowsky, 1995)	incremental	single	single	self	i
SETRED	(Li and Zhou, 2005)	amending	single	single	self	i
SNNRCE	(Wang et al., 2010)	amending	single	single	self	i
Tri-training	(Zhou and Li, 2005)	incremental	multi	single	mutual	iii
Co-Bagging	(Blum and Mitchell, 1998)	incremental	multi	single	mutual	i
Democratic-Co	(Zhou and Goldman, 2004)	incremental	multi	multi	mutual	iii

Table 1: Methods implemented in ssc

Related packages

There are some publicly available R packages that allow semi-supervised classification. Most of them follow the generative paradigm. For instance, the **upclass** package (Russell et al., 2014) uses labeled and unlabeled data to construct a model-based classification method using a Gaussian mixture model. The Expectation Maximization algorithm is also used to obtain maximum likelihood estimates of the model parameters and classifications for the unlabeled data. The **bgmm** package (Biecek et al., 2012) implements partially supervised mixture modeling methods. **Rmixmod** (Lebret et al., 2015) is an exploratory data analysis tool for solving clustering and classification problems by fitting a mixture model to a given dataset. It can be used in semi-supervised situations where the dataset is partially labeled. The **spa** package (Culp, 2011) provides support for semi-supervised classification using graph-based estimation and linear regression.

To the best of our knowledge, there are no R packages that specialize in self-labeled methods. Recently, the **RSSL** (Krijthe, 2017) and **SSL** (Wang, 2016) packages were introduced but their implementations are mostly complementary to those offered in our package. Only the self-training standard model is implemented in these packages and in the **DMwR** package (Torgo, 2010) that covers a collection of data mining functions. On the other hand, **ssc** provides a far more extensive set of self-labeled methods, including various of the most successful approaches according to the extensive overview provided by Triguero et al. (2015).

Package functionalities

In this section we describe the main features of the **ssc** package. It is written in pure R and it provides implementations of the most relevant self-labeled models. Two basic functionalities have been implemented in the package: training a semi-supervised model from data and classification of instances using a trained model. Both functionalities are accessible through two different interfaces: specific and generic. The former is oriented to standard base classifiers and the latter is focused on base classifiers with more complex specifications. Figure 1 shows the main functions involved in both interfaces.

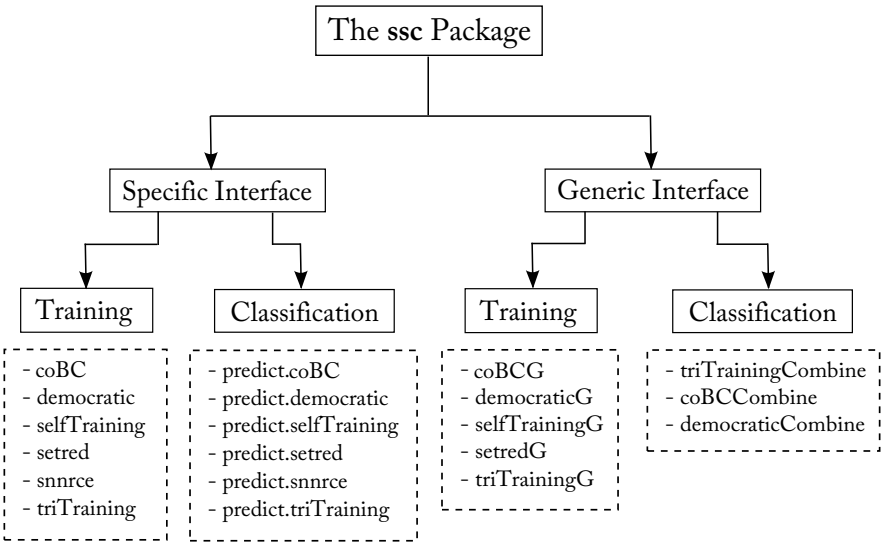

Figure 1: Main functionalities and their implementation in ssc.

