2017-04-01

RSSalg software: A tool for flexible experimenting with co-training based semi-supervised algorithms

Slivka Jelena, Sladić Goran, Milosavljević Branko, Kovačević Aleksandar

Knowledge-Based Systems 121 (2017) 4–6

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

RSSalg software: A tool for flexible experimenting with co-training based semi-supervised algorithms

J. Slivka, G. Sladić, B. Milosavljević, A. Kovačević*

University of Novi Sad/Faculty of Technical Sciences/Computing and Control Department, Novi Sad, Serbia

ARTICLE INFO

Article history:
Received 15 July 2016
Revised 24 December 2016
Accepted 18 January 2017
Available online 21 January 2017

Keywords:
Semi-supervised learning
Co-training
Ensemble methods
Java
Weka

ABSTRACT

RSSalg software is a tool for experimenting with Semi-Supervised Learning (SSL), a set of machine learning techniques able to use both labeled and unlabeled data for training. The goal is to reduce human effort regarding data labeling while preserving model quality.

RSSalg software encompasses the implementation of co-training, a multi-view SSL technique and RSSalg, its single-view alternative. Our tool enables easy comparison of different SSL algorithms. It provides a cross-validation procedure and supports standard metrics for performance evaluation. The tool is free and open source, available on GitHub under the GNU General Public License. It is implemented in Java language using Weka library.

1. Introduction

When training a prediction model we are often faced with a problem of lacking labeled training data. Data labeling requires manual human work which can be expensive regarding both time and money. This issue can be alleviated by Semi-Supervised Learning (SSL) techniques which can use both labeled and unlabeled data for training, with the goal of minimizing human effort regarding data labeling while keeping the desired level of prediction quality [1].

In this paper, we present *RSSalg software*, a tool for experimenting with SSL classification techniques. Our tool encompasses the implementation of two SSL methods: co-training [2] and RSSalg [3]. It provides a cross-validation design for comparing the performance of different SSL algorithms and ensures experiment replicability.

Co-training [2] is a very powerful SSL technique which can be applied on the datasets characterized by a natural separation of features in two disjoint sets (feature split), called views. Ideally, the feature split originates from the existence of two independent sources of information describing the same data. Each view is represented by its unique set of features. In *RSSalg software*, the user can define the natural feature split and run co-training.

Co-training performance greatly depends on the properties of the used feature split [4]. In many real-world problems that would be alleviated by co-training, an adequate feature split might be unknown. Researchers have approached this issue in two general directions[1]:

(1) Designing the optimal artificial feature split for co-training. In *RSSalg software* user can run co-training using a random feature split which proved to be useful when there is enough redundancy in the data [5]
(2) Combining co-training with ensemble methods. RSSalg [3] is one of these techniques.

To the best of our knowledge, a methodology that could guarantee successful co-training application on single-view datasets of various properties remains an open research problem. Software that allows easy implementation of new co-training solutions would be beneficial to the research community. Thus, *RSSalg software* was designed to be easily extendable.

Another open research question in co-training is the lack of principal way of choosing its parameters [6]. Our tool is maximally configurable regarding algorithm parameters allowing the user to explore the relationship between parameter values and algorithm performance.

2. Tool functionalities

In *RSSalg software* user can train and evaluate the following classifiers:

* Corresponding author.
E-mail addresses: slivkaje@uns.ac.rs (J. Slivka), sladicg@uns.ac.rs (G. Sladić), mbranko@uns.ac.rs (B. Milosavljević), kocha78@uns.ac.rs (A. Kovačević).

http://dx.doi.org/10.1016/j.knosys.2017.01.024
0950-7051/

[1] A more detailed survey of the proposed solutions can be found in [3].

- $\mathbf{L_{acc}}$: Supervised learner trained using the labeled instances
- $\mathbf{All_{acc}}$: Supervised learner trained using both labeled and unlabeled instances (with correct labels assigned)
- **Natural**: co-training run with natural (user-provided) feature split
- **Random**: co-training run with random feature split. An average performance of m co-training classifiers trained using m different random splits is reported
- **MajorityVote** (**MV**): a majority vote of m co-training classifiers trained in **Random**
- **RSSalg**: RSSalg [3], an ensemble technique that uses m co-training classifiers trained in **Random** with a more sophisticated vote aggregation procedure than **MV**
- $\mathbf{RSSalg_{best}}$: RSSalg with parameters optimized on test data ($\mathrm{RSSalg_{best}}$ in [3])

RSSalg software encompasses the k-fold-cross evaluation procedure: the dataset is divided in k stratified folds; in each iteration of the procedure:

- a different fold f is used for random selection of $c_1 + c_2 + \ldots + c_l$ labeled instances as labeled data L, where c_i denotes the number of instances per category i and l denotes the number of categories;
- the remaining data from fold f as well as j ($j < k$) adjacent folds are used as unlabeled data U (the label information for these instances is disregarded);
- 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 of L and U, as well as the label distribution in L. Our software supports the following standard metrics for calculating algorithm performance: accuracy, precision, recall, and $f1$-measure.

