Universidade Estadual de Campinas
Instituto de Computação

Luan de Oliveira Silveira

Statistical Analysis of Semi-supervised Algorithms for Tabular Data

Análise Estatística de Algoritmos Semi-supervisionados Para Dados Tabulares

CAMPINAS
2024

Luan de Oliveira Silveira

Statistical Analysis of Semi-supervised Algorithms for Tabular Data

Análise Estatística de Algoritmos Semi-supervisionados Para Dados Tabulares

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Prof. Dr. Jacques Wainer

CAMPINAS
2024

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Silvania Renata de Jesus Ribeiro - CRB 8/6592

Si39s

Silveira, Luan de Oliveira, 1996-
Statistical analysis of semi-supervised algorithms for tabular data / Luan de Oliveira Silveira. – Campinas, SP : [s.n.], 2024.

Orientador: Jacques Wainer.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Aprendizado de máquina. 2. Aprendizagem semi-supervisionada. I. Wainer, Jacques, 1958-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações Complementares

Título em outro idioma: Análise estatística de algoritmos semi-supervisionados para dados tabulares
Palavras-chave em inglês:
Machine learning
Semi-supervised learning
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Jacques Wainer [Orientador]
Marcelo da Silva Reis
Ronaldo Cristiano Prati
Data de defesa: 27-03-2024
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0009-0000-2751-9759
- Currículo Lattes do autor: http://lattes.cnpq.br/3626185560647353

Luan de Oliveira Silveira

Statistical Analysis of Semi-supervised Algorithms for Tabular Data

Análise Estatística de Algoritmos Semi-supervisionados Para Dados Tabulares

Campinas, 27 de março de 2024

Dedication

To the endless pursuit of knowledge and truth.

Veritas Propter Veritatem.

Acknowledgments

I want to thank my family for their continued support in my pursuit of education.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

Resumo

Considerando os benefícios e limitações de cada paradigma de aprendizagem no campo de Aprendizado de Máquina, a área de semi-supervisão vem ganhado destaque em sua tentativa de extrair informação de dados rotulados e não-rotulados simultaneamente. Em sua essência, busca uma performance comparável à aprendizagem supervisionada, enquanto se mantém fácil e barata de ser treinada. Dado o rico ecossistema de algoritmos nesta área, nós propomos uma avaliação de 10 métodos em um grupo diverso de bancos de dados, além de diferentes cenários de disponibilidade de dados. Ademais, propomos uma análise estatística das acurácias global, positiva e negativa como métricas de performance, seguida de testes estatísticos para a análise de diferenças significativas entre os métodos como um grupo, além de diferenças par-a-par, sendo possível determinar algoritmos que superam outros com consistência.

Abstract

Given the strengths and weaknesses of each learning paradigm in Machine Learning, the semi-supervised setting have been gaining traction as an attempt to extract information of both labeled and unlabeled data. Aiming at a performance akin to its supervised counterpart, but being as easy and cheap to train as an unsupervised approach. Given the rich ecosystem of algorithms in this field, we propose an evaluation of 10 methods in a diverse group of datasets and different scenarios of data availability. We aim at a statistical analysis of the global, positive and negative accuracies as the metrics of performance, with following statistical tests to access differences among the methods as a group, as well as pairwise similarities. Being also possible to determine methods that consistently outperform others in a given training setting.

List of Figures

List of Tables

List of Abbreviations and Acronyms

ML	Machine Learning
Rbf	Radial Basis Function
ASB	Assemble algorithm
TSVM	Transductive Support Vector Machine algorithm
LapSVM	Laplacian Support Vector Machine algorithm
SSGMM	Semi-Supervised Gaussian Mixture Model algorithm
LabelS	Label Spreading algorithm
SemiB	SemiBoost algorithm
LabelP	Label Propagation algorithm
SelfT	Self-Training algorithm
CoT	Co-Training algorithm
TriT	Tri-Training algorithm

List of Symbols

$\mathcal{X}$	Input space; set of all possible inputs $\vec{x}$
$\mathcal{Y}$	Output space; set of all possible outputs y; set of classes
$\mathcal{L}$	Labeled dataset composed of pairs $(\vec{x}_i, y_i)$ of inputs $\vec{x}_i$ and their label y_i
$\mathcal{U}$	Unlabeled dataset composed of points $\vec{x}_i$
$\overline{\mathcal{L}}$	Pseudo-labeled dataset; the goal of a transductive learning problem
ℓ	Number of labeled examples, or $\lvert\mathcal{L}\rvert$
u	Number of unlabeled examples, or $\lvert\mathcal{U}\rvert$
$\mathcal{H}$	Supervised algorithm classifier
τ	Confidence threshold for the self-training algorithm
W	Similarity matrix
σ	Similarity coefficient of the rbf kernel
$\mathcal{E}$	Error function to be minimized
f_i	An individual classifier function in boosting methods at an iteration i
F_i	An assemble of classifiers in boosting methods at an iteration i
$\mathcal{C}$	The classifier function; the goal function of an inductive learning problem
$\langle a, b\rangle$	Inner product of a and b
$\odot$	Point-wise product of two vectors; Hadamard product
ξ	Slack variable in a SVM problem

Contents

Chapter 1

Introduction

1.1 Context

Since the boom of the Machine Learning, hundreds of areas have benefited from the potential aptitude of computers to perform certain tasks, not only faster, but more accurately than humans. From self-driving vehicles to medical diagnosis, the plethora of applications of this field of artificial intelligence signals another turning point in the quality of human life.

Commonly, ML is separated into three distinct sub-fields: supervised, unsupervised and reinforcement learning. The first two, who shall be mentioned extensively in this work, groups the methods that are dependent (supervised) or independent (unsupervised) on the desired output, usually serving distinct purposes. However, the acquisition of labels stands a problem in comparison with simply obtaining the inputs, usually requiring a great amount of resources such as time, money and expertise. In this scenario of abundance of unlabeled data and the efficiency of supervised training, a new paradigm called semi-supervised learning is gaining space with the aim to take advantage of labeled and unlabeled data in the train of the same model.

The vast scope of the semi-supervised learning configuration and the shared similarities between methods has given rise to numerous attempts of grouping such methods. The proposed work is based on the novel semi-supervised taxonomy given by [33], shown in Figure (1.1).

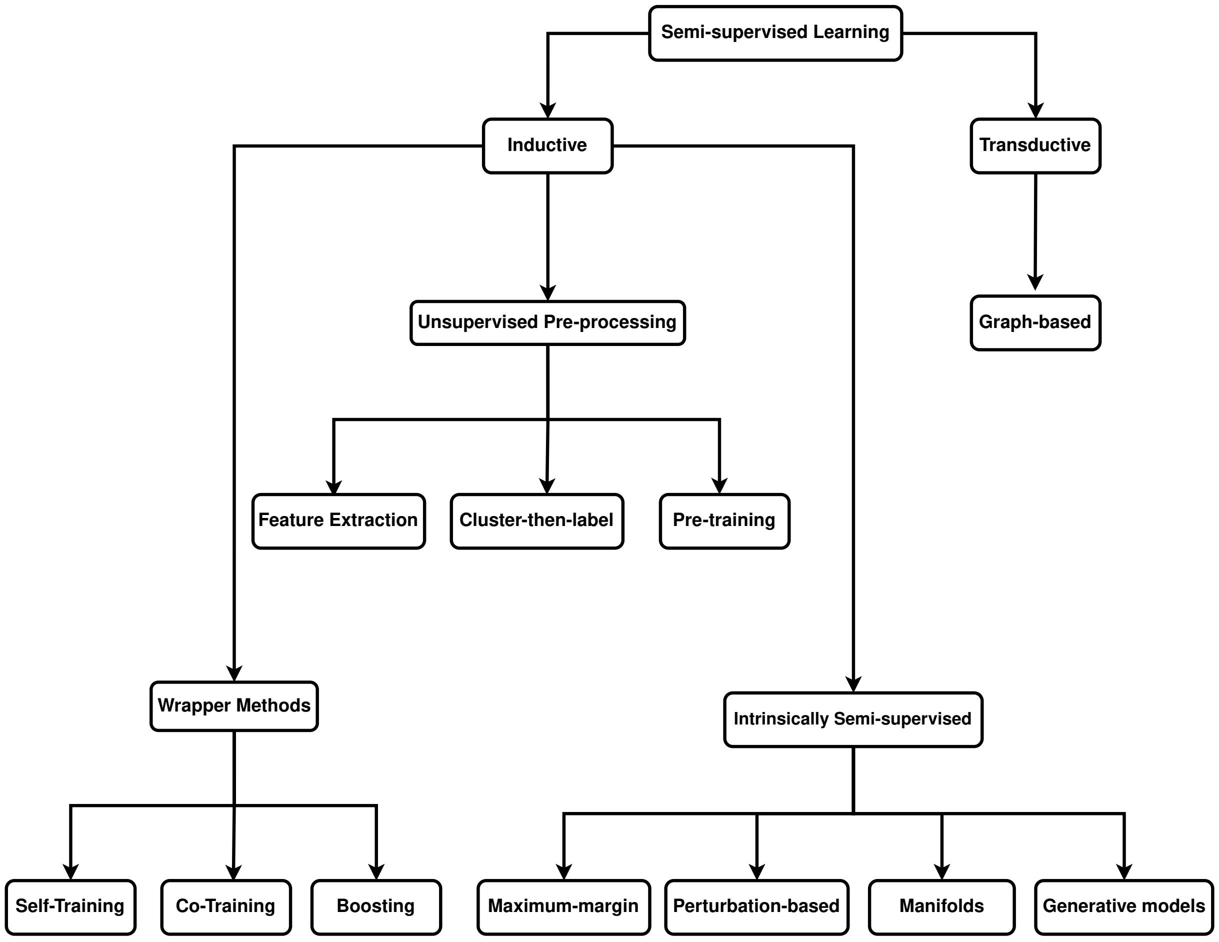

Figure 1.1. Semi-supervised learning taxonomy.
Source: Adapted from [33].

The first major distinction is between inductive and transductive methods. The former represents algorithms whose objective is to find a function capable of handling unseen data, while the latter is only concerned with the best modeling of the dataset used for training, disregarding any extrapolation.

Inductive methods are further divided into 3 categories, namely wrapper methods, unsupervised preprocessing and intrinsically semi-supervised. These groups are distinguished by how the handling of the unlabeled data is conducted. Wrapper methods assembles the algorithms that rely on one or more supervised learners, training on the labeled dataset and pseudo-labeling the rest along with a criteria of confidence to expand the training set for the next iteration. Unsupervised preprocessing on the other hand, sees the unlabeled dataset as a guide to the fine-tuning of hyper-parameters, usually applying known unsupervised learning algorithms to extract information on the set $\mathcal{X}$ of all possible inputs. Finally, the intrinsically semi-supervised agglomerate are characterized by methods that modify the objective function of known supervised algorithms, usually with the goal of modeling the error of the unlabeled dataset.

Transductive methods usually defines a graph over the labeled and unlabeled data points, and models the similarity between points by a weighted adjacency matrix [40]. An objective function is then designed to achieve two goals: the correct classification of the labeled dataset and to classify similar data points with the same labels [33]. The general framework for these methods involves three steps. First, the creation of a graph by creating links through each data point (nodes) with a predetermined similarity measure. Secondly, we weight the resulting similarity matrix, and thirdly the inference step, where we use the obtained graph to predict the unlabeled dataset [21].

1.2 The Problem and Objectives

As the field expands, different approaches and algorithms are proposed to tackle different learning problems. However, these methods are usually presented as top performers on cherry-picked datasets, with very little discussion on their performance on a diverse selection of datasets, as well as a comparison of different methods that might not share similarities in their derivation. With this in mind, we propose an evaluation of the performance of 10 different semi-supervised algorithms for a wide selection of datasets. We accompany these experiments with a statistical analysis of the global, positive and negative accuracies of each method in each condition, and we evaluate their p-values through different existing tests to establish a sensible conclusion to our experiments. This approach allow us to compare the algorithms as a whole, pairwise differences between them, as well as their average ranks on a given learning setting, showing algorithms that consistently performs better than others. Our aim is to strengthen the knowledge of the field, and aid further research and applications in the area, with our comparison of different methods in a diverse setting serving as a guide for those interested in using semi-supervised learning to solve a task, but unsure of the appropriate model, or what to expect.

1.3 Dissertation Structure

We begin the dissertation with a brief literature review of the benchmarks of semi-supervised algorithms. We aim to show how these studies are carried, as well as to offer a critique of their results, which usually suggests a deliberate selection of scenarios where the main algorithm would best perform. We move on to Chapter 3, where we lay the theoretical foundations of the algorithms of interest, offering derivations and the algorithms themselves in pseudo-code, as well as a discussion of their convergence, hyper-parameters and some variations. Chapter 4 contains the methodology of the study, with the description of each experiment carried, the expected format of the results and their intended analysis. We then present the results in Chapter 5 as well as their interpretation. Finally, we end with a conclusion in Chapter 6.

Chapter 2

Literature Review

In this chapter we present the state of the literature regarding the performance of semi-supervised algorithms. We divide it into two sections, one pertaining to the original papers, responsible for proposing a new algorithm and testing its performance against other know methods. And in the other, we present related works in the field that contains similarities in methodology, or comparisons with one or more algorithms of interest to this study.

2.1 Original Papers

2.1.1 Self-Training

In its original paper [37], David Yarowsky (1995) focuses on the specific problem of discerning different meanings of a word depending on the context. The study is conducted for 12 words, with the data being extracted from a 460 million word corpus containing news articles, scientific abstracts, spoken transcripts, and novels, consisting of one of the biggest training and testing sets for the study of sense-disambiguation at the time [37]. The algorithm's performance is compared to two others. The decision list supervised algorithm, where it shows a comparable performance (less than 1% difference), and Schütze's algorithm [29], which was outperformed by the Self-Training.

2.1.2 Label Propagation

The Label Propagation algorithm, proposed by Zhu and Ghaharamani in 2002 [41], was tested against two synthetic datasets, and a single real one. In the two synthetic sets the algorithm was compared to the kNN algorithm, which was outperformed by the Label Propagation. These datasets were highly symmetrical, and possessed an intrinsic geometric aspect to them, likely being chosen for how well the algorithm performs on it. For the real dataset, the algorithm is tested on the handwritten digits dataset originally from the Cedar Buffalo binary digits database [15], and its performance is compared to the 1NN (kNN for k=1), and the p1NN (propagating 1NN). The Label Propagation also outperforms the others in this chosen dataset.

2.1.3 Label Spreading

The Label Spreading algorithm was proposed by Zhou et al. in 2004 [38], and in their work, the algorithm is only tested on three datasets, two of them being real and the other synthetic. The latter is a toy dataset composed of two crescent shaped patterns, or moons, with each one of them representing a class. The performance of the Label Spreading is compared with the SVM method with a RBF kernel, having a superior performance after 400 iterations. For the real datasets, first we have the USPS handwritten digits set [16] is used with only the numbers 1, 2, 3 and 4 as the four classes. The proposed algorithm is then compared with the 1NN, SVM with RBF kernel (one-vs-rest) and harmonic Gaussian [42] methods. Secondly, they use the 20-newsgroups dataset for text classification choosing the single topic *rec*. On both datasets, the proposed Label Spreading, or a slightly variation of it, is the best performing method.

2.1.4 Assemble

The Assemble is an ensemble algorithm proposed by Bennett et al. in 2002 [5]. In their work, the Assemble is tested with two different base classifiers, decision trees and neural networks, on 3 datasets each, for 3 different unlabeled ratios for each dataset. The decision tree experiments were run on 3 of the 13 Gunnar Rätch's benchmark datasets [8] for 20, 40 and 60% unlabeled rate, being compared to the AdaBoost algorithm [11] and having a better performance in 2 of the 3 datasets for all ratios. For the neural network as the base classifier, the three datasets were selected from the UCI Machine Learning repository [24], and run for 10, 25 and 50% unlabeled ratio for each. The comparison was also with the AdaBoost, with the Assemble outperforming its competitor in every dataset and ratio.

2.1.5 SemiBoost

The SemiBoost algorithm was proposed by Kumar et al. in 2009 [22] as a boosting algorithm for semi-supervised learning. In their work, the SemiBoost was evaluated on 16 datasets and against 6 other methods. Three of them are supervised algorithms that were also used as the base classifiers, they are Decision Stump, Decision Tree and SVM. The remaining three algorithms are other semi-supervised methods, being the TSVM, LapSVM and ILDS (Inductive LDS). The results for the supervised comparison show that the SemiBoost with a given supervised base classifier provides a considerable increase over simple using the base classifier. Furthermore, comparing the best SemiBoost version (best performing base classifier) with the other semi-supervised algorithms, the SemiBoost was the best performing method in 8 of the 16 datasets, while also being second in another 5. Another experiment was run in the same setup but with 10 datasets adapted from the 20-newsgroup [1]. In this case, the SemiBoost was the best performer in 7 of the 10 sets, and the second best in 3 where it lost for the TSVM.

2.1.6 Co-Training

For the Co-Training algorithm, proposed by Blum and Mitchell [7] in 2000, the method was only tested for a single dataset, consisting in 1051 web pages collected from the department of Computer Science in four universities [2], and compared against a single supervised algorithm, the Naive Bayes classifier. Two Bayes classifiers are trained, one for each view of the dataset, and a third is created using the combined probability of them. These classifiers are compared to each respective classifier of the Co-Training for each view as well as the combined classifier. The Co-Training outperforms the Naive Bayes in both classifiers as well as the combined one.

2.1.7 Tri-Training

The Tri-Training algorithm was proposed by Zhou and Li in 2005 [39] as an attempt to improve the Co-Training approach. In their paper, three sets of experiments are carried, one for each different base classifier (J4.8 decision tree, BP neural network and Naive Bayes), and each is done for 40, 60 and 80% unlabeled rates. The algorithm is then tested on 12 datasets and against the Co-Training method with a random partition of the set into two for the different views, and the Self-Training. The results show that for each base classifier, the Tri-Training presents the best average improvement of performance when compared to the Co-Training and Self-Training, for all three unlabeled ratios. However, the best performing method for a dataset alternates between the algorithms.

2.1.8 TSVM

The TSVM was proposed by Joachims in 1999 [19], and was thought as an algorithm well suited for text classification. As such, in his paper, Joachims focus the experiments on 3 datasets for text classification and the performance of the TSVM is compared to the supervised algorithms Naive Bayes and SVM. In all 3 cases, the TSVM displayed the best performance (averaged for each word of interest in the dataset), with the SVM second and the Naive Bayes last.

2.1.9 LapSVM

The LapSVM algorithm, proposed by Belkin et al. in 2006 [4], was originally tested on 4 datasets, one synthetic and 3 real ones, while being compared to the SVM and TSVM. For the synthetic dataset, the two moons set is used with only one example from each class. The LapSVM is shown to better model the geometry of the dataset, when compared to the SVM and TSVM. For the 3 real datasets, it was used one for visual recognition, one for speech recognition and one for text categorization. The visual and speech datasets yield a clear winning performance of the LapSVM against the SVM and TSVM. For the text recognition, the LapSVM is superior to the SVM but having an error slightly higher ($\sim 3.5\%$) than the TSVM.

2.1.10 SSGMM

The SSGMM algorithm was proposed by Shahshahani and Landgrebe in 1994 [30]. In their work, the SSGMM is tested on a portion of the AVIRIS dataset for a four-class experiment. Other experiments are carried to evaluate the Hughes phenomenon (loss of classification potential when the dimension of the data is increased), and the impact of the unlabeled ratio in the accuracy of the model. It was shown, that with a higher unlabeled rate and the appropriate dimensionality of the data, the SSGMM can achieve accuracies higher than 95% in this dataset.

2.2 Related Work

Different attempts on benchmarking semi-supervised algorithms were performed over the years. A few noteworthy works are presented here, either by shared algorithms, or a similar approach to our methodology.

On their work, published in 2022, Wang et al. [36] offers a benchmark attempt for classification tasks of 14 deep semi-supervised learning algorithms on 15 datasets, 5 for computer vision, 5 for NLP and 5 for audio. Furthermore, the experiments are carried for 3 seeds, and the average rank of the algorithms are offered, similar to our methodology; however, no intersecting algorithms exists with the present work. Similarly, Oliver et al. [26] presents a smaller comparison of 5 deep semi-supervised algorithms on two datasets, the CIFAR-10 and SVNH. In their work, a focus on a high quality supervised base line, consideration of class distributions, and variations of unlabeled ratio is given, serving as a reference in the field for a quality comparison of semi-supervised methods.

In fact, a great focus has been given to deep semi-supervised learning [14, 32, 10, 31, 20]. As complex Computer Vision, Large Language Models and speech recognition tasks gained notoriety over the last few years, the research on semi-supervision applied on classification of tabular data has lagged behind. Reference works in this area, as presented in the previous section, consists in theorizing a new method or an improvement upon an existing approach, followed by a test of concept with too few datasets and competing methods for a relevant conclusion to be drawn. Moreover, not only these studies but also the more popular work on deep semi-supervision, often fails to report the statistical significance of their results.

Chapter 3

Theoretical Foundations

In this chapter we lay the theoretical foundations of each individual algorithm, including the initial motivations, the mathematics, the algorithm itself, a discussion of the hyper-parameters necessary, the conditions of convergence and possible variations if they exist. Each algorithm is presented in a unique pseudo-code notation. This notation is design to be intuitive and as close to standard mathematical notation as possible.

A few variables are common among the algorithms and can be understood as global variables. We denote the input space, that is, the set of all possible inputs $\vec{x}$, as $\mathcal{X}$. The set of all possible outputs or classes as $\mathcal{Y}$. The set of labeled examples ($(\vec{x}, y)$ pairs) as $\mathcal{L}$, and the set of unlabeled examples as $\mathcal{U}$.

The description of an algorithm begins by listing the inputs and outputs. Each declaration is followed by the variable's type given in set notation. For example, declaring $\mathcal{U} \subset \mathcal{X}$ tells us that $\mathcal{U}$ has the same type of $\mathcal{X}$, and similarly $\tau \in \mathbb{R}$ shows that the variable τ has the same type as the elements in $\mathbb{R}$, in this case τ is a real number. Lastly, we also declare functions using standard mathematical notation, for example $\mathcal{H} : \mathcal{X} \to \mathcal{Y}$ tells us that $\mathcal{H}$ is a function mapping from $\mathcal{X}$ to $\mathcal{Y}$. This can also be done as $\mathcal{H} \in \mathcal{Y}^{\mathcal{X}}$ since in set-theoretical notation $\mathcal{Y}^{\mathcal{X}}$ represents the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$.