The workflow followed to perform the classification task with the **ssc** package is illustrated in

Figure 2. In the training phase we use one of the available training functions in the ssc package. The training function takes as arguments the training set and other specified parameters of the selected model. In the classification phase we use the `predict` function for the specific interface. This function follows the *S3* class style and for that reason the classification process depends on the class of the trained model. In the case of the generic interface, the `predict` function used is the one that corresponds to the base model trained in the case of the single-classifier methods: `selfTraining` and `setred`. For the multi-classifier methods, we provide classification functions to combine the predicted values obtained from each individual base classifier.

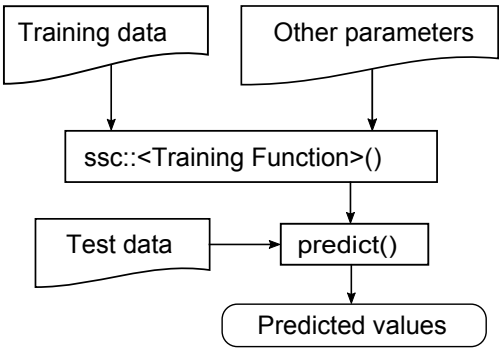

Figure 2: Training of models and prediction.

Training functions

For each semi-supervised method (described in Table 1), a function for training which returns the selected model is implemented. For both interfaces, the arguments are shown in Table 2.

Methods	Specific training interface	Generic training interface
coBC	`x, y, x.inst,` **learner, learner.pars, pred, pred.pars,** `N, perc.full, u, max.iter`	`y,` **gen.learner, gen.pred,** `N, perc.full, u, max.iter`
democratic	`x, y, x.inst,` **learners, learners.pars, preds, preds.pars**	`y,` **gen.learners, gen.preds**
selfTraining	`x, y, x.inst,` **learner, learner.pars, pred, pred.pars,** `perc.full, max.iter, thr.conf`	`y,` **gen.learner, gen.pred,** `perc.full, max.iter, thr.conf`
setred	`x, y, x.inst, dist,` **learner, learner.pars, pred, pred.pars,** `perc.full, max.iter, theta`	`y, D,` **gen.learner, gen.pred,** `perc.full, max.iter, theta`
snnrce	`x, y, x.inst, dist,` **alpha**	
triTraining	`x, y, x.inst,` **learner, learner.pars, pred, pred.pars,**	`y,` **gen.learner, gen.pred,**

Table 2: Input arguments for the training functions in ssc. The arguments defining the base classifiers are highlighted.

The `x`, `y` and `x.inst` arguments are mandatory for all training functions in the specific interface. The `x` argument provides the training instances in a usual matrix form (each row represents an instance). If the base classifier supports the training instances in other formats like a distance or kernel matrix then it is necessary to put as FALSE the argument `x.inst`. In this case, the `x` argument must be a squared matrix of dimension m, where m means the number of training instances. The other common argument `y` is a vector with the class information of the training instances. In this vector the unlabeled instances are specified with the value `NA`.

All training functions use at least one base classifier, following the wrapper methodology used in the SSC framework. In the case of the single-classifier methods: `selfTraining`, `setred`, `triTraining`, and `coBC`, the base classifier can be set using the arguments: `learner`, `learner.pars`, `pred` and `pred.pars`. The defined interface of the `learner` function is as follows:

```
base.class <- learner(x.train, y.train, learner.pars)
```

Here, `x.train` and `y.train` are the training set and `learner.pars` is a list of additional parameters that can be passed to the `learner` function. The returned value is a trained base classifier (the object class depends on the learner specified). The defined interface of the `pred` function is as follows:

```
y.test <- pred(base.class, x.test, pred.pars)
```

Here, `base.class` is the base classifier trained, `x.test` are the instances to predict and `pred.pars` is a list of additional parameters that can be passed to the `pred` function. The returned value is a matrix of class probabilities (one column for each class and one row for each instance in `x.test`).

For the generic interface, the base classifier definition is much more flexible. In this case, the manipulation of the instances occurs entirely inside the functions `gen.learner` and `gen.pred`. Therefore, the semi-supervised method accesses the training instances through indexes (index i refers to i_{th} instance supplied during the training phase). The defined interfaces of the `gen.learner` and `gen.pred` functions are as follows:

```
base.class <- gen.learner(indexes.train, y.train)
y.test <- gen.pred(base.class, indexes.test)
```

The argument `y` is common for all methods in both interfaces and the rest of arguments specifies particular features of the self-labeled techniques. For this reason these arguments appear either in the specific or the generic interface, with the exception of the `setred` method where the `dist` argument of the specific interface appears as argument `D` in the generic interface. In this case, the function `dist` is needed to compute the distance between all training instances and `D` is directly a distance matrix previously computed by the user.