The user manual hosted in the code repository[2] gives a detailed explanation on how to use *RSSalg software* to test the following general traits an SSL algorithm should have[3]:

T1 Is the algorithm able to improve the performance by using (only) additional unlabeled data? If so, its performance should be greater than the performance of $\mathbf{L_{acc}}$.

T2 Does the algorithm reduce the number of labeled instances needed for learning while maintaining model quality? If so, its performance should be close to $\mathbf{All_{acc}}$ performance.

T3 Is the good performance of the algorithm guaranteed? SSL techniques impose certain assumptions about the data and can yield poor performance if they are not met [1]. Thus, it is important to identify the traits a dataset should have for us to guarantee the good performance of an algorithm. In the experiments presented in the user manual, we evaluate the performance of algorithms implemented in our tool on the set of datasets with different properties (size, dimensionality, and feature redundancy).

T4 Is there a principal way of establishing the values of algorithm parameters? In the experiments performed in the user manual, we explore the influence of the number of feature splits m and the choice of underlying classification models used in co-training on the performance of algorithms implemented in our tool.

T5 Does an ensemble perform better than its individual classifiers on average? Our tool allows us to run **RSSalg** and **MV** on the same m classifiers generated in **Random**. We hypothesize that both **RSSalg** and **MV** will perform better than **Random**.

T6 Is the procedure of aggregating votes in RSSalg better than a simple majority vote? If so, **RSSalg** will perform better than **MV**.

T7 The voting procedure in **RSSalg** relies on underlying parameters which are automatically tuned using unlabeled data. How good is this hypothesis, i.e. is it close to the results we would obtain if we had additional labeled data to optimize these parameters? If so, the performance of **RSSalg** will be close to the performance of $\mathbf{RSSalg_{best}}$.

The key findings we drew from the performed empirical evaluation are:

- In all test cases, $\mathbf{RSSalg_{best}}$ was the best performing SSL classifier. While it is not a realistic setting (it utilizes test data labels for determination of parameter values) it shows the potential of **RSSalg** [3].
- **RSSalg** is the best performing setting when using SVM as an underlying classifier in co-training, while **MV** is the best choice when using RBF net classifier. When using Naive Bayes, **RSSalg** yields somewhat better performance on datasets with higher feature redundancy, and it is comparable to MV on the rest of the datasets.
- **Natural** performs worse than **RSSalg** and **MV**. It is best used with RBF net.
- **Random** was most successful on more redundant datasets with Naive Bayes.
- **MV, RSSalg**, and $\mathbf{RSSalg_{best}}$ are robust to the value of m, even though the performance of **Random** varies significantly. Their performance grows as m grows, however, after a certain point adding more co-training classifiers makes little difference.

3. Benefits of the tool

RSSalg software is designed to have the following features:

- **Maximally configurable**. The user can control all algorithm parameters and the experiment generation procedure without changing the current code.
- **Fully reproducible experiments**. The software allows the user to control the random number generator seed which ensures that the same sequence of random numbers is applied to all processes that contain randomness.
- **Easily extendable**. *RSSalg software* is designed to be easily extendable by the usage of strategy design pattern and exposure of interfaces that allow adding new algorithms, feature-splitting methods, performance measures and parameter optimization techniques for RSSalg.
- **Cross-platform**. It is implemented in Java and has no outside dependencies.
- **Easy to install and run**. We offer the software in the form of an executable JAR file with the only prerequisite being the installation of Java. We also provide an Apache Ant script for building the project from the source code and running it.
- **User-friendly**. Our tool offers the GUI that allows easy experiment specification.
- **Well documented**. Along with this manuscript and accompanying user manual, we also provide ready-to-run reproducible experiments which demonstrate the software capabilities. Javadoc documentation is also available in the code repository.

4. Other available co-training implementations

Authors of [7] offer the implementation of self-learning and co-training in the form of an executable JAR. There are several open-source implementations of co-training variants available in MATLAB [8,9] and Java [10]. The open-source Java implementations of

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

[3] These ready-to-run experiments are hosted in the software code repository https://github.com/slivkaje/RSSalg-software/tree/master/data.

basic co-training [2] can be found in [10,11]. These implementations are extremely valuable to the research community. However, they are not open-source [7], offer only the variants of co-training without its basic implementation [8,9], their implementations relies on non-free software [8,9], or they are not designed to be extendable [7–9,11]. *RSSalg software* is an open-source tool implemented in Java that encompasses basic co-training alongside few of its single-view variants, the fold-cross experimental design, multiple performance measures and multiple interfaces for the program extension. It also offers the configuration of algorithm parameters (without changing the code), ensures the result replicability, has a user-friendly GUI and complete documentation.