After the declaration of inputs and outputs the algorithm is described in a sequence of numbered steps with a few standard keywords. We use **let** to declare variables, and **while**, **if**, **for**, **return** works analogously as most programming languages. The assignment operator is the "$\leftarrow$", and assignment of vectors or matrices are done by describing it's value at a certain position. To access the i-th value in a vector $\vec{v}$ we write $\vec{v}_i$. Similarly, to access the i-th row and j-th column in a matrix A we write A_{ij}.

3.1 Self-training

The Self-training algorithm is the most intuitive and straight-forward method in the semi-supervised class. It was initially proposed by David Yarowsky (1995) [37] in one of the first attempts to use both labeled and unlabeled data to a learning task, even classifying it as an unsupervised algorithm since semi-supervised learning was not yet recognized. In his work, Yarowsky attempts to disambiguate senses of a word in different phrases, starting with only a few phrases with tagged senses (called seeds), and the majority remaining untagged (85-98%). The algorithm consists of repeatedly training a supervised classifier and using it to predict the labels of the unlabeled dataset, so these points can be absorbed in future iterations.

Algorithm: 1 Self-training (Transductive)

Input:
$\mathcal{H} : \mathcal{X} \to \mathcal{Y}$ :: Supervised Algorithm Classifier
$\tau \in \mathbb{R}$ :: Confidence Threshold

Output:
$\overline{\mathcal{L}} \subset \mathcal{X} \times \mathcal{Y}$ :: Examples with Labels and Pseudo-labels

```
1.  let $\overline{\mathcal{L}} \subset \mathcal{X} \times \mathcal{Y}$, $\overline{\mathcal{L}}' \subset \mathcal{X} \times \mathcal{Y}$, $\vec{y} \in \mathbb{R}^{|\mathcal{Y}|}$, $c \in \mathbb{N}$
2.  $\overline{\mathcal{L}} \leftarrow \mathcal{L}$, $\overline{\mathcal{L}}' \leftarrow \emptyset$
3.  while $\overline{\mathcal{L}} \neq \overline{\mathcal{L}}'$ do:
4.          let $\mathcal{C} : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|} \leftarrow$ Train$(\mathcal{H}, \overline{\mathcal{L}})$
5.          $\overline{\mathcal{L}}' \leftarrow \overline{\mathcal{L}}$, $\overline{\mathcal{L}} \leftarrow \mathcal{L}$
6.          for $\vec{x} \in \mathcal{U}$ do:
7.                  $\vec{y} \leftarrow \mathcal{C}(\vec{x})$
8.                  $c \leftarrow \underset{j}{\operatorname{argmax}}\, y_j$
9.                  if $y_c \geq \tau$ do:
10.                         $\overline{\mathcal{L}} \leftarrow \overline{\mathcal{L}} \cup \{(\vec{x}, c)\}$
11. return $\overline{\mathcal{L}}$
```

The algorithm receives the labeled and unlabeled datasets, along with a confidence threshold $\tau \in \mathbb{R}$, and a probabilistic supervised classification algorithm $\mathcal{H}$.

We begin in line 1 by setting the altered labeled dataset $\overline{\mathcal{L}}$ as the original labeled set $\mathcal{L}$ for the first iteration. We then start the main loop of the algorithm that should run until convergence. The chosen supervised algorithm $\mathcal{H}$ is trained in the enhanced labeled dataset $\overline{\mathcal{L}}$ ($\mathcal{T}(\mathcal{H}, \overline{\mathcal{L}})$) and the probabilistic classifier $\mathcal{C}$ is obtained. This classifier is used in the unlabeled data set ($\mathcal{C}(\mathcal{U})$) to obtain the classification matrix $Y \in \mathbb{R}^{u \times c}$, where Y_{ij} is the probability of the i-th unlabeled example having the label j. Now we construct the new altered labeled set for the next iteration, starting it as the original set $\mathcal{L}$.

For every unlabeled point $\mathbf{x}_i$ in $\mathcal{U}$, represented in Y by the i-th row, we select the label c_i as the column with the highest probability ($\underset{j}{\operatorname{argmax}}\ Y_{ij}$). If this value, now known to be at position Y_{ic_i}, is greater or equal to the confidence threshold τ, we insert the point $(\mathbf{x}_i, c_i)$ in the enhanced labeled dataset $\overline{\mathcal{L}}$ for the next iteration.

After convergence the prediction matrix Y, now stable, is returned as the final result. The effective labeling of the original unlabeled dataset is given by the mapping
$\{(\mathbf{x}_i, c_i) : \mathbf{x}_i \in \mathcal{U}, c_i = \underset{j}{\operatorname{argmax}}\ Y_{ij}\}$.

Convergence

The convergence of the method is not guaranteed and is highly dependent on the chosen supervised classifier $\mathcal{H}$.

Hyperparameters

The model accepts two hyperparameters. The supervised classifier $\mathcal{H}$ is only required to be probabilistic, that is, given a point it must return the probability distribution for that point labels. This enables the adaptation of the classifier to the semi-supervised setting. Furthermore, since the algorithm must fulfill this single requirement, and given the diversity of probabilistic models, the self-training setting allows for plentiful variations by simple choosing different classifiers.

The second hyperparameter is the confidence threshold τ, representing the confidence necessary in the labeling to assimilate a point in the next iteration. The correct approach is to perform a hyperparameter search to determine τ, with no accepted heuristic to guess its value. However, it's important to point out that high values can cause the model to not absorb any new points in the new iteration, causing the classifier to converge quickly to a poor solution. Also, low values can be equally troublesome, causing the model to degenerate and assimilate all points and converge to a bad solution.

Variations

As discussed, each choice of a probabilistic supervised classifier is a valid variation of the method, making the self-training algorithm more of a framework to adapt supervised learning to benefit from unlabeled data. This sparks another kind of variation, as we move away from the motivation of finding labels to the original unlabeled data (transductive setting), to instead using the model to classify unseen unlabeled data (inductive setting). This way, the original set $\mathcal{U}$ is no longer seen as the main classifying goal, and instead takes the role of supplementary learning information. This is done simply altering the line 10 of the algorithm to "return $\mathcal{C}$", returning the supervised classifier instead of the set $\mathcal{U}$ labels.

3.2 Label Propagation

The Label Propagation algorithm, originally proposed in 2002 by Zhu [41], is the simplest of the graph based methods for semi-supervised learning. It relies on a probabilistic interpretation of the labels, where the label of a point is more likely to propagate or influence another, based on the distance between them. Label propagation is not a hard label method, and it keeps at all stages a list containing the probability of a given point having a given label. At convergence, different methods for choosing the final label can be applied, most notably, the assignment of the label with the highest probability.

The method starts by transforming the dataset into a graph. We interpret an arbitrary d-dimensional point $\mathbf{x}_i$ as the node i, and construct the weighted edges between two nodes i and j (w_{ij}) using the radial basis function (rbf):

$$w_{ij} = \exp\left(-\frac{d_{ij}^2}{2\sigma^2}\right) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x_j}\|^2}{2\sigma^2}\right). \tag{3.1}$$

Once the weight matrix W is constructed, we define the matrix T as the column-normalized W. In this case, a column in T can be interpreted as a probability distribution. More specifically, the entry T_{ij} is viewed as the probability $P(j \to i)$ of node j propagating its label to node i. The objective is to determine the probability of any node having a given label, so we define the matrix $Y \in \mathbb{R}^{n \times c}$, where the entry Y_{ik} is the probability that node i has the label k.

We start with an initial label distribution $Y^{(0)}$ and let they propagate their labels such that each entry Y_{ik} in the label distribution matrix will be updated to the sum of the probabilities of each node propagating the label k to the node i.

$$Y_{ik} \leftarrow \sum_{j=1}^{l+u} P(j \to i) P(\text{label of j} = \text{k}) = \sum_{j=1}^{l+u} = T_{ij} Y_{jk}. \tag{3.2}$$

The process can be easily written in matrix notation as $Y \leftarrow TY$. The rows in the new matrix Y doesn't necessarily add to 1, so we row-normalize it to maintain the probability interpretation of the matrix. Next, we replace the label distribution of the labeled examples in Y to their true labels to maintain consistency. We then repeat this process until Y converges.

Algorithm: 2 Label Propagation

```
Input:
$\sigma \in \mathbb{R}$    :: Similarity Coefficient

Output:
$\overline{\mathcal{L}} \subset \mathcal{X} \times \mathcal{Y}$    :: Examples with Labels and Pseudo-labels
----------------------------------------------------------------
1. let $Y \in \mathbb{R}^{(\ell+u)\times|\mathcal{Y}|} \leftarrow Y_{ij} = \begin{cases} 1, & \text{if } (\vec{x}_i, j) \in \mathcal{L} \\ 0, & \text{otherwise} \end{cases}$
2. let $Y' \in \mathbb{R}^{(\ell+u)\times|\mathcal{Y}|} \leftarrow Y'_{ij} = 0$
3. let $W \in \mathbb{R}^{(l+u)\times(l+u)} \leftarrow W_{ij} = \exp\left(-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{2\sigma^2}\right)$
4. let $T \in \mathbb{R}^{(l+u)\times(l+u)} \leftarrow T_{ij} = W_{ij} / \sum_{k=1}^{l+u} W_{kj}$
5. let $\overline{T} \in \mathbb{R}^{(l+u)\times(l+u)} \leftarrow \overline{T}_{ij} = T_{ij} / \sum_{k=1}^{l+u} T_{ik}$
6. while $Y \neq Y'$ do:
7.         $Y' \leftarrow Y$
8.         $Y \leftarrow \overline{T}Y$
9.         $Y \leftarrow Y_i = \begin{cases} Y'_i, & \text{if } 1 \leq i \leq \ell \\ Y_i, & \text{if } \ell+1 \leq i \leq u \end{cases}$
10. let $\overline{\mathcal{L}} \subset \mathcal{X} \times \mathcal{Y} \leftarrow \{(\vec{x}_i, c) : 1 \leq i \leq \ell + u, c = \operatorname{argmax}_j Y_{ij}\}$
11. return $\overline{\mathcal{L}}$
```

Convergence

To prove the convergence of Y we observe that the first l rows are the labeled examples, while the last u rows are the unlabeled entries. Since the labels are clamped for the labeled examples, the first l rows remain unchanged. Let Y_L be the $l \times c$ matrix made by the first l entries of Y and Y_U by the last u entries. Let $\overline{T}$ be divided in the 4 sub-matrices

$$\overline{T} = \begin{bmatrix} \overline{T}_{ll} & \overline{T}_{lu} \\ \overline{T}_{ul} & \overline{T}_{uu} \end{bmatrix}. \tag{3.3}$$

We need only to prove that the sub-matrix Y_U of Y converges, since the other entries remain fixed. A simple algebraic manipulation shows that

$$Y_U^{(i+1)} \leftarrow \overline{T}_{uu} Y_U^{(i)} + \overline{T}_{ul} Y_L. \tag{3.4}$$

Taking the limit with the number of iterations

$$Y_U = \lim_{n\to\infty} \overline{T}_{uu}^n Y^{(0)} + \left(\sum_{i=1}^{n} \overline{T}_{uu}^{(i-1)} \right) \overline{T}_{ul} Y_L. \tag{3.5}$$

Since $\overline{T}$ is row-normalized and a row in $\overline{T}_{uu}$ have less elements than a row in $\overline{T}$ we know that the row sums of $\overline{T}_{uu}$ must be smaller than 1, that is

$$\exists \gamma < 1, \sum_{j=1}^{u} \overline{T}_{uu_{ij}} \leq \gamma. \tag{3.6}$$

Hence

$$\sum_{j=1}^{u} \overline{T}_{uu_{ij}}^{n} = \sum_{j=1}^{u} \sum_{k=1}^{u} \overline{T}_{uu_{ik}}^{n-1} \overline{T}_{uu_{kj}} \tag{3.7a}$$

$$\sum_{j=1}^{u} \overline{T}_{uu_{ij}}^{n} = \sum_{k=1}^{u} \overline{T}_{uu_{ik}}^{n-1} \sum_{j=1}^{u} \overline{T}_{uu_{kj}} \tag{3.7b}$$

$$\sum_{j=1}^{u} \overline{T}_{uu_{ij}}^{n} < \sum_{k=1}^{u} \overline{T}_{uu_{ik}}^{n-1} \cdot \gamma \tag{3.7c}$$

$$\sum_{j=1}^{u} \overline{T}_{uu_{ij}}^{n} < \gamma^{n}. \tag{3.7d}$$

Since the row summations of $\overline{T}_{uu}^n$ tends to zero, the first term of equation 5 also converges to zero, making the initial choice of $Y^{(0)}$ irrelevant. The second term will converge to

$$(I - \overline{T}_{uu})^{-1} \overline{T}_{ul} Y_L \tag{3.8}$$

which is clearly a fixed value, thus proving that the method will converge.

Hyperparameters

The only hyperparameter involved in the label propagation algorithm is the σ in the rbf function. This value regulates the probability of a node propagating its label to distant points, with higher values resulting in points influencing a wider area with their labels. In their original description of the method, Zhu [41] also proposes an heuristic for the value of σ. It consists of performing Kruskal's algorithm for finding the minimum spanning tree over the data points until the smallest edge between two nodes with different labels is found. The value of σ is then set to be 3 times the Euclidean distance between these two points, thus following the 3σ rule. This way the influence of a point towards a different labeled one is considered insignificant.

Variations

The most traditional way of assigning the final labels is by picking the most likely class, that is, the row with the highest value in Y_U. However, when class proportions are fixed, the most likely method doesn't guarantee that the correct ratio will be preserved. In these situations, two methods are proposed to maintain consistency, Class Mass Normalization and Label Bidding. The first consists of scaling the column sums of the matrix Y_U to be the proportion of labels from that class, and picking the most likely value after scaling. The second considers that there are up_c c labels to be distributed, and each point will bid proportional to $Y_{U_{ic}}$, and if there are c labels still available one will be given to the point bidding, at which point it will quit the process. However, if no more labels c are available, the second highest bid is evaluated, and so on.

3.3 Label Spreading

The Label Spreading algorithm is closely related to the Label Propagation. Their main differences are in the construction of the similarity matrix and the clamping of the labeled data. The algorithm was first proposed by [38] who claims to be inspired by previous works on spreading activation networks and diffusion kernels, and is another example of graph-based methods.

The algorithm can be seen as spreading the label information of each point to its neighbors. This information is in the form of a c-dimensional vector, where c is the number of possible labels, for each point and contains the amount of label i information at entry i. However, differently from the Label Propagation algorithm, there is no concern with a probabilistic interpretation of this information, but the final labeling decision is still made towards the label with the highest value.

Algorithm: 3 Label Spreading

Input:

$\sigma \in \mathbb{R}$	:: Similarity Coefficient
$\alpha \in \mathbb{R}$	:: Weight of Propagation Information

Output:

$\overline{\mathcal{L}} \subset \mathcal{X} \times \mathcal{Y}$	:: Examples with Labels and Pseudo-labels

```
1. let $Y \in \mathbb{R}^{(\ell+u)\times|\mathcal{Y}|} \leftarrow Y_{ij} = \begin{cases} 1, & \text{if } (\vec{x}_i, j) \in \mathcal{L} \\ 0, & \text{otherwise} \end{cases}$
2. let $Y' \in \mathbb{R}^{(\ell+u)\times|\mathcal{Y}|} \leftarrow Y'_{ij} = 0$
3. let $Y^{(0)} \leftarrow Y$
4. let $W \in \mathbb{R}^{(l+u)\times(l+u)} \leftarrow W_{ij} = \exp\left(-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{2\sigma^2}\right)$
5. let $D \in \mathbb{R}^{(l+u)\times(l+u)} \leftarrow D_{ij} = \begin{cases} \sum_{k=1}^{l+u} W_{ik}, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$
6. let $T \in \mathbb{R}^{(l+u)\times(l+u)} \leftarrow D^{-1/2} W D^{-1/2}$
7. while $Y \neq Y'$ do:
8.         $Y' \leftarrow Y$
9.         $Y \leftarrow \alpha T Y + (1-\alpha) Y^{(0)}$
10. let $\overline{\mathcal{L}} \subset \mathcal{X} \times \mathcal{Y} \leftarrow \{(\vec{x}_i, c) : 1 \leq i \leq \ell + u, c = \operatorname{argmax}_j Y_{ij}\}$
11. return $\overline{\mathcal{L}}$
```

Convergence

To show that the algorithm converges take the iterative equation

$$Y^{(i)} = \alpha T Y^{(i-1)} + (1-\alpha) Y^{(0)}. \tag{3.9}$$

Substituting the right hand-side repeatedly, we get

$$Y^{(i)} = (\alpha T)^i Y^{(0)} + (1-\alpha) \sum_{k=0}^{i-1} (\alpha T)^k Y^{(0)}. \tag{3.10}$$

Let the converging matrix be Y, we need to find

$$Y = \lim_{i\to\infty} Y^{(i)} = \lim_{i\to\infty} (\alpha T)^i Y^{(0)} + \lim_{i\to\infty} (1-\alpha) \sum_{k=0}^{i-1} (\alpha T)^k Y^{(0)}. \tag{3.11}$$

Since $0 < \alpha < 1$ and $|\lambda_j| \leq 1$, where λ_j is any eigenvalue of T, we have

$$\lim_{i\to\infty} (\alpha T)^i Y^{(0)} = 0 \tag{3.12}$$

for the first term, and for the second we have

$$(1-\alpha) \lim_{i\to\infty} \sum_{k=0}^{i-1} (\alpha T)^k Y^{(0)} = (1-\alpha)(I - \alpha T)^{-1} Y^{(0)}. \tag{3.13}$$

Yielding

$$Y = (1-\alpha)(I - \alpha T)^{-1} Y^{(0)}. \tag{3.14}$$

This shows that the algorithm converges to the given matrix Y.

Hyperparameters

Two hyperparameters play a role in the Label Spreading algorithm. The σ value controls how fast the affinity between two points decreases with the distance, as already seen when presenting the rbf function. Secondly, the α parameter regulates the trade-off between retaining the initial information and absorbing the information propagated from neighboring points. In their work [38] fixed $\alpha = 0.99$ for better results, suggesting a low retention of the initial information is beneficial.

Variations

Two types of variations of the algorithm are proposed by the authors. The simplest variation is to repeat the algorithm after convergence and take $Y^{(0)}$ as the previous converged matrix. Repeating this process n times would yield $Y = (1-\alpha)(I - \alpha S)^{-n} Y^{(0)}$. The second type arises from choosing another normalization for the affinity matrix, mainly $P = D^{-1}W = D^{-1/2}TD^{1/2}$, or its transpose, what would yield $Y = (1-\alpha)(I - \alpha P^T)^{-1} Y = (1-\alpha)(D - \alpha W)^{-1} Y$.

3.4 Assemble

Boosting algorithms relies on the minimization of an error function. This poses an obstacle in adapting them to the semi-supervised setting, once it's not obvious how we can quantify the error of an unlabeled point since we don't have access to its correct label. To see how this can be solved, lets first consider a simple binary classification task, and later generalize it for a multi-class configuration.

Since our initial goal is to classify between two classes, we can define them to be 1 and -1. This simplifies the interpretation of the output to the class 1 if it is positive or -1 if negative (once the decision boundary is at 0). Furthermore, we allow the classifying ensemble to take any real value, so even though we can obtain the class by taking the sign of the output (positive or negative), we can interpret the absolute value as how certain the classifier is in its prediction. This is necessary for modeling an error function.

Consider a base classifier as a function $f : \mathcal{X} \to \mathcal{Y}$ mapping the input space $\mathcal{X} \subseteq \mathbb{R}^n$ into the output space $\mathcal{Y}$ (in the binary case we have $\mathcal{Y} = \{-1, 1\}$). We want to construct the final classifier F_T as an ensemble of T of these base classifiers by linearly combining them, that is, $F_T(\vec{x}) = \sum_{t=1}^{T} w_t f_t(\vec{x})$, where w_t is a positive real number representing the weight given to the classifier t.

Finally, we attempt to construct the error function $\mathcal{E} : \mathcal{X}^{\mathcal{Y}} \to \mathbb{R}$ that maps the function space into the real numbers, allowing us to search for a function that will minimize this error. The standard approach is to define the error of a single point and sum it over all the points. For this we need to quantify the error of labeled and unlabeled examples. Here we assume that the classifier correctly classified an unlabeled point $\vec{x}_i$ with the absolute value $|F(\vec{x}_i)|$ representing how certain the classifier F is in the label of this point. We can then take the negative of this value to represent uncertainty in the prediction. Using the fact that the labels are 1 and -1, we can get rid of the absolute function by representing the uncertainty as $-y_i F(\vec{x}_i)$. Observe that a correct classification, that is $\text{sign}(y_i) = \text{sign}(F(\vec{x}_i))$ with $\text{sign}(a) = 1$ if $a \geq 0$ or $\text{sign}(a) = -1$ if $a < 0$, will give a negative value and a misclassification ($\text{sign}(y_i) \neq \text{sign}(F(\vec{x}_i))$ will yield a positive one. Assuming that the error is proportional to this uncertainty, we wrap its value in a differentiable and monotonically increasing function $M : \mathbb{R} \to \mathbb{R}$, so the higher the uncertainty the higher the error. We leave M unspecified for now, requiring only that it satisfies the given properties, so specific choices will yield different variations of the algorithm. This strategy works for both labeled and unlabeled data, with y_i being the correct class for labeled examples and equal to $sign(F(\vec{x}_i))$ for unlabeled ones. This treats the predicted class of an unlabeled example as always correct, and is equivalent to the assumption that the error of these unlabeled examples doesn't come from its predicted label, but instead by how sure the classifier is in its prediction.

We now define the total error function $\mathcal{E}(F) : \mathcal{X}^{\mathcal{Y}} \to \mathbb{R}$ as

$$\mathcal{E}(F) = \sum_{i=1}^{l+u} \alpha_i M(-y_i F(\vec{x}_i)) \tag{3.15}$$

so each individual error $M(-y_i F(\vec{x}_i))$ is weighted by α_i and summed to obtain the total error associated with using the function F as a classifier. We now present the generic semi-supervised boosting algorithm [5] upon which the Assemble method is constructed from minor adjustments.