On the other hand, the `snnrce` method uses a fixed base classifier (1NN) that cannot be modified. For that reason, we do not include a training function for this method in the generic interface. The function `democratic` has as arguments a list of learners and predict functions to specify the set of algorithms that will be used as base classifiers, following a multi-classifier approach.

Classification functions

From the training phase we obtain an object whose class depends on the trained model. This object keeps the information needed to perform both inductive and transductive classification. For the specific interface, the classification task is performed by the `predict` function. The main arguments of this function are described as follows:

- `object`: a semi-supervised model previously trained.
- `x`: a matrix with the description of the instances to be classified.

Following the *S3* class style, the `object` argument determines the function used to classify. The `x` argument depends on the training phase. If the model was trained from a distance (kernel) matrix then the expected value of `x` is a distance (kernel) matrix between the instances to be classified and the training instances included in the model. Otherwise, the model was trained from a matrix of instances and therefore the expected value of `x` is the matrix of instances to be classified. The `predict` function returns a factor with the classes predicted by the model.

In the case of the generic interface, the classification task is straightforward for the methods `selfTraining` and `setred`. To classify new instances, it is sufficient to use the `predict` function associated with the final base classifier obtained during the training phase. This base classifier is returned as the `model` attribute of the semi-supervised model trained. For the multi-classifiers methods, the `model` attribute contains a list of base classifiers instead of a single classifier. To classify new instances it is required to classify first those new instances with each base classifier, independently. Then, the final classification is obtained by combining the predictions of each classifier. For this task, we offer a dedicated combination function for each multi-classifier method:

- `coBCCombine`
- `democraticCombine`
- `triTrainingCombine`

Examples of usage

This section presents various examples that illustrate the main functionalities of the **ssc** package. We can install it from CRAN executing the following function in the R environment:

```
install.packages(ssc)
```

Setting up the data

Two example datasets have been included in the **ssc** package: wine (Lichman, 2013) and coffee (Chen et al., 2015). The first is the result of a chemical analysis of wines to determine the type of wine (three classes). The second dataset represents a binary classification problem that stems from the temporal domain. We illustrate the use of the functions in the ssc package using the wine dataset. We can obtain a partition that simulates the semi-supervised context with the following code:

```
library(ssc)
data(wine) # load the Wine dataset

cls <- which(colnames(wine) == "Wine")
x <- wine[, -cls] # instances without classes
y <- wine[, cls] # the classes
x <- scale(x) # scale the attributes for distance calculations
set.seed(3)

# Use 50% of instances for training
tra.idx <- sample(x = length(y), size = ceiling(length(y) * 0.5))
xtrain <- x[tra.idx,] # training instances
ytrain <- y[tra.idx] # classes of training instances

# Use 70% of train instances as unlabeled set
tra.na.idx <- sample(x = length(tra.idx),
                     size = ceiling(length(tra.idx) * 0.7))
ytrain[tra.na.idx] <- NA # remove class of unlabeled instances

# Use the other 50% of instances for inductive test
tst.idx <- setdiff(1:length(y), tra.idx)
xitest <- x[tst.idx,] # test instances
yitest <- y[tst.idx] # classes of instances in xitest

# Use the unlabeled examples for transductive test
xttest <- x[tra.idx[tra.na.idx],] # transductive test instances
yttest <- y[tra.idx[tra.na.idx]] # classes of instances in xttest
```

The training set **xtrain** includes 50% of all instances and the test set (**xitest**) contains the rest. In the **xtrain** set only the 30% of the instances are labeled. This information is included in the factor **ytrain** where the positions that have the value **NA** correspond to the unlabeled instances in **xtrain**. The labeled instances in **xtrain** are randomly selected with only one restriction: all classes must be represented by at least two instances.