5. Conclusions

We have presented a free and open-source tool *RSSalg software*, intended for experimenting with SSL techniques. It encompasses the implementation of co-training and several of its single-view alternatives and has a built-in validation procedure for algorithm evaluation. We have identified important criteria an SSL algorithm should meet and explained how *RSSalg software* could be used to evaluate them.

Our tool offers many benefits for researchers that wish to implement and test their SSL solutions: it is easily extensible and configurable, and all experiments are fully reproducible. It is platform independent and easy to install and use. It is accompanied by a user manual which includes usage examples, a step-by-step tutorial on preparing and running the experiments, a detailed overview of the software architecture and instructions on how to extend the software for custom purposes.

Acknowledgments

Results presented in this paper are a part of the research conducted within the Grant No. III-47003 provided by the Ministry of Education and Science of the Republic of Serbia

Appendix A. Required metadata

Current executable software version

Table 1
Software metadata (optional).

Nr.	(executable) Software metadata description	Please fill in this column
S1	Current software version	v1.0
S2	Permanent link to executables of this version	https://github.com/slivkaje/RSSalg-software/releases/tag/v1.0.1
S3	Legal Software License	GNU General Public License
S4	Computing platform/Operating System	Portable and platform independent
S5	Installation requirements & dependencies	Java Runtime Environment 1.7 or higher
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	https://github.com/slivkaje/RSSalg-software/tree/master/docs
S7	Support email for questions	slivkaje@uns.ac.rs

Appendix B. Current code version

Table 2
Code metadata (mandatory).

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.0
C2	Permanent link to code/repository used of this code version	https://github.com/slivkaje/RSSalg-software/releases/tag/v1.0.1
C3	Legal Code License	GNU General Public License
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Java 1.7
C6	Compilation requirements, operating environments & dependencies	JRE 1.7 (or higher) and weka.jar 3.7.0
C7	If available Link to developer documentation/manual	https://github.com/slivkaje/RSSalg-software/tree/master/docs
C8	Support email for questions	slivkaje@uns.ac.rs

References

[1] X. Zhu, 2005, Semi-supervised learning literature survey.
[2] A. Blum, T. Mitchell, Combining labeled and unlabeled data with co-training, in: Proceedings of the eleventh annual conference on Computational learning theory, ACM, 1998, pp. 92–100.
[3] J. Slivka, A. Kovačević, Z. Konjović, Combining co-training with ensemble learning for application on single-view natural language datasets, Acta Polytechnica Hungarica 10 (2) (2013).
[4] D. Pierce, C. Cardie, Limitations of co-training for natural language learning from large datasets, in: Proc. of the 2001 Conf. on Empirical Methods in Natural Language Processing, 2001, pp. 1–9.
[5] K. Nigam, R. Ghani, Understanding the behavior of co-training, in: Proceedings of KDD-2000 Workshop on Text Mining, 2000, pp. 15–17.
[6] V. Ng, C. Cardie, Weakly supervised natural language learning without redundant views, in: Proc. of the 2003 Conf. of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1, Association for Computational Linguistics, 2003, pp. 94–101.
[7] R. Ibrahim, N.A. Yousri, M.A. Ismail, N.M. El-Makky, miRNA and gene expression based cancer classification using self-learning and co-training approaches, in: Bioinformatics and Biomedicine (BIBM), 2013 IEEE Int'l Conf. on (pp. 495–498), 2013. Code is available at https://sourceforge.net/projects/selflearningcotraining/?source=directory.
[8] M.L. Zhang, Z.H. Zhou, Cotrade: confident co-training with data editing, IEEE Trans. Syst. Man Cybern. Part B (Cybernetics) 41 (6) (2011) 1612–1626. Code is available at http://cse.seu.edu.cn/people/zhangml/Resources.htm.
[9] M. Chen, K.Q. Weinberger, J. Blitzer, Co-training for domain adaptation, Advances in neural information processing systems (pp. 2456–2464), 2011. Code is available at http://www.cs.cornell.edu/~kilian/code/code.html.
[10] http://labic.icmc.usp.br/?q=node/762.
[11] D. Carter, Inferring aspect-specific opinion structure in product reviews (doctoral dissertation, université d'ottawa/university of ottawa), 2015, https://github.com/davecart/cotraining.