Algorithm: 4 Semi-Supervised Boosting (Binary)

Input:

$\mathcal{H} : \mathcal{X} \to \mathcal{Y}$	:: Supervised Algorithm Classifier
$T \in \mathbb{Z}^+$	:: Positive Integer
$\vec{\alpha} \in \mathbb{R}^{l+u}$	:: Example Weight Vector
$\vec{w} \in \mathbb{R}^T$	:: Classifier Weight Vector
$\mathcal{E} : \mathcal{Y}^{\mathcal{X}} \to \mathbb{R}$	:: Error Function

Output:

$\mathcal{C} : \mathcal{X} \to \mathcal{Y}$	:: Classifier

```
 1. let $F_0(\vec{x}) : \mathcal{X} \to \mathbb{R} \leftarrow 0$
 2. let $\mathcal{C} : \mathcal{X} \to \mathcal{Y}$
 3. let $\overline{\mathcal{L}} \leftarrow \mathcal{L}$
 4. for $t \leftarrow 0$ to $T - 1$ do:
 5.         let $(f_{t+1} : \mathcal{X} \to \mathbb{R}) \leftarrow \textbf{Train}(\mathcal{H}, \overline{\mathcal{L}})$
 6.         let $\vec{F}_t \in \mathbb{R}^{\ell+u} \leftarrow F_{t,i} = F_t(\vec{x}_i)$
 7.         let $\vec{f}_{t+1} \in \mathbb{R}^{\ell+u} \leftarrow f_{t+1,i} = f_{t+1}(\vec{x}_i)$
 8.         if $\langle \nabla \mathcal{E}(\vec{F}_t), \vec{f}_{t+1} \rangle > 0$ do:
 9.                 $\mathcal{C} \leftarrow \textbf{sign}(F_t)$
10.                 return $\mathcal{C}(\vec{x})$
11.         let $F_{t+1} \leftarrow F_t + w_{t+1} f_{t+1}$
12.         $\mathcal{C} \leftarrow \textbf{sign}(F_{t+1})$
13.         $\overline{\mathcal{L}} \leftarrow \mathcal{L} \cup \{(\vec{x}, C(\vec{x})) : \vec{x} \in \mathcal{U}\}$
14. return $\mathcal{C}$
```

The algorithm starts the ensemble F with the zero function and sets the training set

$\overline{\mathcal{L}}$ as the labeled dataset. We then begin the main loop that iterates T times. The loop begins by training a new classifier f_{t+1} with a supervised algorithm $\mathcal{H}$ over the training set.

Before we add the newly obtained classifier to the ensemble, we have to determine if this addition will lower the cost function, otherwise it doesn't make mathematical sense and the algorithm should halt. Essentially, we want to know if $\Delta\mathcal{E} = \mathcal{E}(F_{t+1}) - \mathcal{E}(F_t) \leq 0$. There are two ways to obtain this information. The first and more computationally expensive is to fully calculate $\Delta\mathcal{E}$ by Eq. (3.15) and determine its sign. The second is more ingenious and can determine the sign of the difference easier. Instead of seeing $\mathcal{E}$ as a function of F, lets shift the domain to $\mathbb{R}^{l+u}$ as below

$$\mathcal{E}(F) = \mathcal{E}(F(\vec{x}_1), \ldots, F(\vec{x}_{l+u})) = \sum_{i=1}^{l+u} \alpha_i M(-y_i F(\vec{x}_i)). \tag{3.16}$$

The function remains exactly the same. However, since each new variable $F(\vec{x}_i)$ can take any real value, we can treat $\mathcal{E}$ as a continuous function, and as such, we can use calculus to estimate $\Delta\mathcal{E}$. For simplicity, define the vector $(g(\vec{x}_1), \ldots, g(\vec{x}_{l+u}))$ as $\vec{g}$ for any function g. The linear approximation of $\mathcal{E}$ at its current position $\vec{F}_t$ in space yields

$$\mathcal{E}(\vec{x}) - \mathcal{E}(\vec{F}_t) \approx \nabla\mathcal{E}(\vec{F}_t) \cdot (\vec{x} - \vec{F}_t). \tag{3.17}$$

Taking $\vec{x} = \vec{F}_{t+1}$,

$$\mathcal{E}(\vec{F}_{t+1}) - \mathcal{E}(\vec{F}_t) \approx \nabla\mathcal{E}(\vec{F}_t) \cdot (\vec{F}_{t+1} - \vec{F}_t) \tag{3.18}$$

$$\Delta\mathcal{E} \approx \nabla\mathcal{E}(\vec{F}_t) \cdot (\vec{F}_{t+1} - \vec{F}_t). \tag{3.19}$$

We can simplify $\vec{F}_{t+1} - \vec{F}_t$ to

$$\vec{F}_{t+1} - \vec{F}_t = (F_{t+1}(\vec{x}_1), \ldots, F_{t+1}(\vec{x}_{l+u})) - (F_t(\vec{x}_1), \ldots, F_t(\vec{x}_{l+u})) \tag{3.20a}$$

$$\vec{F}_{t+1} - \vec{F}_t = (F_t(\vec{x}_1) + w_{t+1}f_{t+1}(\vec{x}_1), \ldots, F_t(\vec{x}_{l+u}) + w_{t+1}f_{t+1}(\vec{x}_{l+u})) - (F_t(\vec{x}_1), \ldots, F_t(\vec{x}_{l+u})) \tag{3.20b}$$

$$\vec{F}_{t+1} - \vec{F}_t = (w_{t+1}f_{t+1}(\vec{x}_1), \ldots, w_{t+1}f_{t+1}(\vec{x}_{l+u})) \tag{3.20c}$$

$$\vec{F}_{t+1} - \vec{F}_t = w_{t+1}\vec{f}_{t+1}. \tag{3.20d}$$

Combining Eq. (3.19) and Eq. (3.20d) we have

$$\Delta\mathcal{E} \approx w_{t+1}\nabla\mathcal{E}(\vec{F}_t) \cdot \vec{f}_{t+1}. \tag{3.21}$$

Since w_{t+1} is positive, the sign of $\Delta\mathcal{E}$ only depends on the vector product, that is

$$\Delta\mathcal{E} \leq 0 \iff \langle \nabla\mathcal{E}(\vec{F}_t), \vec{f}_{t+1} \rangle \leq 0 \tag{3.22}$$

where the inner product is rewritten in standard $\langle \cdot, \cdot \rangle$ notation. In summary, if this condition fails to be satisfied, adding the classifier $f_{t+1}(\vec{x})$ to the ensemble cannot decrease the error function, and the algorithm stops early. Otherwise, we continue by creating the new ensemble by adding f_{t+1} to the previous one, and update the classifier to be the sign function of this ensemble. Finally, we restart the training set $\overline{\mathcal{L}}$ as the labeled dataset and start adding the pseudo-labeled points from $\mathcal{U}$ to it. After T iterations, the algorithm terminates and returns the classifier $\mathcal{C}(\vec{x}) = sign(F_T(\vec{x}))$.

Observe that another way to view the problem is to find the base classifier that minimizes the difference $\Delta\mathcal{E}$ in the error function at every iteration, and from Eq. (3.21) this is equivalent to minimizing the inner product $\langle \nabla\mathcal{E}(\vec{F}_t), \vec{f}_{t+1} \rangle$. Expanding the product as

$$\langle \nabla\mathcal{E}(\vec{F}_t), \vec{f}_{t+1} \rangle = \sum_{i=1}^{l+u} \alpha_i y_i f_{t+1}(\vec{x}_i) M'(y_i F_t(\vec{x}_i)) \tag{3.23}$$

and observing that $y_i = f_{t+1}(\vec{x}_i) \implies y_i f_{t+1}(\vec{x}_i) = 1$ and $y_i \neq f_{t+1}(\vec{x}_i) \implies y_i f_{t+1}(\vec{x}_i) = -1$, since they can take only the values 1 and -1, we rewrite the inner product as

$$\langle \nabla\mathcal{E}(\vec{F}_t), \vec{f}_{t+1} \rangle = \sum_{y_i = f_{t+1}(\vec{x}_i)} \alpha_i M'(y_i F_t(\vec{x}_i)) - \sum_{y_i \neq f_{t+1}(\vec{x}_i)} \alpha_i M'(y_i F_t(\vec{x}_i)). \tag{3.24}$$

The value $\alpha_i M'(y_i F_t(\vec{x}_i))$ can be interpreted as a misclassification cost of the point $\vec{x}_i$, with it being negative for a correct classification, represented by the first summation, and positive for a misclassification, represented by the second summation (remember that $M'(a) \leq 0$ for all a). Now, we construct a normalized cost vector $\vec{D}$. Let

$$z = \left| \sum_{i=1}^{l+u} \alpha_i M'(y_i F_t(\vec{x}_i)) \right| = -\sum_{i=1}^{l+u} \alpha_i M'(y_i F_t(\vec{x}_i)) \tag{3.25}$$

then

$$D_i = \frac{\alpha_i M'(y_i F_t(\vec{x}_i))}{z} \tag{3.26}$$

is the *ith* entry of vector $\vec{D}$ and represents the cost of misclassifying the point $\vec{x}_i$.

Finally, we can refactor the supervised training step of the method as finding f_{t+1} that minimizes the total cost of incorrect classifications, or

$$\sum_{i: y_i \neq f_{t+1}(\vec{x}_i)} D_i. \tag{3.27}$$

The resulting algorithm is the **A**daptive **S**emi-**S**upervised ens**EMBLE**.

Algorithm: 5 Assemble

Input:

$\mathcal{H} : \mathcal{X} \to \mathcal{Y}$	:: Supervised Algorithm Classifier
$T \in \mathbb{Z}^+$	:: Positive Integer
$\vec{\alpha} \in \mathbb{R}^{l+u}$	:: Example Weight Vector
$\vec{w} \in \mathbb{R}^T$	:: Classifier Weight Vector
$\mathcal{E} : \mathcal{Y}^{\mathcal{X}} \to \mathbb{R}$	:: Total Error Function
$M : \mathbb{R} \to \mathbb{R}$	:: Individual Error Function
$\vec{D} \in \mathbb{R}^{l+u}$	:: Initial Misclassification Vector

Output:

$\mathcal{C} : \mathcal{X} \to \mathcal{Y}$	:: Classifier

```
 1. let $F_0(\vec{x}) : \mathcal{X} \to \mathbb{R} \leftarrow 0$
 2. let $\mathcal{C} : \mathcal{X} \to \mathcal{Y}$
 3. let $\overline{\mathcal{L}} \leftarrow \mathcal{L}$
 4. for $t \leftarrow 0$ to $T - 1$ do:
 5.         let $f_{t+1} \leftarrow \mathbf{Train}(\mathcal{H}, \overline{\mathcal{L}}, \vec{D})$
 6.         if $\sum_{i=1}^{l+u} D_i y_i f_{t+1}(\vec{x}_i) \leq 0$ do:
 7.             $\mathcal{C} \leftarrow \mathbf{sign}(F_t)$
 8.             return $\mathcal{C}$
 9.         let $F_{t+1} \leftarrow F_t + w_{t+1} f_{t+1}$
10.         $\mathcal{C} \leftarrow \mathbf{sign}(F_{t+1})$
11.         $\overline{\mathcal{L}} \leftarrow \mathcal{L} \cup \{(\vec{x}, \mathcal{C}(\vec{x})) : \vec{x} \in \mathcal{U}\}$
12.         $\vec{D} \leftarrow D_i = \dfrac{\alpha_i M'(y_i F_{t+1}(\vec{x}_i))}{\sum_{j=1}^{l+u} \alpha_j M'(y_j F_{t+1}(\vec{x}_j))}$
13. return $\mathcal{C}$
```

The assemble works almost identically to the first algorithm presented, except for the use of misclassification cost vector $\vec{D}$ in the training of a new classifier. The early halting condition in line 5 is also rewritten in terms of $\vec{D}$ and at the end of each iteration the misclassification vector is updated.

Convergence

Since the algorithm loops at most $T+1$ times, and the condition at line 5 guarantees that every new classifier added will improve the performance of the ensemble, the algorithm is improving at every iteration and will eventually stop and return a classifier.

A potential shortcoming of the method is falling in a bad region of the function space in early iterations, and consequently halting the algorithm too soon with a poor classifier. In such scenario, a new selection of hyperparameters like the supervised algorithm $\mathcal{H}$, or the weight of each classifier w_i, can improve the performance.

Hyperparameters

The Assemble algorithm is a wrapper method and relies on a supervised algorithm $\mathcal{H}$ to train a new classifier. This algorithm can be any method that relies on minimization of a cost function.

Two weight vectors are required, $\vec{w} = \langle w_1, w_2, \ldots, w_T \rangle$ representing the weight that each classifier will have in the final ensemble, and $\vec{\alpha}$ being the weight of a particular example for the training (this allows the algorithm to attribute different levels of importance to labeled and unlabeled examples). For the example weights, it is common to have a constant value for labeled examples (α_l) and another for unlabeled ones (α_u), however, it's possible to have non-constant values with minor modifications required [5]. The classifiers weights can also be determined from inside the algorithm instead of being passed as hyperparameters. It can be shown that choosing w_{t+1} to minimize

$$\sum_{i=1}^{l+u} \alpha_i M(y_i(F_t(\vec{x}_i) + w_{t+1} f_{t+1}(\vec{x}_i))) \tag{3.28}$$

will always lead to a decrease in the cost function whenever such a decrease is possible [5].

Lastly, we don't have the misclassification cost vector $\vec{D}$ for the first iteration, and it must be provided a priori to the start of algorithm.

Variations

The most notorious variation of the Assemble is the Assemble-AdaBoost algorithm. It's an award-winning method from the NIPS'2001 workshop, *Competition: Unlabeled Data for Supervised Learning*, where it outperformed 34 algorithms [5].

Algorithm: 6 Assemble-AdaBoost

Input:

$\overline{\mathcal{L}} \subset \mathcal{X} \times \mathcal{Y}$	:: Labeled Dataset With Pseudo-Labeled Dataset
$\mathcal{H} : \mathcal{X} \to \mathcal{Y}$	:: Supervised Algorithm Classifier
$T \in \mathbb{Z}^+$	:: Positive Integer
$\alpha \in \mathbb{R}$	:: Example Weight
$\beta \in \mathbb{R}$	:: Misclassification Cost Constant
$\mathcal{E} : \mathcal{X}^{\mathcal{Y}} \to \mathbb{R}$	:: Error Function

Output:

$\mathcal{C} : \mathcal{X} \to \mathcal{Y}$	:: Classifier

```
1.  let $F_0(\vec{x}) : \mathcal{X} \to \mathbb{R} \leftarrow 0$
2.  let $\vec{D} \in \mathbb{R}^{\ell+u} \leftarrow D_i = \begin{cases} \beta/l, & \text{if } 1 \le i \le l \\ (1-\beta)/u, & \text{if } l+1 \le i \le l+u \end{cases}$
3.  for $t \leftarrow 0$ to $T-1$ do:
4.          let $f_{t+1} \leftarrow \mathbf{Train}(\mathcal{H}, \overline{\mathcal{L}}, \vec{D})$
5.          let $\epsilon \in \mathbb{R} \leftarrow \sum_{y_i \neq f_{t+1}(\vec{x}_i)} D_i$
6.          if $\epsilon > 0.5$ do return $F_t$
7.          let $w_{t+1} \leftarrow 0.5 \log\left(\frac{1-\epsilon}{\epsilon}\right)$
8.          let $F_{t+1} \leftarrow w_{t+1} f_{t+1}$
9.          let $\vec{y} \leftarrow y_i = \begin{cases} y_i, & \text{if } 1 \le i \le \ell \\ F_{t+1}(\vec{x}_i), & \text{if } \ell+1 \le i \le \ell+u \end{cases}$
10.         let $\vec{D} \leftarrow D_i = \alpha e^{-y_i F_{t+1}(\vec{x}_i)} / \sum_{j=1}^{l+u} \alpha e^{-y_j F_{t+1}(\vec{x}_j)}$
11.         let $\overline{\mathcal{L}} \leftarrow \mathcal{L} \cup \{(\vec{x}_i, y_i) : \vec{x}_i \in \mathcal{U}\}$
12.         $\overline{\mathcal{L}} \leftarrow \mathbf{Sample}(\overline{\mathcal{L}}, \ell, \vec{D})$
13. return $F_T$
```

In the Assemble-AdaBoost we have two new inputs, namely $\overline{\mathcal{L}}$ and β. The first is the union between the labeled dataset $\mathcal{L}$ and the set of points $\{(\vec{x}_i, y_i)\}$ where $\vec{x}_i \in \mathcal{U}$ and y_i is the class of the closest labeled data to the point $\vec{x}_i$. This set is necessary to train the first classifier for the ensemble. The second is the hyperparameter β and is used to skew the importance of the labeled data for the first iteration, represented by the vector $\vec{D}$.

The algorithm begins by starting the ensemble with the zero function and creating the vector $\vec{D}$. The main loop then starts by training a new classifier and using it to predict new classes for all the points. These new labels are compared to the labels given by the ensemble F_t on the previous iterations or from the input $\overline{\mathcal{L}}$ if it's the first loop, summing the misclassification costs of all the labels that doesn't match. If this error ϵ is greater than 0.5 it means that the new classifier is no better than a random guesser and the algorithm stops early returning the current ensemble. Otherwise, we add the classifier to the ensemble with the weight given by the function $0.5 \log\left((1-\epsilon)/\epsilon\right)$. This function rapidly decreases as the error increases, resulting in a heavy penalization of a bad classifier in the final ensemble.

The new ensemble is used to update the pseudo-labels of the points in $\mathcal{U}$ and the vector $\vec{D}$ is also updated as the original Assemble but with α being constant and $M(x) = e^x$.

Finally, we sample l points from the union of the labeled dataset with the now pseudo-labeled dataset with weights $\vec{D}$. This is done to keep the size of the training set equal to the size of the labeled set, that way the algorithm has similar complexity with the original AdaBoost.

3.5 SemiBoost

SemiBoost is another boosting algorithm, like Assemble, whose objective is to improve the performance of a supervised algorithm $\mathcal{H}$ through an ensemble of these weaker classifiers, while also extracting information from the unlabeled examples. SemiBoost is exclusively used for binary classification tasks, and is unique in its approach to incorporate the unlabeled data, drawing inspiration from graph-based methods with its modeling of the similarity between points by a matrix S. The algorithm, like many others, works by minimizing an error function, and as such, requires a way to measure the error of wrong classifications. According to [22], the assignment of the pseudo-labels must follow two criteria:

1. Unlabeled examples highly similar to one another must share a label
2. Unlabeled examples highly similar to a labeled point must share its label

The error associated with each of these criteria is encapsulated in an individual cost function, with the total error being the sum of these terms. Let $\vec{y} = (y_1, y_2, \ldots, y_l, y_{l+1}, \ldots, y_{l+u})$ represent the vector of labels, with y_i being the label of point $\vec{x}_i$. The inconsistency between unlabeled examples is quantified by the function:

$$\mathcal{E}_u(\vec{y}, S) = \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} S_{ij} e^{y_i - y_j}. \tag{3.29}$$

Considering that the problem is binary, a pair of labels (y_i, y_j) for two unlabeled points can take on 4 values, namely $(1,1), (-1,-1), (1,-1), (-1,1)$. Since the double summation will pass over each pair twice, and considering S a symmetric matrix, for each possible pair their, term in the summation will be:

$$\begin{cases} (y_i, y_j) = (1,1) & \rightarrow S_{ij}e^0 + S_{ji}e^0 = 2S_{ij} \\ (y_i, y_j) = (-1,-1) & \rightarrow S_{ij}e^0 + S_{ji}e^0 = 2S_{ij} \\ (y_i, y_j) = (1,-1) & \rightarrow S_{ij}e^{1+1} + S_{ji}e^{-1-1} = S_{ij}(e^2 + e^{-2}) \approx 7.5S_{ij} \\ (y_i, y_j) = (-1,1) & \rightarrow S_{ij}e^{-1-1} + S_{ji}e^{1+1} = S_{ij}(e^2 + e^{-2}) \approx 7.5S_{ij} \end{cases} \tag{3.30}$$

We can see that for two highly similar unlabeled points (high S_{ij}) the cost of misclassification ($7.5S_{ij}$) is around 3.75 times higher than if they were correctly labeled ($2S_{ij}$).

The function $\mathcal{E}_u(\vec{y}, S)$ then sums the costs over all possible pairs of unlabeled points. The inconsistency between the labeled and unlabeled points are given by:

$$\mathcal{E}_{lu}(\vec{y}, S) = \sum_{i=1}^{l} \sum_{j=l+1}^{l+u} S_{ij} e^{-2y_i y_j}. \tag{3.31}$$

Evaluating the cost of a pair (y_i, y_j) between a labeled $(\vec{x}_i)$ and unlabeled $(\vec{x}_j)$ point we would get $S_{ij}e^{-2} \approx 0.14S_{ij}$ for a shared label, and $S_{ij}e^{2} \approx 7.4S_{ij}$ otherwise. Both $\mathcal{E}_u$ and $\mathcal{E}_{lu}$ heavily punishes wrong classifications of similar points, contributing to the goals given in criteria 1 and 2.

The final error function is

$$\mathcal{E}(\vec{y}, S) = \mathcal{E}_{lu}(\vec{y}, S) + \gamma \mathcal{E}_u(\vec{y}, S) \tag{3.32a}$$

$$\mathcal{E}(\vec{y}, S) = \sum_{i=1}^{l} \sum_{j=l+1}^{l+u} S_{ij} e^{-2y_i y_j} + \gamma \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} S_{ij} e^{y_i - y_j} \tag{3.32b}$$

where γ represents the weight given to the unlabeled errors.

We are now ready to formulate the optimization loop for this problem. Consider the ensemble $F_t(\vec{x})$ with t classifiers, entering the loop $t+1$ we want to find the function $f_{t+1}(\vec{x})$ and the weight w_{t+1} that will minimize the function $\mathcal{E}$. Considering $y_i = F_t(\vec{x}_i) + w_{t+1}f_{t+1}(\vec{x}_i)$ for $\vec{x}_i \in \mathcal{U}$, we refactor the optimization problem according to *Proposition 1.*

Proposition 1.5.1: Minimizing Eq. (3.32b) is equivalent to minimizing the function

$$\mathcal{E}'(\vec{y}, S) = \sum_{i=l+1}^{l+u} e^{-2w_{t+1}f_{t+1}(\vec{x}_i)} p_i + e^{2w_{t+1}f_{t+1}(\vec{x}_i)} q_i \tag{3.33}$$

where

$$p_i = \sum_{j=1}^{l} S_{ij} e^{-2F_t(\vec{x}_i)} \delta(y_j, 1) + \frac{\gamma}{2} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j) - F_t(\vec{x}_i)} \tag{3.34}$$

$$q_i = \sum_{j=1}^{l} S_{ij} e^{2F_t(\vec{x}_i)} \delta(y_j, -1) + \frac{\gamma}{2} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_i) - F_t(\vec{x}_j)} \tag{3.35}$$

$$\delta(a, b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases} \tag{3.36}$$

Proof: (see Appendix A.1)

Since f_{t+1} and w_{t+1} occur together in the expression, the optimization is difficult. However, we can minimize it by attempting to minimize an upper bound as stated in the next proposition.