The variables **xitest** and **xttest** are two matrices of instances stored row-wise that are used to test the prediction capabilities of the model. Specifically, **xitest** and **xttest** are used to test inductive and transductive prediction, respectively. In addition, the variables **yitest** and **yttest** correspond to the class information of the instances in **xitest** and **xttest**, respectively.

We compute the matrices required to use additional training options from a precomputed distance or kernel matrix, when the argument **x.inst = FALSE**. The following code computes the distance matrix using the Euclidean method implemented in the **proxy** package and the kernel matrix using the Gaussian radial basis function (RBF) with a fixed value of sigma.

```
# computing distance and kernel matrices
dtrain <- as.matrix(proxy::dist(x = xtrain, method = "euclidean", by_rows = TRUE))
ditest <- as.matrix(proxy::dist(x = xitest, y = xtrain, method = "euclidean",
                                by_rows = TRUE))
```

```
ktrain <- as.matrix(exp(- 0.048 * dtrain^2))
kitest <- as.matrix(exp(- 0.048 * ditest^2))
```

The matrices `dtrain` and `ktrain` are used in the training phase, and the matrices `ditest` and `kitest` are used in the inductive prediction phase. We highlight the order of the arguments `x` and `y` passed in the second call to the `dist` function. It is important to guarantee that `x` takes the test set and `y` takes the training set. The goal is to obtain a distance matrix with the following dimensions: the number of rows is equal to the size of the test set and the number of columns is equal to the size of the training set.

Training the model

We illustrate different ways of training a semi-supervised model depending on the base classifier specified and the option used for the `x.inst` argument. We include some examples of the `selfTraining` function to show the available options.

To perform the training phase using directly the instances in `xtrain` and the `knn3` function from the caret package as base classifier, we call the function as follows:

```
library(caret)
m.selft1 <- selfTraining(x = xtrain, y = ytrain, learner = knn3,
                         learner.pars = list(k = 1), pred = "predict")
```

Instead of using the instances in `xtrain` we can use a precomputed matrix in conjunction with a distance-based classifier. In this case, we use the distance matrix `dtrain` and the `oneNN` function available in the ssc package as follows:

```
m.selft2 <- selfTraining(x = dtrain, y = ytrain, x.inst = FALSE, learner = oneNN,
                         pred = "predict", pred.pars = list(type = "prob"))
```

The next example shows how to use the `selfTraining` function with a precomputed kernel matrix. In this case, the selected base classifier is a Support Vector Machines (SVM) implemented in the `ksvm` function from the kernlab package. In the argument `learner.pars` we need to specify the values of the arguments `kernel` and `prob.model` that will be provided to each call of the `ksvm` function. Furthermore, we define a wrapper for the original `predict` function of the "ksvm" object. Thus, we guarantee the selection of the columns that correspond to the support vectors obtained by the model `m`. Additionally, we coerce the matrix object `k` to "kernelMatrix" class before using the `predict` function.

```
library(kernlab)
m.selft3 <- selfTraining(x = ktrain, y = ytrain, x.inst = FALSE, learner = ksvm,
                         learner.pars = list(kernel = "matrix", prob.model = TRUE),
                         pred = function(m, k)
                           predict(m, as.kernelMatrix(k[, SVindex(m)]),
                                   type = "probabilities")
                         )
```

The training process with other methods in the ssc package is quite similar. In the next code snippet we train SETRED, SNNRCE, tri-training, and co-bagging models using the training instances in `xtrain`. For the `setred`, `triTraining`, and `CoBC` methods, we use the `ksvm` function as base classifier. The `snnrce` method has a fixed base classifier.

```
m.snnrce <- snnrce(x = xtrain, y = ytrain, dist = "Euclidean")

m.setred <- setred(x = xtrain, y = ytrain, dist = "Euclidean", learner = ksvm,
               learner.pars = list(prob.model = TRUE), pred = predict,
               pred.pars = list(type = "probabilities"))

m.trit <- triTraining(x = xtrain, y = ytrain, learner = ksvm,
               learner.pars = list(prob.model = TRUE), pred = predict,
               pred.pars = list(type = "probabilities"))
```

```
m.cobc <- coBC(x = xtrain, y = ytrain, N = 5, learner = ksvm,
               learner.pars = list(prob.model = TRUE), pred = predict,
               pred.pars = list(type = "probabilities"))
```

Training with Democratic-Co

In the ssc package, only the `democratic` method requires the specification of more than one base classifier. For that reason, the arguments `learners` and `preds` must be a list of functions instead of a single value. `democratic` assumes that the classifiers provided are from different learning paradigms. We show an example using three different base classifiers: 1NN, SVM and decision trees (implemented in the `C5.0` function from the C50 package). To perform the training process with `democratic`, we use the following code:

```
library(C50)
m.demo <- democratic(x = xtrain, y = ytrain, learners = list(knn3, ksvm, C5.0),
                     learners.pars = list(list(k=1), list(prob.model = TRUE), NULL),
                     preds = list(predict, predict, predict), preds.pars =
                       list(NULL, list(type = "probabilities"), list(type = "prob"))
)
```

In the next example, we show how to use the generic interface for the democratic-Co method. The target is to train from precomputed matrices two base classifiers: SVM and 1NN. The specific interface allows only a single precomputed matrix as argument. To obtain the functionality desired, we need to use the generic interface. At first, we define the learner and prediction functions for each base classifier according to the interfaces introduced in Section 2.2.1. The `tindexes` attribute incorporated in both trained models is used to specify the training instances included in the trained model. The last step is the call of the `democraticG` function:

```
l1nn <- function(indexes, cls){
  m <- oneNN(y = cls)
  attr(m, "tindexes") <- indexes
  m
}
l1nn.prob <- function(m, indexes)  {
  predict(m, dtrain[indexes, attr(m, "tindexes")], type = "prob")
}

lsvm <- function(indexes, cls){
  m = ksvm(ktrain[indexes, indexes], cls, kernel = "matrix", prob.model = TRUE)
  attr(m, "tindexes") <- indexes[SVindex(m)]
  m
}
lsvm.prob <- function(m, indexes)  {
  k <- as.kernelMatrix(ktrain[indexes, attr(m, "tindexes")])
  predict(m, k, type = "probabilities")
}

m.demoG <- democraticG(y = ytrain, gen.learners = list(l1nn, lsvm),
                       gen.preds = list(l1nn.prob, lsvm.prob))
```

Classifying seen and unseen instances

In the following we explain how to classify new instances. We illustrate this with various examples. The models used in the following examples were trained previously in Section 2.3.2.

In the first example we use the model `m.selft1` to perform inductive classification. Because this model was trained using an instance matrix, we need the instance matrix `xitest` to classify new instances. We predict the classes of the test instances with the following code:

```
p.selft1 <- predict(m.selft1, xitest)
```

Now, we use the models `m.selft2` and `m.selft3` that were trained using precomputed distance and kernel matrices, respectively. Therefore, we provide the precomputed test matrices (`ditest`

and `kitest`) to perform inductive classification. The classification obtained is stored in the vector `p.selft1`.

```
p.selft2 <- predict(m.selft2, ditest[, m.selft2$instances.index])
p.selft3 <- predict(m.selft3, as.kernelMatrix(kitest[, m.selft3$instances.index]))
```

The internal attribute `instances.index` in the objects `m.selft2` and `m.selft3` stores the indexes of the training instances used in the built model. During the training phase, the learning function selects the instances that will be included in the returned model. According to this, for each precomputed matrix we select the sub matrix corresponding to the unseen test instances and the selected training instances.

On the other hand, we illustrate with the `m.selft3` model how to perform transductive classification. Here, to predict the classes of the unlabeled training instances (referenced by the `tra.na.idx` variable) we pass directly the matrix `ktrain`, used during the training phase:

```
p.selft3transd <- predict(m.selft3, as.kernelMatrix(ktrain[tra.na.idx,
                                                    m.selft3$instances.index]))
```

For the rest of the single classifier models, we perform inductive classification of the test instances provided in the matrix `xitest`.

```
p.snnrce <- predict(m.snnrce, xitest)
p.setred <- predict(m.setred, xitest)
p.trit <- predict(m.trit, xitest)
p.cobc <- predict(m.cobc, xitest)
```