Proposition 1.5.2: The function $\mathcal{E}'(\vec{y}, S)$ in Eq. (3.33) has the following upper bound:

$$\mathcal{E}'(\vec{y}, S) \leq \sum_{i=l+1}^{l+u} (p_i + q_i)(e^{2w_{t+1}} + e^{-2w_{t+1}} - 1) - \sum_{i=l+1}^{l+u} 2w_{t+1}f_{t+1}(\vec{x}_i)(p_i - q_i). \tag{3.37}$$

Proof: (see Appendix A.2)

Let the right hand-side of Eq. (3.37) be $\mathcal{E}''(\vec{y}, S)$.

Corollary 1.5.3: To minimize $\mathcal{E}''(\vec{y}, S)$ the predicted label of a point $\vec{x}_i$ should be $z_i = sign(p_i - q_i)$, and the sampled points should be the ones that give a high value of $|p_i - q_i|$.

Proof: From Eq. (3.37) we see that only the second term contains f_{t+1}, and since w_{t+1} is positive, the smallest possible term for a point $\vec{x}_i$ is when $f_{t+1}(\vec{x}_i)(p_i - q_i) > 0$, or equivalently $f_{t+1}(\vec{x}) = sign(p_i - q_i)$, since f_{t+1} can only take the values 1 and -1. Also, the higher the value of $|p_i - q_i|$ the higher the decrease in the function $\mathcal{E}''$. Therefore, the points with high $|p_i - q_i|$ are more desirable to be used in the next training iteration. ■

Lastly, we calculate the value of w_{t+1} according to the following proposition.

Proposition 1.5.4: The value of w_{t+1} that minimizes $\mathcal{E}'$ is

$$w_{t+1} = \frac{1}{4} \ln \left(\frac{\sum\limits_{i=l+1}^{l+u} p_i \delta(f_{t+1}(\vec{x}_i), 1) + q_i \delta(f_{t+1}(\vec{x}_i), -1)}{\sum\limits_{i=l+1}^{l+u} p_i \delta(f_{t+1}(\vec{x}_i), -1) + q_i \delta(f_{t+1}(\vec{x}_i), 1)} \right). \tag{3.38}$$

Proof: (see Appendix A.3)

So far, we devised a way to calculate w_{t+1} and we know which label $\overline{y}_i$ is optimal for a point $\vec{x}_i$ to have so the error is minimized. Now we only need to find the classifier f_{t+1} that will best predict these desired labels. To do this we construct the training set for the algorithm $\mathcal{H}$ as the union between the labeled dataset and sampling a fraction λ from the set of pseudo-labeled points $\overline{\mathcal{U}} = \{(\vec{x}_{l+1}, z_{l+1}), \ldots, (\vec{x}_{l+u}, z_{l+u})\}$, with each example having a weight $|p_i - q_i|$ as discussed in *Corollary 1*.

Before determining f_{t+1}, we check if it's worth to add it to the ensemble by checking if $w_{t+1} > 0$, in which case the new classifier will improve the ensemble. Otherwise, we stop the algorithm and return the current ensemble. This finalizes the loop $t + 1$. The algorithm then continues for a total of T iterations or until the condition $w_{t+1} \leq 0$ is met.

Algorithm: 7 SemiBoost

Input:

$\mathcal{H} : \mathcal{X} \to \mathcal{Y}$	:: Supervised Algorithm Classifier
$S \in \mathbb{R}^{(l+u)\times(l+u)}$	:: Similarity Matrix
$T \in \mathbb{Z}^{+}$	:: Number of Classifiers
$\gamma \in \mathbb{R}$	:: Unlabeled Inconsistency Weight
$\lambda \in \mathbb{R}$	:: Proportion of examples to sample

Output:

$\mathcal{C} : \mathcal{X} \to \mathcal{Y}$	:: Classifier

```
1.  let $F_0(\vec{x}) : \mathcal{X} \to \mathbb{R} \leftarrow 0$
2.  let $\vec{p}, \vec{q}, \vec{z}, \vec{D} \in \mathbb{R}^{l+u}$
3.  let $\delta(a,b) : \mathcal{Y}^2 \to \{0,1\} \leftarrow \begin{cases} 1, & \text{if } a = b \\ 0, & \text{if } a \neq b \end{cases}$
4.  for $t \leftarrow 0$ to $T-1$ do:
5.          $\vec{p} \leftarrow p_i = \sum_{j=1}^{l} S_{ij} e^{-2F_t(\vec{x}_i)} \delta(y_j, 1) + \frac{\gamma}{2} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j) - F_t(\vec{x}_i)}$
6.          $\vec{q} \leftarrow q_i = \sum_{j=1}^{l} S_{ij} e^{2F_t(\vec{x}_i)} \delta(y_j, -1) + \frac{\gamma}{2} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_i) - F_t(\vec{x}_j)}$
7.          $\vec{z} \leftarrow z_i = sign(p_i - q_i)$
8.          $\vec{D} \leftarrow D_i = |p_i - q_i|$
9.          let $\mathcal{U}' \leftarrow$ Sample$(\{(\vec{x}_i, z_i) : \ell + 1 \leq i \leq \ell + u\}, \lambda, \vec{D})$
10.         let $\overline{\mathcal{L}} \leftarrow \mathcal{L} \cup \mathcal{U}'$
11.         let $w_{t+1} \leftarrow \frac{1}{4} \ln \left( \frac{\sum_{i=l+1}^{l+u} (p_i \delta(z_i, 1) + q_i \delta(z_i, -1))}{\sum_{i=l+1}^{l+u} (p_i \delta(z_i, -1) + q_i \delta(z_i, 1))} \right)$
12.         if $w_{t+1} \leq 0$ do return $F_t$
13.         let $f_{t+1} \leftarrow$ Train$(\mathcal{H}, \overline{\mathcal{L}})$
14.         let $F_{t+1} \leftarrow F_t + w_{t+1} f_{t+1}$
15. return $F_T$
```

The algorithm starts by setting the ensemble to the zero function and declaring the utility function δ. Next, we start the main loop of T iterations and calculate the entries of the utility vectors $\vec{p}, \vec{q}$, the pseudo-label vector $\vec{z}$, and the sampling weight vector $\vec{D}$, for each point.

We then sample with weights $\vec{D}$ a fraction λ from the unlabeled dataset and their pseudo-labels $\vec{z}$, constructing the training set $\mathcal{T}$ as the union from it and the original labeled set.

The optimal weight w_{t+1} is calculated and checked to see if a new classifier is disadvantageous to the ensemble, in which case the algorithm stops and returns the current ensemble. Otherwise, the new classifier is trained and added to the ensemble with the calculated weight. If the early stop condition is never met, the algorithm continues and returns the ensemble F_T.

Convergence

To determine the convergence of the algorithm we rely on the following proposition

Proposition 1.5.5: Let $\mathcal{E}'_{t+1}(\vec{y}, S)$ be the upper error function at iteration $t+1$, then

$$\mathcal{E}'_{t+1}(\vec{y}, S) \leq \mathcal{E}'_0(\vec{y}, S) e^{-\sum\limits_{i=1}^{t+1} 2w_i}, \tag{3.39}$$

where

$$\mathcal{E}'_0(\vec{y}, S) = \sum_{i=l+1}^{l+u} \left[\sum_{j=1}^{l} S_{ij} + \gamma \sum_{j=l+1}^{l+u} S_{ij} \right]. \tag{3.40}$$

Proof: (see Appendix A.4).

This shows that the upper bound error function follows a exponential decay from its initial value, decreasing with each iteration and consequent addition of a new classifier k with its positive weight w_k.

Hyperparameters

As an ensemble method, the first hyperparameter to be considered is the number of classifiers T. It has been shown that a T of around 20 provides a good performance [12].

The similarity matrix conveys the necessary information to satisfy the two criteria upon which the algorithm is based. The standard approach is to define S with the rbf kernel, similar to the graph based methods [22], that is

$$S_{ij} = \exp\left(-\frac{||\vec{x}_i - \vec{x}_j||^2}{\sigma^2}\right). \tag{3.41}$$

This introduces a new hyperparameter to be determined. The value σ regulates how quickly the similarity diminishes with the Euclidean distance between points, with high values limiting the similarity to only the closest points, and low values allowing the influence of points to spawn further across the input space.

The hyperparameter γ dictates the importance of the unlabeled-unlabeled error for the error function. Once the number of unlabeled examples can overwhelm the labeled ones, it is important to scale down this term of the error function to maintain the labeled-unlabeled inconsistency relevant to the minimization problem. A valid heuristic is setting $\gamma = l/u$ [22].

The selection of which unlabeled points to pick for the next iteration is a delicate step. Allowing too few points won't allow the method to learn from them. On the other hand, as they are pseudo-labeled by decreasingly weaker classifiers, some labels are prone to error and will harm the learning process. This is solved by sampling a fraction λ of these points by their weight $|p_i - q_i|$, as it was shown in *Corollary 1.* The empirical value of $\lambda = 0.1$ provides good results [22].

Variations

The main variations of the SemiBoost algorithm are given by different choices of base classifiers. The Decision Stump, Decision Tree and SVMs are among the base classifiers that were shown to significantly have their performances improved by the SemiBoost method [22].

3.6 Co-Training

The Co-Training algorithm was first proposed by Blum and Mitchel [7] in an attempt to classify the contents of web-pages from the information contained within the page itself, as well as the text on pages with a hyperlink pointing to it. Later renaming these types of information as different "views" of the input space, in their proposed framework for a learning algorithm that can take advantage of such a naturally split input space.

Suppose that the input space can be written as the Cartesian product of two subspaces $\mathcal{X}_A$ and $\mathcal{X}_B$, that is $\mathcal{X} = \mathcal{X}_A \times \mathcal{X}_B$. In this case, any point in $\mathcal{X}$ has the form $\vec{x} = (\vec{x}_A, \vec{x}_B)$, with $\vec{x}_A \in \mathcal{X}_A$ and $\vec{x}_B \in \mathcal{X}_B$. The idea behind the Co-Training algorithm is to train a classifier in each of these subspaces iteratively, and create a final classifier composed of these two weaker classifiers, now specialized in a single view of the input space. This is only possible if we assume a conditional independence between the labeling from classifier A and B. For this training approach to be effective, both $\mathcal{X}_A$ and $\mathcal{X}_B$ have to be enough on their own to predict the labels by themselves, while also not being so tightly correlated that the information in one subspace is redundant and unhelpful [7].

To construct the combined classifier we take advantage of the conditional independence assumption. The probability of a point having the label m then becomes

$$P(\text{Label}(\vec{x}_A, \vec{x}_B) = m) = P(\text{Label}(\vec{x}_A) = m)P(\text{Label}(\vec{x}_B) = m). \tag{3.42}$$

However, the values $P(\text{Label}(\vec{x}_A) = m)$ and $P(\text{Label}(\vec{x}_B) = m)$ are given by the m-th entry on $f_A(\vec{x})$ and $f_B(\vec{x})$, respectively.

$$P(\text{Label}(\vec{x}_A) = m) = (f_A(\vec{x}))_m \tag{3.43}$$

$$P(\text{Label}(\vec{x}_B) = m) = (f_B(\vec{x}))_m. \tag{3.44}$$

Since the left hand-side of Eq. (3.42) is the m-th entry of the desired classifier $\mathcal{C}$, for the binary classification task we have

$$\overline{\mathcal{C}}(\vec{x}) = ((f_A(\vec{x}))_1(f_B(\vec{x}))_1, (f_A(\vec{x}))_2(f_B(\vec{x}))_2) \tag{3.45}$$

$$\overline{\mathcal{C}}(\vec{x}) = f_A(\vec{x}) \odot f_B(\vec{x}) \tag{3.46}$$

where $\odot$ represents the element-wise or Hadamard product. To maintain the probabilistic interpretation, we re-normalize the vector to obtain the final classifier

$$\mathcal{C}(\vec{x}) = f_A(\vec{x}) \odot f_B(\vec{x})/||f_A(\vec{x}) \odot f_B(\vec{x})|| \tag{3.47}$$

Algorithm: 8 Co-Training (Binary)

Input:

$\mathcal{H} : \mathcal{X} \to \mathcal{Y}$	:: Supervised Algorithm Classifier
$T \in \mathbb{Z}^+$	:: Number of Iterations
$p \in \mathbb{Z}^+$	:: Number of Positively Labeled Examples to Select
$n \in \mathbb{Z}^+$	:: Number of Negatively Labeled Examples to Select
$u' \in \mathbb{Z}$	:: Unlabeled Examples Buffer

Output:

$\mathcal{C} : \mathcal{X} \to \mathbb{R}^{\|\mathcal{Y}\|}$	:: Probabilistic Classifier

```
1.  let $\overline{\mathcal{U}} \leftarrow$ Sample$(\mathcal{U}, u')$
2.  let $\overline{\mathcal{L}} \leftarrow \mathcal{L}$
3.  let $k \in \mathbb{N} \leftarrow 1$
4.  while $k \leq T$ and $\overline{\mathcal{U}} \neq \emptyset$ do:
5.          let $f_A(\vec{x}) : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|} \leftarrow$ Train$(\mathcal{H}, \overline{\mathcal{L}}, \mathcal{X}_A)$
6.          let $f_B(\vec{x}) : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|} \leftarrow$ Train$(\mathcal{H}, \overline{\mathcal{L}}, \mathcal{X}_B)$
7.          let $P_A \leftarrow$ Select$(f_A, p, n, \overline{\mathcal{U}})$
8.          let $P_B \leftarrow$ Select$(f_B, p, n, \overline{\mathcal{U}})$
9.          $\overline{\mathcal{L}} \leftarrow \overline{\mathcal{L}} \cup \{(\vec{x}_i, f_A(\vec{x}_i)) : \vec{x}_i \in P_A\} \cup \{(\vec{x}_i, f_B(\vec{x}_i)) : \vec{x}_i \in P_B\}$
10.         $\overline{\mathcal{U}} \leftarrow \overline{\mathcal{U}} \cup$ Sample$(\mathcal{U}, 2p + 2n)$
11.         $k \leftarrow k + 1$
12. let $\mathcal{C} \leftarrow f_A(\vec{x}) \odot f_B(\vec{x}) / ||f_A(\vec{x}) \odot f_B(\vec{x})||$
13. return $\mathcal{C}$
```

The algorithm begins creating a buffer unlabeled set $\overline{\mathcal{U}}$ by sampling u' examples from the original unlabeled dataset, and also creates the training set as the labeled dataset for the first iteration.

The main loop starts for T iterations or until there are no more unlabeled examples to be classified. Two new probabilistic classifiers are trained from the training set $\overline{\mathcal{L}}$, taking into account only their respective view. We then call the function **Select**[1] that will apply a classifier to the buffer set $\overline{\mathcal{U}}$ and select the p most confident positive predictions and its n most confident negative ones, returning a set containing the inputs that generated these confident predictions. We consider that the function removes these examples from the set $\overline{\mathcal{U}}$, and after two calls the buffer set now has size $u' - (2p + 2n)$. The training set

[1]A function **Select** that works as intended is proposed in Appendix B.1

is then updated to contain these confident examples and their predicted labels, and the buffer unlabeled set is refilled with $2p + 2n$ sampled without reposition from $\mathcal{U}$.

After the main loop is finished we have the final probabilistic classifiers for each view. We then construct the combined classifier as the re-normalized Hadamard product between f_A and f_B.

Convergence

As a wrapper method, the convergence of the Co-Training algorithm is dependent on the base classifiers. Blum and Mitchel [7] showed that if f_A and f_B are learnable even with classification noise, and the conditional independence assumption is satisfied, then $\mathcal{C}$ is also learnable.

Hyperparameters

The first hyperparameter T dictates the maximum number of iterations the main loop should perform. It serves as an early stopping criteria in case we don't want to use all of the unlabeled dataset.

The values of p and n are passed as a measure to even the number of positive and negative examples drawn in the selection process, with its values usually following the ratio of the positive-negative in the underlying distribution [7].

Lastly, u dictates the size of the buffer dataset, regulating the number of examples to be evaluated by the single view classifiers in each iteration.

Proposition 1.6.1: If higher predictions from the single view classifiers lead to better results, then higher values of u can only improve the performance.

Proof: All a higher value of u can do is allow for extra unlabeled examples to be drawn. In case these extra examples result in a less confident prediction, they will simple be ignored. Otherwise, they will be picked, and as higher confident examples they will lead to better results. $\blacksquare$

However, higher values of u can impact the speed of the algorithm. Considering that each call to **Select** has to order the examples in $\overline{\mathcal{U}}$ twice, one for the positive and one for the negative selection, the cost impact of u' over T iterations is $\mathcal{O}(Tu' \log u')$. Therefore, the trade-off between performance and training time needs to be considered, with a hyperparameter search being recommended.

Variations

A noteworthy variation is proposed by Nigam and Ghani [25], where a few adjustments are performed to adapt the Co-Training algorithm to a multi-class scenario.

Algorithm: 9 Co-Training

```
Input:
$\mathcal{H} : \mathcal{X} \to \mathcal{Y}$          :: Supervised Algorithm Classifier
$T \in \mathbb{Z}^+$                 :: Number of Iterations
$\vec{v} \in \mathbb{Z}^{|\mathcal{Y}|}$               :: Number of Examples To Select For Each Class
$u' \in \mathbb{Z}$                  :: Unlabeled Examples Buffer

Output:
$\mathcal{C} : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|}$        :: Probabilistic Classifier
---------------------------------------------------------------
 1. let $\overline{\mathcal{U}} \leftarrow$ Sample$(\mathcal{U}, u')$
 2. let $\overline{\mathcal{L}} \leftarrow \mathcal{L}$
 3. let $k \in \mathbb{N} \leftarrow 1$
 4. while $k \leq T$ and $\overline{\mathcal{U}} \neq \emptyset$ do:
 5.         let $f_A(\vec{x}) : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|} \leftarrow$ Train$(\mathcal{H}, \overline{\mathcal{L}}, \mathcal{X}_A)$
 6.         let $f_B(\vec{x}) : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|} \leftarrow$ Train$(\mathcal{H}, \overline{\mathcal{L}}, \mathcal{X}_B)$
 7.         let $P_A \leftarrow$ Select$(f_A, \vec{v}, \overline{\mathcal{U}})$
 8.         let $P_B \leftarrow$ Select$(f_B, \vec{v}, \overline{\mathcal{U}})$
 9.         $\overline{\mathcal{L}} \leftarrow \overline{\mathcal{L}} \cup \{(\vec{x}_i, f_A(\vec{x}_i)) : \vec{x}_i \in P_A\} \cup \{(\vec{x}_i, f_B(\vec{x}_i)) : \vec{x}_i \in P_B\}$
10.         $\overline{\mathcal{U}} \leftarrow \overline{\mathcal{U}} \cup$ Sample$(\mathcal{U}, 2\sum_{i=1}^{|\mathcal{Y}|} v_i)$
11.         $k \leftarrow k + 1$
12. let $\mathcal{C} : \mathcal{X} \to \mathbb{R}^{|\mathcal{Y}|} \leftarrow f_A(\vec{x}) \odot f_B(\vec{x}) / ||f_A(\vec{x}) \odot f_B(\vec{x})||$
13. return $\mathcal{C}$
```

The only difference in the algorithm is the necessity to pass the number of examples from each class to select. This is done using the vector $\vec{v}$, whose *ith* entry represents the number of examples to be drawn for class i. A new function **Select'**[2] is used to perform this new task. Lastly, the sampling to refill the buffer set $\overline{\mathcal{U}}$ needs to restore a total of $2\sum_{i=1}^{|\mathcal{Y}|} v_i$ examples.

[2]A function **Select'** is proposed in Appendix B.2

3.7 Tri-Training

The Tri-Training algorithm was proposed by Zhou [39], inspired by the Co-Training and Self-Training methods. The algorithm functions by training three base-classifiers at each iteration, while selecting new examples to be pseudo-labeled for a classifier as the points where the other two classifiers agree on a labeling. This allows Tri-Training to be less restrictive on the type of base-classifiers used, since it lifts the requirement of it being probabilistic. Furthermore, the agreement-base pseudo-labeling is less computationally expensive than the Co-Training approach, once it doesn't require the computation of probabilities for each example and their ordering so the most confident points can be selected.

Let $f_{i,t}$ be the classifier i at iteration t. The training set $\mathcal{T}_{i,t}$ of classifier i at iteration t is constructed as the union of the original labeled set $\mathcal{L}$ and the set of pseudo-labeled examples from $\mathcal{U}$, where the other two classifiers agree on a label. Let this second set be denoted by $\mathcal{L}_{i,t}$, we have

$$\mathcal{L}_{i,t} = \{(\vec{x}, y) : \vec{x} \in \mathcal{U}, y = f_{j,t}(\vec{x}) = f_{k,t}(\vec{x}), i \neq j \neq k\}. \tag{3.48}$$

Yielding the final training set for classifier i as

$$\mathcal{T}_{i,t} = \mathcal{L} \cup \mathcal{L}_{i,t}. \tag{3.49}$$

Now with a training set, we need to check if this training process will yield a better classifier. If the pseudo-labeling by the other classifiers is correct, then we have a valid example for the next training loop. Otherwise, we will get a noisy and unhelpful label. According to Angluin and Laird [3], even accounting for wrongfully labeled examples in the training set, the learning process can be achieved if enough new examples are used. Inspired by their work, Zhou [39] shows that the number of examples ℓ_i needed by classifier i is

$$\ell_i = \frac{c}{\epsilon_i^2(1-2\eta_i)^2}, \tag{3.50}$$

where ϵ_i is the upper-bound fraction of wrongfully labeled examples by the classifier i, η_i is the upper-bound fraction of wrongfully labeled examples in the training set for i, and c is a positive constant. In other words, if the training set of classifier i has ℓ_i examples, then the error rate will not exceed ϵ_i.

We then compare the upper-bound of the error between iterations, with the desired result being $\epsilon_{i,t} < \epsilon_{i,t-1}$.