Classifying with Democratic-Co

For the specific interface, the classification task using the `democratic` function is similar to the previous examples. We predict the classes of the test instances as follows:

```
p.demo <- predict(m.demo, xitest)
```

However, this task using the generic interface requires a previous step, consisting in the prediction of the test instances by each base classifier contained in the ensemble. Subsequently, we use the `democraticCombine` function to create the final hypotheses.

```
m1.pred1 <- predict(m.demoG$model[[1]], ditest[, m.demoG$model.index[[1]]],
                    type ="class")
m1.pred2 <- predict(m.demoG$model[[2]],
                    as.kernelMatrix(kitest[, m.demoG$model.index[[2]]]))

p.demoG <- democraticCombine(pred = list(m1.pred1, m1.pred2), m.demoG$W,
                             m.demoG$classes)
```

Comparison between the models trained

In this example we perform a comparison between a selection of the trained models to determine the most competitive one for the wine classification problem.

```
p <- list(p.selft3, p.snnrce, p.setred, p.trit, p.cobc, p.demo)
acc <- sapply(X = p, FUN = function(i) {caret::confusionMatrix(table(i,
                                                    yitest))$overall[1]})

names(acc) <- c("SelfT","SNNRCE","SETRED","TriT", "coBC","Demo")
barplot(acc, beside = T, ylim = c(0.80,1), xpd = FALSE, las = 2,
        col=rainbow(n = 6, start = 3/6, end = 4/6, alpha = 0.6) ,
        ylab = "Accuracy")
```

The bar plot generated with the evaluation is shown in Figure 3. Tri-training obtains the most accurate results for the wine problem.

Another useful analysis is the comparison with the supervised paradigm. For this we train a supervised classifier (for simplicity SVM) to obtain a baseline of the classification results. The SVM classifier trained from the initial labeled instances in `xtrain` can be used as a lower bound of accuracy. We evaluate the supervised classifier in the test set `xitest` and compare this result with the semi-supervised performance. In the following code we train and evaluate the SVM classifier:

```
labeled.idx <- which(!is.na(ytrain))# indices of the initially labeled instances
xilabeled <- xtrain[labeled.idx,] # labeled instances
yilabeled <- ytrain[labeled.idx] # related classes

svmBL <- ksvm(x = xilabeled, y = yilabeled, prob.model = TRUE) # build SVM
p.svmBL <- predict(object = svmBL, newdata = xitest) # classify with SVM

abline(h = caret::confusionMatrix(table(p.svmBL, yitest))$overall[1], col = "red",
       lwd = 2)
legend(x = 2, y = 1.0, col = c("red"), legend=c("Base line"), lty = 1, lwd = 2)
```

The baseline generated is shown in Figure 3. Most self-labeled methods obtain an accuracy gain by taking into account the unlabeled instances during the training. In particular, `triTraining` obtains an accuracy gain of 0.05.

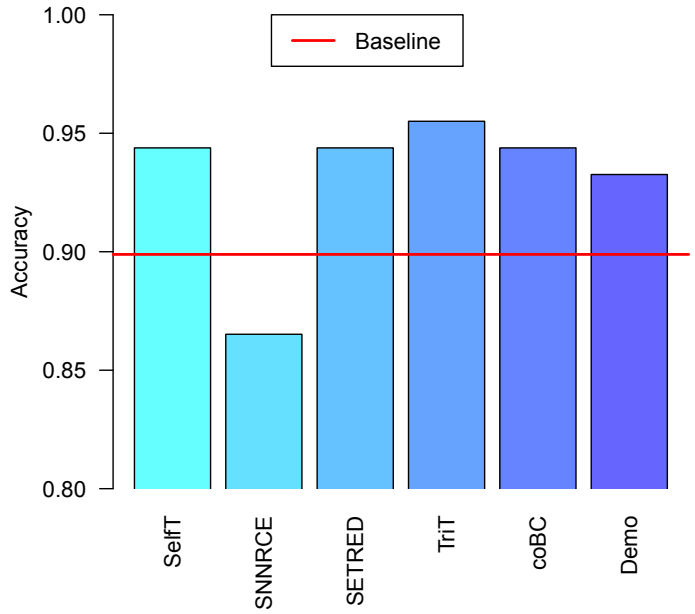

Figure 3: Comparison between various semi-supervised models evaluated for the wine problem.