Isolating the classifier error in Eq. (3.50), we obtain:

$$\epsilon_{i,t} = \frac{\sqrt{c/|\mathcal{L} \cup \mathcal{L}_{i,t}|}}{(1 - 2\eta_{i,t})}. \tag{3.51}$$

First, we need a way to estimate the fraction $\eta_{i,t}$ of wrongfully classified examples in the training set. Let η_L be this fraction for the set $\mathcal{L}$, and let $e_{i,t}$ be the wrongfully labeled examples in $\mathcal{L}_{i,t}$, that is, misclassifications that j and k agreed on. With this we have the total number of misclassified examples in $\mathcal{T}_{i,t}$, namely $\eta_L|\mathcal{L} + e_{i,t}|\mathcal{L}_{i,t}|$. Therefore, the desired fraction becomes

$$\eta_{i,t} = \frac{\eta_L|\mathcal{L}| + e_{i,t}|\mathcal{L}_{i,t}|}{|\mathcal{L} \cup \mathcal{L}_{i,t}|}. \tag{3.52}$$

Substituting Eq. (3.52) into Eq. (3.51)

$$\epsilon_{i,t} = \frac{\sqrt{c/|\mathcal{L} \cup \mathcal{L}_{i,t}|}}{\left(1 - 2\frac{\eta_L|\mathcal{L}| + e_{i,t}|\mathcal{L}_{i,t}|}{|\mathcal{L} \cup \mathcal{L}_{i,t}|}\right)}. \tag{3.53}$$

Finally, the inequality that we want to satisfy is

$$\frac{\sqrt{c/|\mathcal{L} \cup \mathcal{L}_{i,t}|}}{\left(1 - 2\frac{\eta_L|\mathcal{L}| + e_{i,t}|\mathcal{L}_{i,t}|}{|\mathcal{L} \cup \mathcal{L}_{i,t}|}\right)} < \frac{\sqrt{c/|\mathcal{L} \cup \mathcal{L}_{i,t-1}|}}{\left(1 - 2\frac{\eta_L|\mathcal{L}| + e_{i,t-1}|\mathcal{L}_{i,t-1}|}{|\mathcal{L} \cup \mathcal{L}_{i,t-1}|}\right)}. \tag{3.54}$$

To solve this inequality we rely on the following proposition:

Proposition 1.7.1: if $|\mathcal{L}_{i,t-1}| < |\mathcal{L}_{i,t}|$ and $e_{i,t}|\mathcal{L}_{i,t}| < e_{i,t-1}|\mathcal{L}_{i,t-1}|$, then inequality (3.54) is satisfied.

Proof: (see Appendix A.6).

Let $|\mathcal{L}_{i,t-1}| < |\mathcal{L}_{i,t}|$ be condition 1 and $e_{i,t}|\mathcal{L}_{i,t}| < e_{i,t-1}|\mathcal{L}_{i,t-1}|$ be condition 2. It is possible that $|\mathcal{L}_{i,t}|$ is big enough to satisfy condition 1 and violate condition 2, depending on the values of $e_{i,t}$ and $e_{i,t-1}$. In such cases, we can randomly subsample $\mathcal{L}_{i,t}$ to decrease its size until it satisfies condition 2 while being careful so it remains bigger than $\mathcal{L}_{i,t-1}$. Reorganizing condition 2 we have:

$$|\mathcal{L}_{i,t}| < \frac{e_{i,t-1}|\mathcal{L}_{i,t-1}|}{e_{i,t}} \tag{3.55}$$

We choose the new size $\ell'_{i,t}$ of $\mathcal{L}_{i,t}$ as the biggest integer that still satisfies (3.55).

$$\ell'_{i,t} = *\frac{e_{i,t-1}|\mathcal{L}_{i,t-1}|}{e_{i,t}} - 1 \tag{3.56}$$

Proposition 1.7.2: If $|\mathcal{L}_{1,t-1}| > \dfrac{e_{1,t}}{e_{1,t-1} - e_{1,t}}$ then the new size $\ell'_{i,t}$ of $\mathcal{L}_{i,t}$ will also satisfy condition 1.

Proof: (see Appendix A.7)

With this, we have a way to construct the training set for each classifier and can guarantee that its error decreases with each iteration.

When dealing with multiple classifier, it is important to guarantee that the classifiers are different enough, otherwise the algorithm will degenerate into the Self-Training method. In the Co-Training approach, the diversity in the classifiers is achieved from the two different training sets, or views, used. To achieve this diversity in Tri-Training, we create the first training set for the classifiers by a bootstrap sampling of the labeled set $\mathcal{L}$. The classifiers trained on these sets will be sufficiently different, and will be improved at every iteration by the proposed algorithm.

Algorithm: 10 Tri-Training

Input:
$\mathcal{H} : \mathcal{X} \to \mathcal{Y}$ :: Supervised Algorithm Classifier

Output:
$\mathcal{C} : \mathcal{X} \to \mathcal{Y}$ :: Classifier

```
1.  for $i \leftarrow 1$ to 3 do:
2.      let $\mathcal{T}_{i,0} \leftarrow$ BootstrapSample($\mathcal{L}$)
3.      let $f_{i,0} \leftarrow$ Train($\mathcal{H}, \mathcal{T}_{i,0}$)
4.      let $e_{i,0} \in \mathbb{R} \leftarrow 0.5$, $\ell_{i,0} \in \mathbb{N} \leftarrow 0$, $u_i \in \mathbb{Z} \leftarrow 1$
5.  while $(u_1 = 1$ or $u_2 = 1$ or $u_3 = 1)$ do:
6.      for $i \leftarrow 1$ to 3 do:
7.          $u_i \leftarrow 0$
8.          let $e_{i,t} \in \mathbb{R} \leftarrow$ Error($f_{j,t}, f_{k,t}, \mathcal{L}$)
9.          if $e_{i,t} < e_{i,t-1}$ do:
10.             let $\mathcal{L}_{i,t} \leftarrow \{(\vec{x}, y) : \vec{x} \in \mathcal{U}, y = f_{j,t-1}(\vec{x}) = f_{k,t-1}(\vec{x}), i \neq j \neq k\}$
11.             let $\ell_{i,t} \leftarrow |\mathcal{L}_{i,t}|$
12.             if $\ell_{i,t-1} = 0$ do $\ell_{i,t-1} \leftarrow \lfloor e_{i,t}/(e_{i,t-1} - e_{i,t}) + 1 \rfloor$
13.             if $\ell_{i,t-1} < \ell_{i,t}$ do:
14.                 if $e_{i,t}\ell_{i,t} < e_{i,t-1}\ell_{i,t-1}$ do $u_i \leftarrow 1$
15.                 else if $\ell_{i,t-1} > \dfrac{e_{i,t}}{e_{i,t-1} - e_{i,t}}$ do:
16.                     $\mathcal{L}_{i,t} \leftarrow$ Subsample($\mathcal{L}_{i,t}, \lceil e_{i,t-1}\ell_{i,t-1}/e_{i,t} - 1 \rceil$)
17.                     $u_i \leftarrow 1$
18.     for $i \leftarrow 1$ to 3 do:
19.         if $u_i = 1$ do $f_{i,t} \leftarrow$ Train($\mathcal{H}, \mathcal{L} \cup \mathcal{L}_{i,t}$)
20.         if $u_i = 0$ do $f_{i,t} \leftarrow f_{i,t-1}$
21. let $\mathcal{C} : \mathcal{X} \to \mathcal{Y} \leftarrow \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \sum_{\substack{f_{i,t}(\vec{x})=y \\ i \in \{1,2,3\}}} 1$
22. return $\mathcal{C}$
```

The algorithm starts by creating the initial training set from a bootstrapping [9] of

$\mathcal{L}$, the function **BootstrapSample**[3] is called to perform this task. Next, we set the initial classifier error rate $e_{i,0}$ to its maximum theoretical value 0.5, and the size of the pseudo-label set to 0.

The main loop starts and will run while all three classifiers are not updated within a single loop. For each classifier we start by setting a variable u_i to 0 to signal if the classifier will be updated. Here we need a way to estimate the error rate of every pair of classifiers. In his work, Zhou [39] suggests that assuming the unlabeled examples hold the same distribution as the labeled ones, we can estimate this error only by the labeled dataset. This is done by calling the function **Error**[4].

With the new error value, we check if it's smaller than the error of the previous iteration. If this check fails, we continue to the beginning of the next iteration. Otherwise, we construct the pseudo-labeled dataset $\mathcal{L}_{i,t}$ for each classifier and calculate its size $\ell_{i,t}$. If we're on the first iteration, then the previous value of ℓ will be zero as set on line 4, we then update it to the smallest integer that still satisfies the condition in *Proposition 1.7.2*.

To determine if the new dataset is to be used to train a new classifier, we first check if condition 1 ($\ell_{i,t-1} < \ell_{i,t}$) is satisfied and next condition 2 ($e_{i,t}\ell_{i,t} < e_{i,t-}\ell_{i,t-1}$). If both are satisfied, we signal that a new classifier can be trained from the new set by setting u_i equal to 1. However, if condition 2 is initially violated, we can try to subsample $\mathcal{L}_{i,t}$ into an appropriate size. For this to be possible, we check if the condition on *Proposition 1.7.2* is satisfied. If so, we call the function **Subsample**[5] that will receive a set $\mathcal{L}_{i,t}$ and sample $\lceil e_{i,t-1}\ell_{i,t-1}/e_{i,t} - 1\rceil)$ elements without reposition from it, updating u_i right after.

Once we calculated $\mathcal{L}_{i,t}$ and u_i, we train a new classifier i from the training set $\mathcal{L} \cup \mathcal{L}_{i,t}$ if $u_i = 1$. Otherwise, the dataset would lead to a worst classifier, so we maintain the previous one for the next iteration.

Finishing the main loop, we construct the final classifier $\mathcal{C}$ as the majority vote from the three classifiers. Here we can assume that for a multi-classification problem, if each classifier gives a different label, a random one is chosen.

Convergence

The algorithm is designed to guarantee that a classifier i, at iteration t, has a bounded error of $\epsilon_{i,t}$. Therefore, at each loop, an improved classifier is trained or the previous one is maintained, assuring the process is consistent and converging to a better classifier.

Hyperparameters

The Algorithm receives no hyperparameters beyond the choice of base classifier $\mathcal{H}$.

[3]See Appendix B.3.
[4]See Appendix B.4.
[5]see Appendix B.5.

Variations

As a part of the wrapper family, the Tri-Training algorithm supports the natural variations that arise from different choices of base classifiers. In his work, Zhou [39] uses J4.8 decision trees, BP neural networks and Naive Bayes, showing promising results with Tri-Training performing better than Co-Training and Self-Training on a number of datasets.

3.8 TSVM

The Transductive Support Vector Machine algorithm was initially proposed by Vapnik [35], and improved upon by Joachims [19]. This method behaves similarly to wrapper methods, since it uses an underlying supervised algorithm, while incorporating unlabeled data in the training process through a pseudo-labeling process.

The standard soft-margin inductive SVM for a binary classification problem is given by the following minimization problem:

$$\min_{\vec{w},b,\xi_1,\dots,\xi_\ell} \left(\frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{\ell} \xi_i \right), \tag{3.57}$$

$$\text{subject to:} \quad \forall_{i=1}^{\ell}((y_i(\vec{w}\cdot\vec{x}_i + b) \geq 1 - \xi_i) \wedge (\xi_i > 0)),$$

where $\vec{w}$ is an orthogonal vector to the margin hyperplane, b is a real constant that characterizes the plane's offset from the origin, ξ_i is a slack variable associated with the point $(\vec{x}_i, y_i)$, and C is a positive real constant.

The main idea behind TSVM is to solve the problem 3.57 on the labeled dataset, and use this solution to pseudo-label the set $\mathcal{U}$. With a new augmented training set composed of the original labeled set and the new pseudo-labeled set, we find the new maximum margin by solving a slight variation of 3.57 given by:

$$\min_{\vec{w},b,\xi_1,\dots,\xi_{\ell+u}} \left(\frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{\ell} \xi_i + C'\sum_{j=\ell+1}^{\ell+u} \xi_i \right), \tag{3.58}$$

$$\text{subject to:} \quad \forall_{i=1}^{\ell+u}((y_i(\vec{w}\cdot\vec{x}_i + b) \geq 1 - \xi_i) \wedge (\xi_i > 0)),$$

where we include the slack variables for the pseudo-labeled points with the constant C'. This allows the algorithm to control the importance of the label and unlabeled examples.

In order to control accuracy metrics, such as recall and precision, Joachims [19] proposes the number of desired positive labels to be passed as a hyperparameter, allowing new constants C'_- and C'_+ to be defined in the optimization problem to accommodate the ratio of positive and negative examples. The final optimization problem for the TSVM is given by:

$$\min_{\vec{w},b,\xi_1,\dots,\xi_{\ell+u}} \left(\frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{\ell} \xi_i + C'_+\sum_{y_i=1} \xi_i + C'_-\sum_{y_i=-1} \xi_i \right), \tag{3.59}$$

$$\text{subject to:} \quad \forall_{i=1}^{\ell+u}((y_i(\vec{w}\cdot\vec{x}_i + b) \geq 1 - \xi_i) \wedge (\xi_i > 0)).$$

The problem (3.59) is encapsulated by the function **SolveSVM**$(\mathcal{T}, C, C'_-, C'_+)$, yielding the solution $(\vec{w}, b, \vec{\xi})$, where $\mathcal{T}$ is the training set, and $\vec{\xi} \in \mathbb{R}^{\ell+u}$ is the vector where ξ_i

is the slack variable of example $\vec{x}_i$.

Algorithm: 11 TSVM (Binary)

Input:

$C \in \mathbb{R}$	:: Weight of the Slack Variables for Labeled Examples
$C' \in \mathbb{R}$	:: Weight of the Slack Variables for Unlabeled Examples
$p \in \mathbb{N}$	:: Number of Unlabeled Points to Receive a Positive Label

Output:

$\overline{\mathcal{L}} \subset \mathcal{X} \times \mathcal{Y}$	:: Pseudo-labeled Dataset

```
 1. let $\vec{w} \in \mathbb{R}^{\ell}$, $b \in \mathbb{R}$, $\vec{\xi} \in \mathbb{R}^{\ell+u}$
 2. $\vec{w}, b, \vec{\xi} \leftarrow$ SolveSVM$(\mathcal{L}, C, 0, 0)$
 3. let $V \in \mathbb{R}^{u\times 2} \leftarrow V_i = (\vec{x}_i, (\vec{w} \cdot \vec{x}_i + b))$, $\vec{x}_i \in \mathcal{U}$
 4. $V \leftarrow$ Order$(V, V_{i2})$
 5. let $\overline{\mathcal{L}} \subset \mathcal{X} \times \mathcal{Y} \leftarrow \{(\vec{x}_i, y_i) : \vec{x}_i \in V_1, (y_i = 1 \wedge 1 \leq i \leq p) \vee (y_i = -1 \wedge i > p)\}$
 6. let $C'_+ \in \mathbb{R} \leftarrow \frac{p}{u-p} \cdot 10^{-5}, C'_- \in \mathbb{R} \leftarrow 10^{-5}$
 7. let $I_u \in \mathbb{N} \leftarrow \{\ell+1, \ldots, \ell+u\}$
 8. while $(C'_- < C') \vee (C'_+ < C')$ do:
 9.     $\vec{w}, b, \vec{\xi} \leftarrow$ SolveSVM$(\mathcal{L} \cup \overline{\mathcal{L}}, C, C'_-, C'_+)$
10.     while $\exists i, j \in I_u((y_i y_j < 0) \wedge (\xi_i > 0) \wedge (\xi_j > 0) \wedge (\xi_i + \xi_j > 2))$ do:
11.         $\overline{\mathcal{L}} \leftarrow (\overline{\mathcal{L}} \cup \{(x_i, -y_i), (x_j, -y_j)\}) \setminus \{(x_i, y_i), (x_j, y_j)\}$
12.         $\vec{w}, b, \vec{\xi} \leftarrow$ SolveSVM$(\mathcal{L} \cup \overline{\mathcal{L}}, C, C'_-, C'_+)$
13.     $C'_- \leftarrow \min\{2C'_-, C'\}$, $C'_+ \leftarrow \min\{2C'_+, C'\}$
14. return $\overline{\mathcal{L}}$
```

The algorithm begins by solving a inductive SVM over the labeled dataset $\mathcal{L}$. Next, we create and order the pairs $(\vec{x}_i, (\vec{w} \cdot \vec{x}_i + b))$ by decreasing order of their second component, and construct the pseudo-labeled set $\overline{\mathcal{L}}$ by setting the label $y_i = 1$ for the first p points in V and $y = -1$ for the last $u - p$.

The constants C'_+ and C'_- are initiated to small values. Inspired by Joachims [19], we use 10^{-5}, with the value of C'_+ multiplied by the ratio of positive to negative examples. These values will be progressively incremented with each iteration until they reach the value C'.

The main loop starts by solving the altered SVM problem (3.59) on the training set

$\mathcal{L} \cup \overline{\mathcal{L}}$. We then search for pairs of points in the pseudo-labeled dataset with opposite labels ($y_i y_j < 0$), where both have a positive influence in the cost $\xi_i > 0 \wedge \xi_j > 0$, and a conjoined influence above a certain threshold - here we use $\xi_i + \xi_j > 2$ [19]. For every such pair we swap the labels and retrain the model to find a new margin. When exhausted, we increase the cost of mislabeling by increasing C'_+ and C'_- and repeat the process.

Convergence

To show the algorithm converges we just need to prove that the inner and outer while loops stops at a finite number of iterations.

Proposition 1.8.1: The inner loop of the algorithm stops at a finite number of loops.

Proof: (see Appendix A.8)

Proposition 1.8.2: The outer loop of the algorithm stops at a finite number of loops.

Proof: Since by *Proposition 1.8.1* the inner loop converges, and the condition of the outer loop only depends on the values of C'_- and C'_+ which are doubling every iteration, the loop breaks when the last of the two constants reach a value higher than C'. Mathematically, let $n \in \mathbb{N}$ be the smallest number of iterations that satisfy this condition, we have:

$$2^n \cdot \min\{C'_-, C'_+\} \geq C' \tag{3.60a}$$

$$n \geq \log_2\left(\frac{C'}{\min\{C'_-, C'_+\}}\right) \tag{3.60b}$$

$$n = *\log_2\left(\frac{C'}{\min\{C'_-, C'_+\}}\right). \tag{3.60c}$$

$\blacksquare$

Hyperparameters

The algorithm receives three hyperparameters as inputs. The constants C and C' are positive real numbers designed to attribute different levels of importance to labeled and unlabeled misclassification, while p is the number of unlabeled examples to receive a positive pseudo-label.

Variations

The most common variation of the algorithm is the adaptation to remove the hyperparameter p and allow the method to freely classify the unlabeled examples.

Algorithm: 12 TSVM 2 (Binary)

Input:

$C \in \mathbb{R}$	:: Weight of the Slack Variables for Labeled Examples
$C' \in \mathbb{R}$	:: Weight of the Slack Variables for Unlabeled Examples

Output:

$\overline{\mathcal{L}} \subset \mathcal{X} \times \mathcal{Y}$	:: Pseudo-labeled Dataset

```
1. let $\vec{w} \in \mathbb{R}^{\ell}$, $b \in \mathbb{R}$, $\vec{\xi} \in \mathbb{R}^{\ell+u}$
2. $\vec{w}, b, \vec{\xi} \leftarrow \mathbf{SVM}(\mathcal{L}, C, 0)$
3. let $\overline{\mathcal{L}} \subset \mathcal{X} \times \mathcal{Y} \leftarrow \{(\vec{x}_i, y_i) : \vec{x}_i \in \mathcal{U}, y_i = \mathbf{sign}(\vec{w} \cdot \vec{x}_i + b)\}$
4. let $I_u \in \mathbb{N} \leftarrow \{\ell+1, \ldots, \ell+u\}$
5. while $(C' < C)$ do:
6.     $\vec{w}, b, \vec{\xi} \leftarrow \mathbf{SVM}(\mathcal{L} \cup \overline{\mathcal{L}}, C, C')$
7.     while $\exists i, j \in I_u((y_i y_j < 0) \wedge (\xi_i > 0) \wedge (\xi_j > 0) \wedge (\xi_i + \xi_j > 2))$ do:
8.             $\overline{\mathcal{L}} \leftarrow (\overline{\mathcal{L}} \cup \{(x_i, -y_i), (x_j, -y_j)\}) \setminus \{(x_i, y_i), (x_j, y_j)\}$
9.             $\vec{w}, b, \vec{\xi} \leftarrow \mathbf{SVM}(\mathcal{L} \cup \overline{\mathcal{L}}, C, C')$
10.    $C' \leftarrow \min\{2C', C\}$
11. return $\overline{\mathcal{L}}$
```

Where the function **SVM** solves the minimization problem (3.58).

3.9 LapSVM

The Laplacian Support Vector Machine (LapSVM) was first proposed by Belkin et al. in 2006 [4] as an adaptation of the well known SVM algorithm to handle unlabeled data.

Take the standard SVM minimization problem using the hinge loss function:

$$f^* = \operatorname*{argmin}_{f\in\mathcal{H}_K} \frac{1}{\ell}\sum_{i=1}^{\ell}(1-y_i f(\vec{x}_i))_+ + \gamma_A ||f||_K^2, \tag{3.61}$$

where $(1-y_if(\vec{x}_i))_+ = \max\{0, 1-y_if(\vec{x}_i)\}$ is the hinge loss function, $K : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ is a Mercer kernel and $\mathcal{H}_K$ is the associated Reproducing Kernel Hilbert Space (RKHS) of functions $\mathcal{X} \to \mathbb{R}$ with a given norm $||\cdot||_K$. The expression $\gamma_A||f||_K^2$ is then a regularizing term to impose smoothness [4].

Furthermore, by the classical Representer Theorem, the above equation can be represented by a linear combination of the Kernel function evaluated on the training examples.

$$f^*(\vec{x}) = \sum_{i=1}^{\ell}\alpha_i K(\vec{x}_i, x). \tag{3.62}$$

This turns the problem into a search for the coefficients α_i. In their approach, Belkin et al. [4] shows that the standard SVM minimization problem:

$$\min_{f\in\mathcal{H}_K,\xi_i\in\mathcal{R}} \frac{1}{\ell}\sum_{i=1}^{\ell}\xi_i + \gamma_A||f||_K^2, \tag{3.63a}$$

$$\text{s.t. } y_if(\vec{x}_i) \geq 1-\xi_i, \quad i = 1,\ldots,\ell \tag{3.63b}$$

$$\xi_i \geq 0, \quad i = 1,\ldots,\ell, \tag{3.63c}$$

can be rewritten through the Lagrange multiplies $\beta = [\beta_1,\ldots,\beta_\ell]^T$ as

$$\alpha = \frac{Y\beta^*}{2\gamma_A}, \tag{3.64}$$

where

$$Y \in \mathcal{Y}^{\ell\times\ell} = \begin{cases} Y_{ij} = y_i & \text{if } i = j \\ Y_{ij} = 0 & \text{if } i \neq j \end{cases}, \tag{3.65}$$

$$Q = Y\left(\frac{Y}{2\gamma_A}\right)Y, \tag{3.66}$$

$$\beta^* = \max_{\beta\in\mathbb{R}^\ell}\sum_{i=1}^{\ell}\beta_i - \frac{1}{2}\beta^TQ\beta. \tag{3.67}$$

Now we turn to the adaptation of the given problem to the semi-supervised setting. The way the information of unlabeled examples are absorbed into the training process is through another regularizing term $\gamma_I ||f||_I^2$ that encodes the intrinsic structure of the distribution of the inputs in $\mathcal{X}$. A natural choice for the term $||f||_I^2$ is $\int_{x\in\mathcal{M}} ||\nabla_{\mathcal{M}} f||^2 dP_{\mathcal{X}}$ [4], where $\mathcal{M} \subset \mathbb{R}^n$ is a compact sub-manifold, $\nabla_{\mathcal{M}} f$ is the gradient of f in $\mathcal{M}$, and $P_{\mathcal{X}}$ is the probability distribution of the inputs x. The regularizing term is approximated as:

$$\gamma_I \int_{x\in\mathcal{M}} ||\nabla_{\mathcal{M}} f||^2 dP_{\mathcal{X}} \approx \frac{\gamma_I}{(u+\ell)^2} \sum_i^{\ell+u} \sum_i^{\ell+u} (f(\vec{x}_i) - f(\vec{x}_j))^2 W_{ij}, \tag{3.68}$$

where W is the similarity matrix of the data. We can also write it in matrix notation

$$\frac{\gamma_I}{(u+\ell)^2} \sum_i^{\ell+u} \sum_i^{\ell+u} (f(\vec{x}_i) - f(\vec{x}_j))^2 W_{ij} = \frac{\gamma_I}{(u+\ell)^2} \mathbf{f}^T L \mathbf{f}, \tag{3.69}$$

where $\mathbf{f} = [f(\vec{x}_1), \ldots, f(\vec{x}_{\ell+u})]^T$ and $L = D - W$ is the graph Laplacian with D being the diagonal matrix given by $D_{ii} = \sum_{j=1}^{\ell+u} W_{ij}$. The minimization problem for the graph Laplacian is then:

$$f^* = \underset{f\in\mathcal{H}_K}{\operatorname{argmin}} \frac{1}{\ell} \sum_{i=1}^{\ell} (1 - y_i f(\vec{x}_i))_+ + \gamma_A ||f||_K^2 + \gamma_I ||f||_I^2, \tag{3.70a}$$

$$f^* = \underset{f\in\mathcal{H}_K}{\operatorname{argmin}} \frac{1}{\ell} \sum_{i=1}^{\ell} (1 - y_i f(\vec{x}_i))_+ + \gamma_A ||f||_K^2 + \frac{\gamma_I}{(u+\ell)^2} \mathbf{f}^T L \mathbf{f}. \tag{3.70b}$$

Similar to the standard SVM problem, Belkin et al. [4] shows that the minimization problem can be solved through Lagrange multipliers as:

$$\beta^* = \max_{\beta\in\mathbb{R}^\ell} \sum_{i=1}^{\ell} \beta_i - \frac{1}{2} \beta^T Q \beta, \tag{3.71}$$

$$s.t. \ \sum_{i=1}^{\ell} \beta_i y_i = 0 \text{ and } 0 \le \beta_i \le \frac{1}{\ell}, \ \ i = 1, \ldots, \ell, \tag{3.72}$$

where

$$Q = Y J \overline{K} (2\gamma_A I + 2 \frac{\gamma_I}{(\ell+u)^2} L \overline{K})^{-1} J^T Y, \tag{3.73}$$

$$Y = \begin{cases} Y_{ij} = y_i & \text{if } i = j \\ Y_{ij} = 0 & \text{if } i \neq j \end{cases} \tag{3.74}$$

$$J \in \mathbb{R}^{\ell \times (\ell+u)} = \begin{cases} J_{ij} = 1, & \text{if } i = j \text{ and } j \leq \ell \\ J_{ij} = 0, & \text{otherwise} \end{cases}, \tag{3.75}$$

$$\overline{K} \in \mathbb{R}^{(\ell+u)\times(\ell+u)} = \Big\{ \overline{K}_{ij} = K(\vec{x}_i, \vec{x}_j). \tag{3.76}$$

The solution for β^* is then used to calculate the coefficients α_i through:

$$\alpha = (2\gamma_A I + 2\frac{\gamma_I}{\ell+u}^2 LK)^{-1} J^T Y \beta^*, \tag{3.77}$$

with the final solution being given by:

$$f^*(\vec{x}) = \sum_{i=1}^{\ell+u} \alpha_i K(\vec{x}, \vec{x}_i). \tag{3.78}$$

Algorithm: 13 LapSVM

```
Input:
$W \in \mathbb{R}^{(\ell+u)\times(\ell+u)}$   :: Similarity Matrix
$K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$   :: Kernel Function
$\gamma_A \in \mathbb{R}$   :: Regularization constant for the ambient space
$\gamma_I \in \mathbb{R}$   :: Regularization constant for the input space

Output:
$\mathcal{C} : \mathcal{X} \to \mathcal{Y}$   :: Classifier

1. let $D \in \mathbb{R}^{(\ell+u)\times(\ell+u)} \leftarrow D_{ij} = \begin{cases} \sum_{i=1}^{\ell+u} W_{ij}, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$
2. let $\overline{K} \in \mathbb{R}^{(\ell+u)\times(\ell+u)} \leftarrow \overline{K}_{ij} = K(\vec{x}_i, \vec{x}_j)$
3. let $L \in \mathbb{R}^{(\ell+u)\times(\ell+u)} \leftarrow D - W$
4. let $Y \in \mathbb{R}^{\ell\times\ell} \leftarrow Y_{ij} = \begin{cases} y_i, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$
5. let $J \in \mathbb{R}^{\ell\times(\ell+u)} \leftarrow J_{ij} = \begin{cases} 1, & \text{if } i = j \text{ and } j \leq \ell \\ 0, & \text{otherwise} \end{cases}$
6. let $I \in \mathbb{R}^{(\ell+u)\times(\ell+u)} \leftarrow I_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$
7. let $Q \in \mathbb{R}^{\ell\times\ell} \leftarrow YJ\overline{K}\left(2\gamma_A I + 2\frac{\gamma_I}{(\ell+u)^2}L\overline{K}\right)^{-1} J^T Y$
8. let $\beta^* \in \mathbb{R}^{\ell\times 1} \leftarrow \textbf{Solve}\left(\max \sum_{i=1}^{\ell} \beta_i - \frac{1}{2}\beta^T Q \beta, \sum_{i=1}^{\ell} \beta_i y_i = 0, 0 \leq \beta_i \leq \frac{1}{\ell}\right)$
9. let $\alpha \in \mathbb{R}^{\ell+u} \leftarrow \left(2\gamma_A I + 2\frac{\gamma_I}{(\ell+u)^2}LK\right)^{-1} J^T Y \beta^*$
10. let $f^*(\vec{x}) : \mathcal{X} \to \mathbb{R} \leftarrow \sum_{i=1}^{\ell+u} \alpha_i K(\vec{x}_i, \vec{x})$
11. let $\mathcal{C}(\vec{x}) : \mathcal{X} \to \mathcal{Y} \leftarrow \text{sign}(f^*(\vec{x}))$
12. return $\mathcal{C}$
```

The algorithm is straightforward, with a sequence of definitions and calculations of the described matrices, and then the optimization problem of finding β^* so the coefficients α_i can be finally calculated and the classifier $\mathcal{C}$ defined.

Convergence

The convergence of the LapSVM depends on the optimization problem behind β^*, since the remaining of the algorithm is simply calculating different matrices.

Hyperparameters

The LapSVM, like graph-based models, uses a similarity matrix W to model the similarity between points. This is used to approximate the regularizing term $||f||_I^2$ as shown. The algorithm also requires a kernel function K, and the constants γ_A and γ_I that controls the penalization of f in the ambient space and the input space $\mathcal{X}$ respectively.

Variations

Since the algorithm requires the calculation of the inverse of the $(\ell+u)\times(\ell+u)$ matrix $\overline{K}$, which has time complexity $\mathcal{O}((\ell+u)^3)$, the training process can be slow for large datasets. To address this, Belkin et al. [4] proposes the equation a variation of the LapSVM where $f^* = w^T x$ is used instead of Eq. (3.62).

3.10 SSGMM

The Semi-Supervised Gaussian Mixture Model (SSGMM) algorithm was initially proposed by Shahshahani and Landgrebe in 1994 [30]. The algorithm has in essence the same structure of a simple supervised expectation maximization (EM) algorithm for a mixture of Gaussians. The difference only arises in the formula for updating the values of the mixture coefficients π, the mean vectors $\vec{\mu}$, and the Covariance matrices Σ. Since the SSGMM is a generative algorithm, our goal is to model the input space $\mathcal{X}$ itself, that is, we want to determine the likelihood that a point $\vec{x}$ is observed, or $P(\vec{x})$. Let the distribution of the input space be a mixture of $|\mathcal{Y}|$ multivariate Gaussians (one for each class), then our desired probability is given by:

$$P(\vec{x}|\Theta) = \sum_{c \in \mathcal{Y}} \pi_c \mathcal{N}(\vec{x}|\vec{\mu}_c, \Sigma_c), \tag{3.79}$$

where $\Theta = \{\pi_1, \vec{\mu}_1, \Sigma_1, \ldots, \pi_{|\mathcal{Y}|}, \vec{\mu}_{|\mathcal{Y}|}, \Sigma_{|\mathcal{Y}|}\}$ are all the Gaussian parameters, with $\pi_i \in \mathbb{R}$, $\vec{\mu}_i \in \mathbb{R}^n$, and $\Sigma_i \in \mathbb{R}^{n \times n}$, where n is the dimension of the point $\vec{x}_i$. Lastly, the term $\mathcal{N}(\vec{x}|\vec{\mu}, \Sigma_c)$ is the multivariate Gaussian distribution with mean vector $\vec{\mu}_c$ and covariance matrix Σ_c. The learning procedure is then stated as follows: Let there be the set of labeled examples $\mathcal{L} = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_\ell, y_\ell)\}$ of size ℓ and the set $\mathcal{U} = \{\vec{x}_{\ell+1}, \ldots, \vec{x}_{\ell+u}\}$ with u unlabeled examples, and let $S_c = \{\vec{x} : (\vec{x}, c) \in \mathcal{L}\}$ be the set of all inputs from the labeled dataset with the label c. We perform an iterative process of updating the values of $\pi_c, \vec{\mu}_c, \Sigma_c$ for every class c, until convergence, and using the following equations [30]:

$$r_{i,c} = \frac{\pi_c \mathcal{N}(\vec{x}_{\ell+i}|\mu_c, \Sigma_c)}{\sum\limits_{j \in \mathcal{Y}} \pi_j \mathcal{N}(\vec{x}_{\ell+i}|\mu_j, \Sigma_j)}, \tag{3.80}$$

$$\pi_c = \frac{1}{u} \sum_{i=1}^{u} r_{i,c}, \tag{3.81}$$

$$\vec{\mu}_c = \frac{1}{u\pi'_c + |S_c|} \left(\sum_{i=1}^{u} r_{i,c} \vec{x}_{\ell+i} + \sum_{\vec{x} \in S_c} \vec{x} \right), \tag{3.82}$$

$$\Sigma_c = \frac{\sum\limits_{i=1}^{u} r_{i,c}(\vec{x}_{\ell+i} - \vec{\mu}_c)(\vec{x}_{\ell+i} - \vec{\mu}_c)^T + \sum\limits_{\vec{x} \in S_c} (\vec{x} - \vec{\mu}_c)(\vec{x} - \vec{\mu}_c)^T}{u\pi'_c + |S_c|}, \tag{3.83}$$

where $r_{i,c}$ is our current estimate of the probability of a point $\vec{x}_i$ having a class c, and we represent the previous value of $\pi, \vec{\mu}$ or Σ by $\pi', \vec{\mu}'$ and Σ', respectively.

Algorithm: 14 SSGMM

Input:

$\pi_1, \dots, \pi_{\lvert\mathcal{Y}\rvert} \in \mathbb{R}$	:: Initial estimation of the mixture coefficients for each class
$\vec{\mu}_1, \dots, \vec{\mu}_{\lvert\mathcal{Y}\rvert} \in \mathbb{R}^n$	:: Initial estimation of the average vectors
$\Sigma_1, \dots, \Sigma_{\lvert\mathcal{Y}\rvert} \in \mathbb{R}^{n\times n}$	:: Initial estimation of the Covariance matrices

Output:

$\mathcal{C} : \mathcal{X} \to \mathcal{Y}$	:: Classifier

```
 1. for $c \leftarrow 1$ to $|\mathcal{Y}|$ do:
 2.         $S_c \leftarrow \{\vec{x} \in \mathcal{X} : (\vec{x}, c) \in \mathcal{L}\}$
 3.         $\pi'_c \leftarrow -\pi_c,\ \vec{\mu}'_c \leftarrow -\vec{\mu}_c,\ \Sigma'_c \leftarrow -\Sigma_c$
 4. while $\forall_{j\in\mathcal{Y}} \big((\pi_j \neq \pi'_j) \vee (\vec{\mu}_j \neq \vec{\mu}'_j) \vee (\Sigma_j \neq \Sigma'_j)\big)$ do:
 5.         for $i \leftarrow 1$ to $u$ do:
 6.                 for $c \in \mathcal{Y}$ do:
 7.                         $r_{i,c} \leftarrow \dfrac{\pi_c \mathcal{N}(\vec{x}_{\ell+i}|\mu_c, \Sigma_c)}{\sum_{j\in\mathcal{Y}} \pi_j \mathcal{N}(\vec{x}_{\ell+i}|\mu_j, \Sigma_j)}$
 8.         for $c \in \mathcal{Y}$ do:
 9.                 $\pi'_c \leftarrow \pi_c,\ \vec{\mu}'_c \leftarrow \vec{\mu}_c,\ \Sigma'_c \leftarrow \Sigma_c$
10.                 $\pi_c \leftarrow \dfrac{1}{u}\sum_{i=1}^{u} r_{i,c}$
11.                 $\vec{\mu}_c \leftarrow \dfrac{1}{u\pi'_c + |S_c|}\left(\sum_{i=1}^{u} r_{i,c}\vec{x}_{\ell+i} + \sum_{\vec{x}\in S_c} \vec{x}\right)$
12.                 $\Sigma_c \leftarrow \dfrac{\sum_{i=1}^{u} r_{i,c}(\vec{x}_{\ell+i} - \vec{\mu}_c)(\vec{x}_{\ell+i} - \vec{\mu}_c)^T + \sum_{\vec{x}\in S_c} (\vec{x} - \vec{\mu}_c)(\vec{x} - \vec{\mu}_c)^T}{u\pi'_c + |S_c|}$
13. let $\mathcal{C}(\vec{x}) : \mathcal{X} \to \mathcal{Y} \leftarrow \underset{y}{\operatorname{argmax}}\ \mathcal{N}(\vec{x}|\vec{\mu}_y, \Sigma_y)$
14. return $\mathcal{C}$
```

We start the algorithm by defining the sets S_c for each class and starting the variables that will retain the previous values of $\pi, \vec{\mu}$ and Σ. Here we set them to be the negative of their respecting starting values to assure the main loop will start. We then set the convergence criteria as when no difference is noted among the previous and current value of the parameters. The main loop then starts by estimating the probabilities of each unlabeled point i having the class c, that is $r_{i,c}$. We then update the variables holding the previous values of the parameters and update the parameters themselves according to Eq. (3.81), (3.82) and (3.83).

Once the algorithm converges, we define the classifier as a function of a point $\vec{x}$ that selects the class that has the higher likelihood of generating the point $\vec{x}$.

Convergence

According with McLachlan et al. [23], the EM algorithm is guaranteed to increase the desired likelihood at every iteration. However, it need not converge to a global maximum, being possible that it converges to a local maximum or a saddle point.

Hyperparameters

The only hyperparameters the algorithm requires are the initial estimates of the mixture coefficients π_i, the average vectors $\vec{\mu}_i$ and the covariance matrices Σ_i for each class i. According to Shahshahani and Landgrebe [30], a reasonable set of starting values can be obtained by using the labeled examples only.

Variations

The variations of the SSGMM algorithm are obtained when different assumptions or type of distributions are used. In particular, in Shahshahani and Landgrebe [30] discusses alternatives like the non-parametric case (not Gaussians), or a semi-parametric approach where some assumptions of the parametric case still holds, and a different set of equations for updating the values $\pi, \vec{\mu}$ and Σ are derived.

Chapter 4

Methodology

4.1 OpenML-CC18

OpenML is a free online framework for the study of Machine Learning, containing datasets, algorithms and experiments designed to help further knowledge and study of the field [34].

Among the benchmark suites offered, the OpenML-CC18 [6] is a collection of 72 curated datasets for classification tasks, with the following admission criteria [28]:

- No artificial datasets;
- No dataset is a subset of a larger datasets;
- No dataset is a binarization of other datasets;
- No dataset is perfectly predictable by a single feature;
- $500 \leq$ Number of Instances ≤ 10000;
- $2 \leq$ Number of Classes;
- At least 20 examples per class;
- No class is less than 5% of the total number of examples;
- Number of features after auto-encoding < 5000.

The datasets are distributed by number of examples and by number of features (after auto-encoding) according to Figure 4.1.

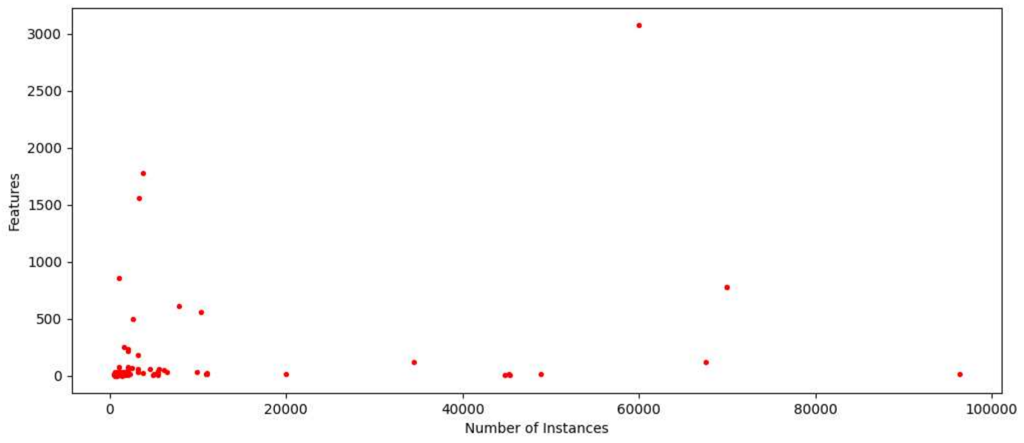

Figure 4.1. Distribution of datasets by instances and features.
Source: Adapted from [6].

4.2 LAMDA-SSL

LAMDA-SSL is a Python library for the study and applications of semi-supervised learning algorithms, encompassing modules for data processing, data augmentation, hyperparameter search, model training and evaluation, among others. The framework contains 30 semi-supervised algorithms implemented in Python, 12 of which are statistical learning methods and 18 are deep learning models, being able to perform classification, regression and clustering tasks in a wide variety of data types, such as text, image, graph and tabular data [18].

The algorithms of interest are the nine statistical learning methods for classification tasks, according to Figure 4.2.

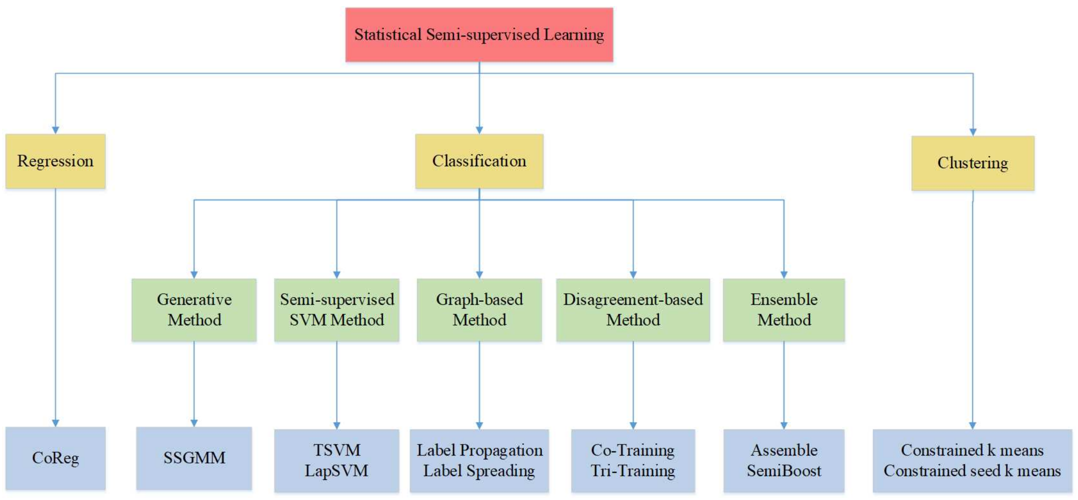

Figure 4.2. Statistical learning algorithms in LAMDA-SSL.
Source: [17].

4.3 General Setup

The first step of the research consists in collecting implementations of semi-supervised algorithms, as well as the datasets to be used. We suggest the use of the Python library LAMDA-SSL (4.2) with 9 algorithms based on statistical Machine Learning, as well as the Self-Training implementation by the Scikit-Learn [27] Python library. The diversity of algorithms in the proposed repository fills every category of interest presented in Figure (1.1), except the perturbation-based category, for which no satisfactory implementation was found. The categories presented in Figure (1.1) are filled in the following manner:

- Self-Training: Self-Training
- Co-Training: Co-Training, Tri-Training
- Boosting: Assemble, SemiBoost
- Maximum-margin: LapSVM
- Perturbation-based:
- Manifolds: LapSVM
- Generative Models: SSGMM
- Transductive: TSVM, Label Spreading, Label Propagation

with LapSVM being an intersectional model from the maximum-margin approach and manifold regularization, and disregarding unsupervised-preprocessing since it's not the focus of the research.

For the data we propose the OpenML-CC18 collection of 72 curated datasets (4.1), 37 of them being for binary classification and the remaining 35 for multi-class with a well distributed class proportion. We restrict the study to a subset of 44 of these datasets, selected by size in order to allow the experiments to run in a manageable time frame.

Two categories of experiments are carried, inductive and transductive.

4.4 Inductive Setup

The inductive setup is characterized by two consecutive partitions of the dataset. We start with an initial 75%-25% split for the training and test examples respectively. Next, we adapt the dataset to the semi-supervised setting with a further split of the training set between two parts. One remains untouched and will be regarded as the labeled dataset, while the other has their labels removed to simulate unlabeled points. The new labeled dataset is used to do a hyperparameter search, along with a list of possible values for each desired hyperparameter to be evaluated. Once all the previous steps are finished, the training can be conducted using the labeled and unlabeled data, along with the optimal hyperparameters found. This results in the desired model, and the training time is measured for future comparisons. Lastly, we evaluate the model in the test data to obtain the positive, global and negative accuracy. A diagram of this process is presented in Figure 4.3.