Empirical evaluation of performance

In this section, we illustrate the performance of some methods implemented in the ssc package. We show the comparison between the baseline and the semi-supervised accuracy results applied to five datasets taken from the UCI repository. The `SVM` with the RBF kernel function is used as base classifier and benchmark supervised classifier in all comparisons. The semi-supervised methods evaluated are: `selfTraining`, `setred`, `coBC` and `triTraining`.

In the preparation process, we follow the same procedure used in Section 2.3.1 to split the wine dataset: 50% of the instances to train ($L \cup U$) and 50% of the instances to test (T). The set L represents 30% of the training instances. To train the supervised method we use only the avaliable instances from L. To test all methods we use the set T.

The results of our experiment are shown in Table 3. All results that represent an accuracy gain in the semi-supervised paradigm are printed in a boldface font. The results show that, in general,

Datasets	SVM	selfTraining	setred	coBC	triTraining
Iris	0.68	**0.88**	**0.88**	**0.88**	**0.90**
Parkinsons	0.86	0.86	**0.87**	**0.87**	0.86
Wine	0.95	**0.97**	**0.97**	0.95	**0.97**
Vertebral column	0.77	0.75	0.77	**0.78**	**0.78**
Fertility	0.90	0.90	0.90	0.72	0.90

Table 3: Accuracy classification results.

self-labeled techniques show competitive results to face classification problems from diverse domains. Specifically, in the presence of a reduced set of labeled examples.

Conclusions

We have presented the R package ssc which provides a collection of self-labeled techniques to deal with the semi-supervised classification problem that occurs in multiple domains. The implemented techniques can take advantage of partially labeled datasets during the training phase to create a classifier. The classifiers obtained can be used to perform either transductive or inductive classification. The ssc package offers a wrapper framework to train models. Depending on the base classifier selected, the models can be trained from instances or directly from a precomputed distance or kernel matrix. In addition, the ssc package supports a generic interface for base classifiers with other specifications, increasing the flexibility of this approach. We have shown in the experimental results that these techniques can provide better results than supervised classification at low ratios of labeled data.

Acknowledgments

This work was supported in part by "Proyecto de Investigación de Excelencia de la Junta de Andalucía, P12-TIC-2958" and "Proyecto de Investigación del Ministerio de Economía y Competitividad, TIN2013-47210-P". This work was partly performed while M. González held a travel grant from the Asociación Iberoamericana de Postgrado (AUIP), supported by Junta de Andalucía, to undertake a research stay at University of Granada.

Bibliography

P. Biecek, E. Szczurek, M. Vingron, and J. Tiuryn. The R package bgmm: Mixture modeling with uncertain knowledge. *Journal of Statistical Software*, 47(3):1–32, 2012. [p3]

A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *18th International Conference on Machine Learning*, 2001. [p1]

A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 92–100, New York, NY, USA, 1998. ACM. ISBN 1-58113-057-0. doi: 10.1145/279943.279962. [p1, 2, 3]

O. Chapelle, B. Schölkopf, A. Zien, et al., editors. *Semi-supervised learning*. MIT press Cambridge, 2006. [p1, 2]

Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The ucr time series classification archive, July 2015. www.cs.ucr.edu/~eamonn/time_series_data/. [p6]

M. Culp. spa: Semi-supervised semi-parametric graph-based estimation in r. *Journal of Statistical Software*, 40(1):1–29, 2011. ISSN 1548-7660. doi: 10.18637/jss.v040.i10. [p3]

A. Fujino, N. Ueda, and K. Saito. Semisupervised learning for a hybrid generative/discriminative classifier based on the maximum entropy principle. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(3):424–437, March 2008. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.70710. [p1]

T. Joachims. Transductive inference for text classification using support vector machines. In *16th International Conference on Machine Learning*, ICML '99, pages 200–209, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. [p1]

J. H. Krijthe. Rssl: Semi-supervised learning in r. In B. Kerautret, M. Colom, and P. Monasse, editors, *Reproducible Research in Pattern Recognition*, pages 104–115, Cham, 2017. Springer International Publishing. ISBN 978-3-319-56414-2. [p3]