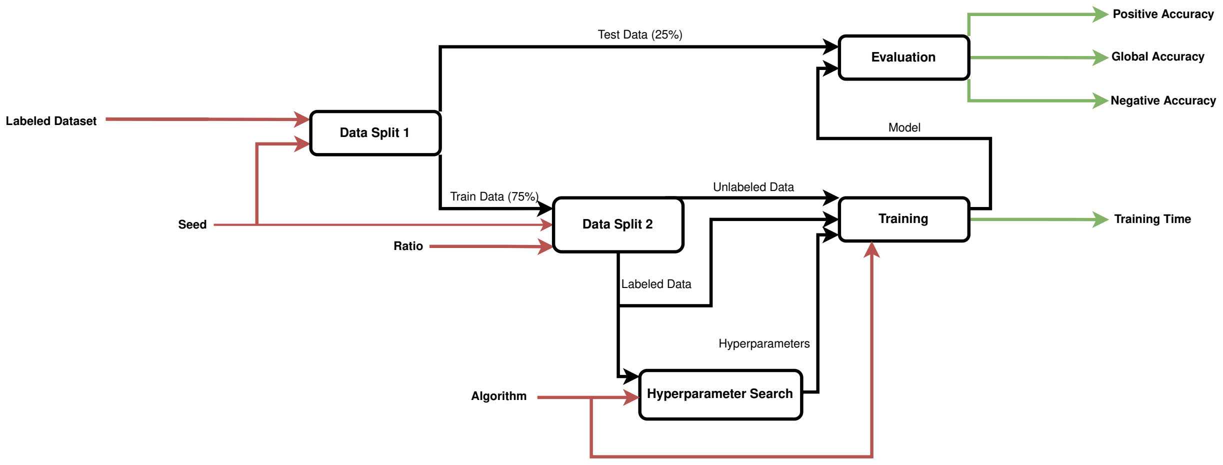

Figure 4.3. Inductive setup.

Two categories of experiments are performed in the described setting. The first consists in running only multi-class datasets in all algorithms that are supported by this framework. The second is a binary classification setup, where all algorithms are run in the originally binary datasets, as well as the multi-class sets after a forced binarization. This is achieved by selecting the most frequent class as the positive label, and all others are grouped as the negative label. This is done to secure a reasonable amount of positively labeled examples and prevent the models from degenerating into classifying everything into the negative category.

The binary experiments are performed for all 44 datasets, and 9 of the 10 algorithms, leaving the TSVM method out for its intrinsically transductive nature. We also note that even though the Label Spreading and Label Propagation algorithms are classified as transductive, they are frequently used in inductive settings, justifying the decision to include them in the inductive experiments as well.

For the multi-class, we can only carry the experiments in 17 out of the 44 datasets, since they represent the multi-class subset of our selection. As the for algorithms tested, we used 7 out of the 10 proposed, since the Assemble and SemiBoost methods are appropriate only for binary classification, and the TSVM doesn't qualify for the inductive setting as discussed before.

The final difference between the binary and multi-class settings, is in the meaning of the positive and negative examples. Here we defined the positive class for the binary experiments, commonly referred as the target, as the least populous class, while the negative is the most populous one. The positive accuracy is then defined as the proportion of correctly labeled positive examples, and the negative accuracy is equivalently defined as the proportion of correctly labeled negative ones. The positive and negative accuracy are commonly named as sensitivity and specificity, respectively. However, since a multi-class dataset with n classes would have n such accuracies, we define the positive accuracy of a multi-class model as accuracy of the least frequent class, and the negative accuracy as the accuracy of the most frequent class, analogously to the binary classification. Thus, disregarding the intermediary classes which are only considered in the global accuracy measurement.

4.5 Transductive Setup

The transductive setup differs from the inductive in the main goal of the learning task. Here we want the best possible classification for the unlabeled data, something that is potentially undesirable in an inductive task, since it can signal an overfitting of the training data. The setup begins with a dataset, a unlabeled ratio and a seed used for reproducibility purposes. The dataset is then split in two parts, one will remain unaffected and will serve as the labeled dataset, and the other will have its labels removed to simulate the unlabeled dataset – these labels will be saved for posterior calculations of accuracy. The labeled set is then used for a hyperparameter search, and the training is conducted with the optimal hyperparameters, the labeled dataset and the unlabeled dataset. Once the training is over, the time elapsed is registered, and the model is evaluated. Here we unlabeled dataset fulfills the role of the test set, with the posterior saved labels being the correct labels.

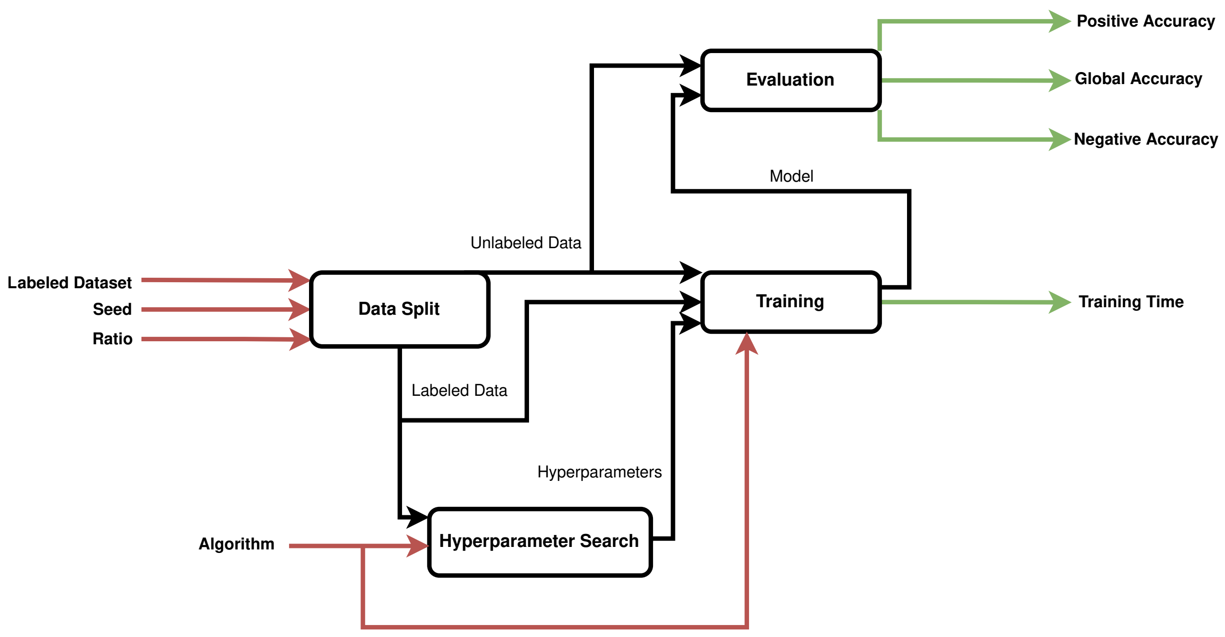

Figure 4.4. Transductive setup.

The experiments are carried in the same fashion as the binary setting for the inductive. This decision was made since it enable us to use all the 10 proposed algorithms – including the exclusively binary TSVM, Assemble and SemiBoost – as well as all the 44 datasets, permitting a more meaningful comparison between the methods. Furthermore, even though only the TSVM, Label Spreading and Label Propagation are naturally transductive, this setting is easily adapted to any semi-supervised method, since they share as a basic training premise the search for the best possible labels for the given unlabeled points. This setting is then carried for the originally binary datasets and the binarized multi-class datasets with the most frequent class as the positive and all the others grouped as the negative. The positive and negative accuracy retains their previously discussed meanings.

4.6 Result Analysis

Given the inductive and transductive setup described, a complete experiment consists of running a setup for every possible combination (respecting binary and multi-class limitations) of dataset, algorithm, ratio and seed, generating entries of the form

n	dataset	algorithm	seed	ratio	time	global accuracy	+ accuracy	- accuracy

for the n-th entry. These results are compiled and analyzed for the three proposed experiments, namely inductive binary, inductive multi-class and transductive.

Since semi-supervised tasks are sensitive to the ratio between labeled and unlabeled data, we propose four ratios of 0.25, 0.5, 0.75 and 0.9 of unlabeled data to evaluate different scenarios of data availability. Moreover, we use the same three seeds for all experiments, generating three respective measurements of time and accuracies for each combination of dataset, algorithm and ratio. These three measurements are then averaged and the result is used for the next step of statistical inference.

For the result analysis we perform a Wilcoxon signed-rank test for each separate pair of performance metric and ratio. A group of observations is composed of the chosen metric and ratio value for an algorithm, for each dataset, with each group being paired one-to-one by the dataset. This test is non-parametric and ranked, allowing for the assessment of global and pairwise differences between all groups, as well as the average rankings of each algorithm, making possible observations like a consistent superior or inferior performance of a method, all while not assuming a normal distribution of the data. The Holm p-correction technique [13] is applied to avoid false positives, since the comparison is between a sufficiently large number of pairs of algorithms, and such errors would be expected.

Chapter 5

Results

In this section we present the results of the inductive binary, inductive multi-class and the transductive setup from experiments run as describe in out methodology. We initially present a table of average ranks among each algorithm. This is done by ordering the algorithms for each dataset in ascending order, and assigning a rank according to its position - with ties being solved by averaging the upper and lower ranks, e.g., 1.5 when two algorithms tie for first. Following this information, we present a table containing the p-values[1] for each pairwise comparison among algorithms to access the significance of the result. Each entry in this table is colored green if the result is significant ($p \leq 0.05$) or red if it's not. The results of a given setup is further divided into the global, positive and negative accuracy. After presenting the data, we discuss the results.

5.1 Inductive

5.1.1 Binary

Global Accuracy

Ratio	ASB	LabelS	LapSVM	SemiB	LabelP	SelfT	CoT	TriT	SSGMM
0.25	2.68	3.09	4.91	4.49	3.80	2.54	4.09	3.43	4.79
0.50	2.52	3.30	5.23	4.08	3.55	3.02	4.16	3.89	5.11
0.75	2.52	3.52	5.46	2.38	3.45	2.95	4.61	3.78	4.70
0.90	2.61	3.61	5.89	2.62	3.84	3.56	4.34	3.89	5.09

Table 5.1: Average rank of algorithms per ratio (Inductive binary global accuracy).

[1]For a more concise chapter, some tables can be found in Appendix C.

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	0.286	0.024	0.006	0.003	1.000	0.575	1.000	0.000
LabelS	0.286	-	0.581	0.039	0.008	1.000	1.000	0.882	0.000
LabelP	0.024	0.581	-	0.225	0.117	1.000	0.581	0.156	0.000
LapSVM	0.006	0.039	0.225	-	1.000	0.286	0.071	0.003	1.000
SemiB	0.003	0.008	0.117	1.000	-	0.156	0.225	0.027	1.000
SelfT	1.000	1.000	1.000	0.286	0.156	-	0.293	0.148	0.028
CoT	0.575	1.000	0.581	0.071	0.225	0.293	-	0.766	0.457
TriT	1.000	0.882	0.156	0.003	0.027	0.148	0.766	-	0.125
SSGMM	0.000	0.000	0.000	1.000	1.000	0.028	0.457	0.125	-

Table 5.2: Pairwise comparison p-values (25% Inductive binary global accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	0.509	0.419	0.004	0.066	1.000	0.014	0.074	0.001
LabelS	0.509	-	1.000	0.024	0.770	1.000	0.048	1.000	0.000
LabelP	0.419	1.000	-	0.017	1.000	1.000	0.174	1.000	0.001
LapSVM	0.004	0.024	0.017	-	0.012	0.770	1.000	0.776	1.000
SemiB	0.066	0.770	1.000	0.012	-	1.000	1.000	1.000	0.129
SelfT	1.000	1.000	1.000	0.770	1.000	-	0.985	1.000	0.027
CoT	0.014	0.048	0.174	1.000	1.000	0.985	-	1.000	0.985
TriT	0.074	1.000	1.000	0.776	1.000	1.000	1.000	-	0.187
SSGMM	0.001	0.000	0.001	1.000	0.129	0.027	0.985	0.187	-

Table 5.3: Pairwise comparison p-values (50% Inductive binary global accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	0.056	0.156	0.001	1.000	1.000	0.464	1.000	0.001
LabelS	0.056	-	1.000	0.019	0.002	1.000	1.000	1.000	0.000
LabelP	0.156	1.000	-	0.003	0.036	1.000	1.000	1.000	0.001
LapSVM	0.001	0.019	0.003	-	0.000	0.110	0.019	0.011	1.000
SemiB	1.000	0.002	0.036	0.000	-	1.000	0.073	0.863	0.000
SelfT	1.000	1.000	1.000	0.110	1.000	-	0.911	1.000	0.031
CoT	0.464	1.000	1.000	0.019	0.073	0.911	-	0.250	0.332
TriT	1.000	1.000	1.000	0.011	0.863	1.000	0.250	-	0.038
SSGMM	0.001	0.000	0.001	1.000	0.000	0.031	0.332	0.038	-

Table 5.4: Pairwise comparison p-values (75% Inductive binary global accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	0.251	0.017	0.002	1.000	1.000	0.038	0.084	0.002
LabelS	0.251	-	1.000	0.004	0.004	1.000	0.216	1.000	0.001
LabelP	0.017	1.000	-	0.007	0.011	1.000	0.216	1.000	0.001
LapSVM	0.002	0.004	0.007	-	0.000	0.057	0.274	0.144	1.000
SemiB	1.000	0.004	0.011	0.000	-	1.000	0.005	0.049	0.000
SelfT	1.000	1.000	1.000	0.057	1.000	-	0.875	1.000	0.057
CoT	0.038	0.216	0.216	0.274	0.005	0.875	-	1.000	1.000
TriT	0.084	1.000	1.000	0.144	0.049	1.000	1.000	-	0.606
SSGMM	0.002	0.001	0.001	1.000	0.000	0.057	1.000	0.606	-

Table 5.5: Pairwise comparison p-values (90% Inductive binary global accuracy).

We begin our discussion with the evaluation of the global accuracy results for the binary runs. From Table 5.1 we can discern the best performing algorithms on average for each ratio of unlabeled data. The table suggests a superior performance of the Assemble and Self-training algorithms, with a clear inferior performance of the LapSVM and SSGMM. To access the pairwise differences, we suggest the comparison of the average rank of the algorithms in Table 5.1, followed by the verification of the p-value associated with that pair on the table with the appropriate unlabeled proportion. Take the Assemble algorithm at 50% unlabeled ratio as an example. This is the best performing algorithm at this ratio, with the following ranking:

$$\begin{array}{ccccccccc} \text{Assemble} > & \text{SelfT} > & \text{LabelS} > & \text{LabelP} > & \text{TriT} > & \text{SemiB} > & \text{CoT} > & \text{LapSVM} > & \text{SSGMM} \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ & 1.000 & 0.509 & 0.419 & 0.074 & 0.066 & 0.014 & 0.004 & 0.001 \end{array}$$

Here we color the algorithms with a statistically significant result (p-value ≤ 0.05) as green, and with red otherwise. The comparison of the Assemble and Self-Training yields a p-value of 1, while the Assemble and Label Spreading yields 0.509, with decreasing p-values until $p = 0.014$ is reached for the Co-Training as the first significant result.

The comparison of any two algorithms is done as described. First comparing their average ranks in Table 5.1 for the appropriate ratio, followed by checking the p-value in Tables 5.2,5.3,5.4 or 5.5. The possible inferences about performance are presented in Table 5.6:

25%	50%	75%	90%
ASB > LapSVM	ASB > LapSVM	ASB > LapSVM	ASB > LapSVM
ASB > SemiB	ASB > CoT	ASB > SSGMM	ASB > LabelP
ASB > LabelP	ASB > SSGMM	LabelS > LapSVM	ASB > CoT
ASB > SSGMM	LabelS > LapSVM	SemiB > LabelS	ASB > SSGMM
LabelS > LapSVM	LabelS > CoT	LabelS > SSGMM	LabelS > LapSVM
LabelS > SemiB	LabelS > SSGMM	SemiB > LapSVM	SemiB > LabelS
LabelS > SSGMM	SemiB > LapSVM	LabelP > LapSVM	LabelS > SSGMM
TriT > LapSVM	LabelP > LapSVM	CoT > LapSVM	SemiB > LapSVM
TriT > SemiB	LabelP > SSGMM	TriT > LapSVM	LabelP > LapSVM
LabelP > SSGMM	SelfT > SSGMM	SemiB > LabelP	SemiB > LabelP
SelfT > SSGMM		SemiB > SSGMM	SemiB > CoT
		LabelP > SSGMM	SemiB > TriT
		SelfT > SSGMM	SemiB > SSGMM
		TriT > SSGMM	LabelP > SSGMM

Table 5.6: Possible inferences (Inductive binary global accuracy).

A few comparisons are noteworthy, namely the five pairs with statistically significant p-values on all ratios, representing algorithms that unequivocally outperforms the other at any ratio. These results are:

- Assemble > LapSVM;
- Assemble > SSGMM;
- Label Spreading > LapSVM;
- Label Spreading > SSGMM;
- Label Propagation > SSGMM.

Finally, it is important to note that there is a balance between accuracy and training time to be considered. Even though the Assemble algorithm is among the best performers, it has the slowest training time of all, while algorithms like Self-Training being incredible fast in comparison, for a non-significant difference in accuracy. We present the average training time for each algorithm as a reference in Table 5.7.

Algorithm	Average Training Time (s)	Normalized Time
Label Spreading	3.48	1.00
Self-Training	7.41	2.13
Label Propagation	15.75	4.53
Tri-Training	25.17	7.23
SemiBoost	88.01	25.29
SSGMM	95.02	27.30
Co-Training	189.42	54.43
LapSVM	205.57	59.07
Assemble	319.73	91.88

Table 5.7: Average training time (Inductive binary).

Positive Accuracy

Ratio	ASB	LabelS	LapSVM	SemiB	LabelP	SelfT	CoT	TriT	SSGMM
0.25	3.32	2.86	4.60	4.84	3.48	3.07	4.87	4.26	2.49
0.50	3.00	3.18	5.20	4.54	3.45	3.59	4.27	3.61	2.68
0.75	2.97	3.25	5.31	3.49	3.30	3.24	5.26	4.26	2.39
0.90	3.03	3.59	5.49	3.35	3.48	3.66	4.25	3.91	2.64

Table 5.8: Average rank of algorithms per ratio (Inductive binary positive accuracy).

We turn to the evaluation of the performance of the algorithms for the positive accuracy.

The positive accuracy comparisons are done as discussed with the aid of Table 5.8 for the ranks, and the p-value tables (Appendix C). The only pair with a statistically significant dominance is the SSGMM algorithm which outperformed the Co-Training in all ratio scenarios. For a broad comparison, we see that the SSGMM is the clear winner in all ratios, with the Assemble second, and LabelSpreading, Self-Training and LabelPropagation with a similar performance. The Laplacian Support Vector Machine follows the trend in the global accuracy measures and is the worst performing algorithm.

We show the comparison of the algorithms at 50% unlabeled ratio and their respective p-values when compared to the best performing method.

SSGMM >	Assemble >	LabelS >	LabelP >	SelfT >	TriT >	CoT >	SemiB >	LapSVM
	↓	↓	↓	↓	↓	↓	↓	↓
	1.000	0.367	0.738	0.031	0.249	0.023	0.012	0.008

Here we observe that the transitive aspect of the performance does not carry to the p-values, with the values showing a non-monotonous decrease. Although there is a tendency to decrease the p-values as we compare models further apart - after all, the more discrepant their performance, the more likely it is that the Wilcoxon test will signal that they come from different distributions - the true value can fluctuate as some models can fail to be trained at certain ratios or datasets, making the number of paired observations vary slightly between pairs.

25%	50%	75%	90%
ASB > SemiB	ASB > LapSVM	ASB > LapSVM	ASB > LapSVM
LabelS > SemiB	ASB > SemiB	LabelS > LapSVM	LabelS > LapSVM
LabelS > CoT	ASB > CoT	SemiB > LapSVM	SemiB > LapSVM
LabelP > SemiB	LabelS > LapSVM	LabelP > LapSVM	LabelP > LapSVM
SSGMM > SemiB	LabelS > SemiB	CoT > LapSVM	CoT > LapSVM
SSGMM > CoT	LabelS > CoT	TriT > LapSVM	TriT > LapSVM
SSGMM > TriT	LabelP > LapSVM	SSGMM > LapSVM	SSGMM > LapSVM
	SSGMM > LapSVM	SSGMM > SelfT	SSGMM > SelfT
	LabelP > SemiB	SSGMM > CoT	SSGMM > CoT
	SSGMM > SemiB	SSGMM > TriT	SSGMM > TriT
	SSGMM > SelfT		
	SSGMM > CoT		

Table 5.9: Possible inferences (Inductive binary positive accuracy).

Negative Accuracy

Ratio	ASB	LabelS	LapSVM	SemiB	LabelP	SelfT	CoT	TriT	SSGMM
0.25	2.81	3.20	2.66	2.08	3.00	2.68	2.87	3.17	4.37
0.50	3.39	3.68	2.74	2.70	3.57	2.54	3.57	3.43	4.98
0.75	3.09	3.36	2.23	2.49	3.18	2.46	3.04	3.61	4.36
0.90	3.33	3.64	2.54	3.24	3.64	2.59	3.45	3.57	4.86

Table 5.10: Average rank of algorithms per ratio (Inductive binary negative accuracy).

Finally, we evaluate the negative accuracy using Table 5.10 for the ranks and Tables C.5, C.6, C.7 and C.8 for the p-values. It is clear that of all the metrics, the negative accuracy yields the least amount of statistically significant results. We have the dominance of the Self-Training algorithm over the SSGMM, with sparse significant results among other pairs, all involving the SSGMM method. The LapSVM, Self-Training and SemiBoost are the best performers by a close margin, with the SSGMM representing the worst method, what justifies it being the source of the significant results in this comparison.

25%	50%	75%	90%
ASB > CoT	LabelS > SSGMM	SemiB > SSGMM	SemiB > SSGMM
LabelS > SSGMM	SelfT > SSGMM	SelfT > SSGMM	SelfT > SSGMM
LabelP > SSGMM		CoT > SSGMM	
SelfT > SSGMM		TriT > SSGMM	
CoT > SSGMM			
TriT > SSGMM			

Table 5.11: Possible inferences (Inductive binary negative accuracy).