R. Lebret, S. Iovleff, F. Langrognet, C. Biernacki, G. Celeux, and G. Govaert. Rmixmod: The r package of the model-based unsupervised, supervised, and semi-supervised classification mixmod library. *Journal of Statistical Software*, 67(1):1–29, 2015. ISSN 1548-7660. doi: 10.18637/jss.v067.i06. [p3]

M. Li and Z. Zhou. Setred: Self-training with editing. In *Advances in Knowledge Discovery and Data Mining*, volume 3518 of *Lecture Notes in Computer Science*, pages 611–621. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-26076-9. doi: 10.1007/11430919_71. [p1, 3]

M. Lichman. UCI machine learning repository, 2013. http://archive.ics.uci.edu/ml. [p6]

N. Russell, L. Cribbin, and T. B. Murphy. *upclass: Updated Classification Methods using Unlabeled Data*, 2014. URL http://CRAN.R-project.org/package=upclass. R package version 2.0. [p3]

L. Torgo. *Data Mining with R, learning with case studies*. Chapman and Hall/CRC, 2010. [p3]

I. Triguero, S. García, and F. Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*, 42(2):245–284, 2015. ISSN 0219-1377. doi: 10.1007/s10115-013-0706-y. [p1, 2, 3]

J. Wang. *SSL: Semi-Supervised Learning*, 2016. URL https://CRAN.R-project.org/package=SSL. R package version 0.1. [p3]

Y. Wang, X. Xu, H. Zhao, and Z. Hua. Semi-supervised learning based on nearest neighbor rule and cut edges. *Knowledge-Based Systems*, 23(6):547–554, 2010. ISSN 0950-7051. doi: http://dx.doi.org/10.1016/j.knosys.2010.03.012. [p1, 3]

I. H. Witten, E. Frank, and M. A. Hall. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, Boston, third edition edition, 2011. ISBN 978-0-12-374856-0. doi: http://dx.doi.org/10.1016/B978-0-12-374856-0.00018-3. [p1]

D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 189–196. Association for Computational Linguistics, 1995. [p1, 3]

Y. Zhou and S. Goldman. Democratic co-learning. In *IEEE 16th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 594–602. IEEE, Nov 2004. doi: 10.1109/ICTAI.2004.48. [p1, 3]

Z. Zhou and M. Li. Tri-training: exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1529–1541, Nov 2005. ISSN 1041-4347. doi: 10.1109/TKDE.2005.186. [p1, 3]

X. Zhu and A. B. Goldberg. *Introduction to Semi-Supervised Learning*. Morgan & Claypool Publishers, 2009. [p1, 2]

Mabel González
Department of Computer Science, Universidad Central "Marta Abreu" de Las Villas
Camajuaní road Km. 5 y 1/2, Santa Clara 50100
Cuba
ORCiD 0000-0003-0152-444X
mabelc@correo.ugr.es

Osmani Rosado
Department of Computer Science, Universidad Central "Marta Abreu" de Las Villas
Camajuaní road Km. 5 y 1/2, Santa Clara 50100
Cuba
ORCiD 0000-0002-2639-3354
osmanir@uclv.cu

José D. Rodríguez
Department of Computer Science, Universidad Central "Marta Abreu" de Las Villas
Camajuaní road Km. 5 y 1/2, Santa Clara 50100
Cuba
ORCiD 0000-0002-8489-4106
josedaniel@uclv.cu

Christoph Bergmeir
Faculty of Information Technology, Monash University, Melbourne
P.O. Box 63 Monash University, Victoria 3800
Australia
ORCiD 0000-0002-3665-9021
christoph.bergmeir@monash.edu

Isaac Triguero
School of Computer Science, University of Nottingham
Jubilee Campus, Wollaton Road, Nottingham NG8 1BB
United Kingdom
ORCiD 0000-0002-0150-0651
isaac.triguero@nottingham.ac.uk

José M. Benítez
Department of Computer Science and Artificial Intelligence, University of Granada
C/ Periodista Daniel Saucedo Aranda s/n, 18071, Granada
Spain
ORCiD 0000-0002-2346-0793
j.m.benitez@decsai.ugr.es