5.1.2 Multi-class

The multi-class experiments were conducted as described and the results are available in Appendix C. An evaluation of these results shows that no statistically significant result was achieved for any of the global, positive or negative accuracies. This is mainly attributed to the smaller amount of datasets, 17 in comparison to the 44 binary comparison. Furthermore, the combination of algorithm, dataset and ratio can fail in the training process, limiting the amount of groups even further for the comparison in the Wilcoxon signed rank test. The average training time of each algorithm is presented in Table 5.12.

Algorithm	Average Training Time (s)	Normalized Time
Label Spreading	9.33	1.00
Self-Training	26.37	2.83
Label Propagation	47.15	5.05
Tri-Training	63.13	6.77
SSGMM	264.70	28.37
Co-Training	1105.44	118.48
Assemble	20042.05	2148.13

Table 5.12: Average training time (Inductive multi-class).

Observing the above table we notice that, removing the LapSVM and SemiBoost algorithms from the comparison - once they are exclusively binary - we see that the rankings of each algorithm's training time remain exactly the same from the binary experiment presented in Table 5.7, noticeably with the Label Spreading leading as the fastest method, and the Assemble as the slowest.

5.2 Transductive

Global Accuracy

Ratio	ASB	LabelP	LabelS	LapSVM	SelfT	SemiB	SSGMM	TSVM
0.25	2.42	3.39	3.20	4.34	1.89	4.64	5.11	2.69
0.50	2.37	3.20	2.95	4.90	1.98	4.19	4.83	2.45
0.75	2.20	3.05	2.80	4.31	1.98	2.53	4.36	1.93
0.90	1.89	2.45	2.32	3.59	1.70	2.07	3.81	1.96

Table 5.13: Average rank of algorithms per ratio (Transductive global accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	0.011	0.011	0.011	0.000	0.525	0.000	1.000
LabelS	0.011	-	0.898	0.252	0.003	0.003	0.000	0.162
LabelP	0.011	0.898	-	0.525	0.008	0.000	0.001	0.179
LapSVM	0.011	0.252	0.525	-	1.000	0.003	0.898	0.043
SemiB	0.000	0.003	0.008	1.000	-	0.000	1.000	0.009
SelfT	0.525	0.003	0.000	0.003	0.000	-	0.000	1.000
SSGMM	0.000	0.000	0.001	0.898	1.000	0.000	-	0.001
TSVM	1.000	0.162	0.179	0.043	0.009	1.000	0.001	-

Table 5.14: Pairwise comparison p-values (25% Transductive binary global accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	0.032	0.007	0.003	0.001	0.684	0.000	1.000
LabelS	0.032	-	0.128	0.032	0.009	0.006	0.000	0.277
LabelP	0.007	0.128	-	0.032	0.009	0.001	0.001	0.128
LapSVM	0.003	0.032	0.032	-	0.168	0.001	1.000	0.018
SemiB	0.001	0.009	0.009	0.168	-	0.000	0.306	0.032
SelfT	0.684	0.006	0.001	0.001	0.000	-	0.000	1.000
SSGMM	0.000	0.000	0.001	1.000	0.306	0.000	-	0.000
TSVM	1.000	0.277	0.128	0.018	0.032	1.000	0.000	-

Table 5.15: Pairwise comparison p-values (50% Transductive binary global accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	0.051	0.051	0.006	0.817	1.000	0.000	0.817
LabelS	0.051	-	0.673	0.039	0.181	0.006	0.000	0.054
LabelP	0.051	0.673	-	0.022	0.128	0.004	0.001	0.112
LapSVM	0.006	0.039	0.022	-	0.002	0.002	1.000	0.006
SemiB	0.817	0.181	0.128	0.002	-	0.268	0.000	0.771
SelfT	1.000	0.006	0.004	0.002	0.268	-	0.000	1.000
SSGMM	0.000	0.000	0.001	1.000	0.000	0.000	-	0.000
TSVM	0.817	0.054	0.112	0.006	0.771	1.000	0.000	-

Table 5.16: Pairwise comparison p-values (75% Transductive binary global accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	0.259	0.099	0.006	1.000	1.000	0.000	1.000
LabelS	0.259	-	1.000	0.011	0.308	0.127	0.000	1.000
LabelP	0.099	1.000	-	0.025	0.308	0.131	0.001	0.768
LapSVM	0.006	0.011	0.025	-	0.002	0.002	1.000	0.015
SemiB	1.000	0.308	0.308	0.002	-	0.278	0.000	1.000
SelfT	1.000	0.127	0.131	0.002	0.278	-	0.000	1.000
SSGMM	0.000	0.000	0.001	1.000	0.000	0.000	-	0.002
TSVM	1.000	1.000	0.768	0.015	1.000	1.000	0.002	-

Table 5.17: Pairwise comparison p-values (90% Transductive binary global accuracy).

We begin our analysis of the transductive experiments with the global accuracy. We notice that the best performers are the Assemble, Self-Training and TSVM methods. However, consulting p-value tables we conclude that there are no significant difference among them. For the other tail of the performance spectrum, the SSGMM and LapSVM share the spot for worse performance, since they alternate as having the highest average rank, while no significant difference is observed between them. We present the possible inferences from the joint examination of average rank and p-value tables.

25%	50%	75%	90%
ASB > LapSVM	ASB > LapSVM	ASB > LapSVM	ASB > LapSVM
ASB > SSGMM	ASB > SSGMM	ASB > SSGMM	ASB > SSGMM
LabelP > SSGMM	LabelP > SSGMM	LabelP > SSGMM	LabelP > SSGMM
LabelS > SSGMM	LabelS > SSGMM	LabelS > SSGMM	LabelS > SSGMM
SelfT > LapSVM	SelfT > LapSVM	SelfT > LapSVM	SelfT > LapSVM
TSVM > LapSVM	TSVM > LapSVM	TSVM > LapSVM	TSVM > LapSVM
SelfT > SSGMM	SelfT > SSGMM	SelfT > SSGMM	SelfT > SSGMM
TSVM > SSGMM	TSVM > SSGMM	TSVM > SSGMM	TSVM > SSGMM
ASB > LabelP	ASB > LabelP	SelfT > LabelP	LabelP > LapSVM
ASB > LabelS	ASB > LabelS	LabelP > LapSVM	LabelS > LapSVM
ASB > SemiB	ASB > SemiB	LabelS > LapSVM	SemiB > LapSVM
SelfT > LabelP	SelfT > LabelP	SelfT > LabelS	SemiB > SSGMM
LabelP > SemiB	LabelP > SemiB	SemiB > LapSVM	
SelfT > LabelS	SelfT > LabelS	SemiB > SSGMM	
LabelS > SemiB	LabelS > SemiB		
SelfT > SemiB	SelfT > SemiB		
TSVM > SemiB	TSVM > SemiB		
	LabelP > LapSVM		
	LabelS > LapSVM		

Table 5.18: Possible inferences (Transductive global accuracy).

From a comparison with the inductive experiments it's evident the higher incidence of statistically significant results. We attribute this to the bigger size of the datasets, once the initial split between the training and testing sets are not required, the training set is 25% larger. There are 8 cases of unequivocally outperformance between methods:

- Assemble > LapSVM;
- Assemble > SSGMM;
- Label Propagation > SSGMM;
- Label Spreading > SSGMM;
- Self-Training > LapSVM;
- TSVM > LapSVM;
- Self-Training > SSGMM;
- TSVM > SSGMM.

Lastly, we present the average training time of each algorithm in the transductive setup at Table 5.19.

Algorithm	Average Training Time (s)	Normalized Time
Label Spreading	2.36	1.00
TSVM	21.71	9.20
Label Propagation	30.69	13.00
Self-Training	42.18	17.87
SemiBoost	95.57	40.50
SSGMM	161.34	68.36
LapSVM	180.12	76.32
Assemble	924.73	391.83

Table 5.19: Average training time (Transductive).

The Label Spreading method maintains its position as the fasted algorithm with a 10-fold advantage over the second, while the Assemble is solidified as the slowest. The TSVM, as a model exclusively to the transductive setup, has the second best performance in time, as well as one of the top performances in the global accuracy metric. As for the remaining algorithms, they maintain their relative positioning with the exception of the Self-Training and Label Propagation which had their positions swapped.

Positive Accuracy

Ratio	ASB	LabelP	LabelS	LapSVM	SelfT	SemiB	SSGMM	TSVM
0.25	3.36	3.45	2.61	5.90	2.75	5.58	2.50	2.24
0.50	3.34	3.48	2.80	6.28	3.05	5.39	2.44	1.90
0.75	2.69	3.18	2.86	5.48	2.86	3.75	2.08	1.97
0.90	2.20	2.34	2.32	3.85	2.27	2.53	1.86	1.78

Table 5.20: Average rank of algorithms per ratio (Transductive positive accuracy).

For the positive accuracy, we see that the TSVM is the best performer for every ratio, with the SSGMM as a close second, however, there is no statistical significance between them at any ratio. The worst performer is the LapSVM in every ratio, being a source of a considerable portion of the significant results when compared to the other methods. The possible inferences are displayed in Table 5.21.

25%	50%	75%	90%
ASB > LapSVM	ASB > LapSVM	ASB > LapSVM	ASB > LapSVM
ASB > SemiB	ASB > SemiB	ASB > SemiB	LabelP > LapSVM
TSVM > ASB	TSVM > ASB	TSVM > ASB	LabelS > LapSVM
LabelS > LabelP	LabelS > LabelP	LabelP > LapSVM	SelfT > LapSVM
LabelP > LapSVM	LabelP > LapSVM	TSVM > LabelP	SemiB > LapSVM
LabelP > SemiB	LabelP > SemiB	LabelS > LapSVM	SSGMM > LapSVM
TSVM > LabelP	TSVM > LabelP	TSVM > LabelS	TSVM > LapSVM
LabelS > LapSVM	LabelS > LapSVM	SelfT > LapSVM	TSVM > SemiB
LabelS > SemiB	LabelS > SemiB	SemiB > LapSVM	
SelfT > LapSVM	TSVM > LabelS	SSGMM > LapSVM	
SSGMM > LapSVM	SelfT > LapSVM	TSVM > LapSVM	
TSVM > LapSVM	SSGMM > LapSVM	SelfT > SemiB	
SelfT > SemiB	TSVM > LapSVM	TSVM > SelfT	
SSGMM > SemiB	SelfT > SemiB	TSVM > SemiB	
TSVM > SemiB	TSVM > SelfT		
	SSGMM > SemiB		
	TSVM > SemiB		

Table 5.21: Possible inferences (Transductive positive accuracy).

Next, we point the 7 cases of algorithms that outperformed another in every ratio scenario.

- Assemble > LapSVM;
- LabelP > LapSVM;
- LabelS > LapSVM;
- SelfT > LapSVM;
- SSGMM > LapSVM;
- TSVM > LapSVM;
- TSVM > SemiB.

With the exception of the dominance of the TSVM over the SemiBoost, this just displays the poor performance of the LapSVM algorithm in the positive accuracy metric.

Negative Accuracy

Ratio	ASB	LabelP	LabelS	LapSVM	SelfT	SemiB	SSGMM	TSVM
0.25	2.91	2.89	3.77	1.38	2.52	1.39	5.03	3.93
0.50	3.00	2.98	3.55	1.28	2.43	1.53	4.75	3.79
0.75	2.49	2.61	2.84	1.17	2.16	1.86	4.14	3.28
0.90	1.86	2.18	2.09	1.04	1.61	1.63	3.14	2.48

Table 5.22: Average rank of algorithms per ratio (Transductive negative accuracy).

In contrast to the other experiments and metrics, the negative accuracy in the transductive setup shows a outstanding performance of the LapSVM, with it being the top performer at every ratio. On the other hand, the SSGMM is the worst performer for all ratios, opposing its good performance in the positive metric. The remaining algorithms have an overall similar performance, with the possible inferences being displayed on Table 5.23

25%	50%	75%	90%
LapSVM > ASB	LapSVM > ASB	LapSVM > ASB	LapSVM > ASB
SemiB > ASB	SemiB > ASB	ASB > SSGMM	ASB > SSGMM
ASB > SSGMM	ASB > SSGMM	LapSVM > LabelP	LapSVM > LabelP
LabelP > LabelS	LapSVM > LabelP	LabelP > SSGMM	LabelP > SSGMM
LapSVM > LabelP	SemiB > LabelP	LapSVM > LabelS	LapSVM > LabelS
SemiB > LabelP	LabelP > SSGMM	SemiB > LabelS	LabelS > SSGMM
LabelP > SSGMM	LapSVM > LabelS	LabelS > SSGMM	LapSVM > SSGMM
LapSVM > LabelS	SelfT > LabelS	LapSVM > SelfT	LapSVM > TSVM
SelfT > LabelS	SemiB > LabelS	LapSVM > SemiB	SelfT > SSGMM
SemiB > LabelS	LabelS > SSGMM	LapSVM > SSGMM	SemiB > SSGMM
LabelS > SSGMM	LapSVM > SelfT	LapSVM > TSVM	TSVM > SSGMM
LapSVM > SelfT	LapSVM > SSGMM	SelfT > SSGMM	
LapSVM > SSGMM	LapSVM > TSVM	SemiB > SSGMM	
LapSVM > TSVM	SemiB > SelfT	TSVM > SSGMM	
SemiB > SelfT	SelfT > SSGMM		
SelfT > SSGMM	SemiB > SSGMM		
SemiB > SSGMM	SemiB > TSVM		
SemiB > TSVM	TSVM > SSGMM		
TSVM > SSGMM			

Table 5.23: Possible inferences (Transductive negative accuracy).

There are 11 cases of absolute outperformance between algorithms. They are:

- LapSVM > Assemble;
- Assemble > SSGMM;
- LapSVM > LabelP;
- LabelP > SSGMM;
- LapSVM > LabelS;
- LabelS > SSGMM;
- LapSVM > SSGMM;
- LapSVM > TSVM;
- SelfT > SSGMM;
- SemiB > SSGMM;
- TSVM > SSGMM.

It is easy to see that they represent the dominance of the LapSVM algorithm, or the poor performance of the SSGMM.

Chapter 6

Conclusion

As the semi-supervised learning field expands, different approaches and algorithms are proposed to tackle different learning problems. However, these methods are usually presented as top performers on cherry-picked datasets, with very little discussion on their performance on a diverse selection of datasets, as well as lacking a satisfactory statistical analysis of the results. With this critique in mind, we presented the current work aiming to supplement the field with a quantitative comparison of semi-supervised algorithms in diverse scenarios, and with a necessary statistical analysis.

The execution of two classes of experiments, inductive and transductive, allowed for a more complete evaluation of different scenarios where semi-supervision can be applied. The binary results allow us to draw useful conclusion on the comparative performance of popular semi-supervised algorithms. Simple methods such as Self-Training and Label Spreading have been proven to be among the best performers, while more complex ones such as the LapSVM, SemiBoost and SSGMM didn't perform as expected. Moreover, the average training time shown of each method further corroborates the appeal of such simpler methods, with the noticeable 40 times higher training time of the Assemble in comparison with the Self-Training, for a somewhat equivalent performance. On the other hand, the positive and negative accuracy have shown that some models are more appropriate depending on what kind of error is desired to be minimized. The SSGMM have shown a significant improvement when the positive accuracy is taken into consideration. Similarly, the SemiBoost results suggest the algorithm is more appropriate when the negative accuracy is an important factor. In contrast to the binary configuration, the multi-class experiments showed no relevant results. This is conjectured to be a consequence of the lower amount of datasets for this setting, since only 17 of the 44 used datasets are multi-class. As for the transductive setup, we observe a higher number of significant results. The relative performance of the methods remains similar to the inductive setup, with the new TSVM algorithm being one of the best performers in the global and positive accuracy as discussed.

Finally, the study fulfils its proposed goal, delivering a quantitative analysis of popular semi-supervised algorithms, focused on tabular data. The statistical rigor of the study is an attempt to provide a trustworthy source of information regarding the performance of such algorithms. Being expected of the reader a correct assessment of their learning problem, and the correct usage of the results presented here and the possible inferences as well as their limitations. We also aim to inspire further research on the field of semi-supervision, an area that has been gaining notoriety for its attempt to solve well-known limitations of supervised and unsupervised learning paradigms.

Bibliography

[1] 20-newsgroup dataset. `http://qwone.com/~jason/20Newsgroups/`.

[2] Cmu world wide knowledge base project. `https://www.cs.cmu.edu/afs/cs/project/theo-11/www/wwkb/`.

[3] Dana Angluin and Philip D. Laird. Learning from noisy examples. *Machine Learning*, 2:343–370, 1988.

[4] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7(85):2399–2434, 2006.

[5] Kristin P. Bennett, Ayhan Demiriz, and R. Maclin. Exploiting unlabeled data in ensemble methods. *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[6] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. Openml benchmarking suites. *arXiv:1708.03731v2 [stat.ML]*, 2019.

[7] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. *Proceedings of the Annual ACM Conference on Computational Learning Theory*, 10 2000.

[8] T. Diethe. 13 benchmark datasets derived from the UCI, DELVE and STATLOG repositories. `https://github.com/tdiethe/gunnar_raetsch_benchmark_datasets/`, 2015.

[9] B. Efron and R.J. Tibshirani. *An Introduction to the Bootstrap.* Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1994.

[10] Yue Fan, Anna Kukleva, and Bernt Schiele. Revisiting consistency regularization for semi-supervised learning, 2021.

[11] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[12] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). *The Annals of Statistics*, 28(2):337 – 407, 2000.

[13] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

[14] Zhe Huang, Gary Long, Benjamin Wessler, and Michael C. Hughes. A new semi-supervised learning benchmark for classifying view and diagnosing aortic stenosis from echocardiograms, 2021.

[15] J.J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.

[16] J.J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.

[17] Lin-Han Jia, Lan-Zhe Guo, Zhi Zhou, and Yu-Feng Li. Lamda-ssl repository. https://github.com/YGZWQZD/LAMDA-SSL, 2022.

[18] Lin-Han Jia, Lan-Zhe Guo, Zhi Zhou, and Yu-Feng Li. Lamda-ssl: Semi-supervised learning in python. *arXiv preprint arXiv:2208.04610*, 2022.

[19] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning*, 1999.

[20] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning, 2017.

[21] Wei Liu, Jun Wang, and Shih-Fu Chang. Robust and scalable graph-based semisupervised learning. *Proceedings of the IEEE*, 100(9):2624–2638, 2012.

[22] Pavan Kumar Mallapragada, Rong Jin, Anil K. Jain, and Yi Liu. Semiboost: Boosting for semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(11):2000–2014, 2009.

[23] Geoffrey J. McLachlan and Thriyambakam Krishnan. The em algorithm and extensions. 1996.

[24] C.L. Blake D.J. Newman and C.J. Merz. UCI repository of machine learning databases, 1998.

[25] Kamal Nigam and Rayid Ghani. Analyzing the effectiveness and applicability of co-training. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, CIKM '00, page 86–93, New York, NY, USA, 2000. Association for Computing Machinery.

[26] Avital Oliver, Augustus Odena, Colin A Raffel, Ekin Dogus Cubuk, and Ian Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. *Advances in neural information processing systems*, 31, 2018.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[28] Jan Van Rjin. Openml-cc18 curated classification benchmark. https://www.openml.org/search?type=study&study_type=task&id=99&sort=runs_included, 2018.

[29] H. Schutze. Dimensions of meaning. In *Supercomputing '92:Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, pages 787–796, 1992.

[30] B.M. Shahshahani and D.A. Landgrebe. The effect of unlabeled samples in reducing the small sample size problem and mitigating the hughes phenomenon. *IEEE Transactions on Geoscience and Remote Sensing*, 32(5):1087–1095, 1994.

[31] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 596–608. Curran Associates, Inc., 2020.

[32] Jong-Chyi Su, Zezhou Cheng, and Subhransu Maji. A realistic evaluation of semi-supervised learning for fine-grained classification, 2021.

[33] Jesper E. van Engelen and Holger H. Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, November 2019.

[34] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[35] Vladimir Naumovich Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10 5:988–99, 1999.

[36] Yidong Wang, Hao Chen, Yue Fan, Wang Sun, Ran Tao, Wenxin Hou, Renjie Wang, Linyi Yang, Zhi Zhou, Lan-Zhe Guo, et al. Usb: A unified semi-supervised learning benchmark for classification. *Advances in Neural Information Processing Systems*, 35:3938–3961, 2022.

[37] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Annual Meeting of the Association for Computational Linguistics*, 1995.

[38] Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Olkopf. Learning with local and global consistency. *Advances in Neural Information Processing Systems 16*, 16, 03 2004.

[39] Zhi-Hua Zhou and Ming Li. Tri-training: Exploiting unlabeled data using three classifiers. *Knowledge and Data Engineering, IEEE Transactions on*, 17:1529– 1541, 12 2005.

[40] Xiaojin Zhu. Semi-supervised learning literature survey. *Comput Sci, University of Wisconsin-Madison*, 2, 07 2008.

[41] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.

[42] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *International Conference on Machine Learning*, 2003.

Appendix A

Proofs

A.1 Proposition 1.5.1

We rewrite Eq. (3.32b by making the substitution $y_i = F_t(\vec{x}_i) + w_{t+1}f_{t+1}(\vec{x}_i)$ for every unlabeled point $\vec{x}_i$.

$$\mathcal{E}(\vec{y}, S) = \sum_{i=1}^{l}\sum_{j=l+1}^{l+u} S_{ij}e^{-2y_i(F_t(\vec{x}_j)+w_{t+1}f_{t+1}(\vec{x}_j))} + \gamma \sum_{i=l+1}^{l+u}\sum_{j=l+1}^{l+u} S_{ij}e^{F_t(\vec{x}_i)+w_{t+1}f_{t+1}(\vec{x}_i)-F_t(\vec{x}_j)-w_{t+1}f_{t+1}(\vec{x}_j)} \tag{A.1}$$

$$\mathcal{E}(\vec{y}, S) = \sum_{i=1}^{l}\sum_{j=l+1}^{l+u} S_{ij}e^{-2y_iF_t(\vec{x}_j)}e^{-2y_iw_{t+1}f_{t+1}(\vec{x}_j)} + \gamma \sum_{i=l+1}^{l+u}\sum_{j=l+1}^{l+u} S_{ij}e^{F_t(\vec{x}_i)-F_t(\vec{x}_j)}e^{w_{t+1}(f_{t+1}(\vec{x}_i)-f_{t+1}(\vec{x}_j))}. \tag{A.2}$$

Using the inequality $e^{a(x-y)} \leq 1/2(e^{2ax} + e^{-2ay})$ on the second term of Eq. (A.2):

$$\begin{aligned}\mathcal{E}(\vec{y}, S) \leq &\sum_{i=1}^{l}\sum_{j=l+1}^{l+u} S_{ij}e^{-2y_iF_t(\vec{x}_j)}e^{-2y_iw_{t+1}f_{t+1}(\vec{x}_j)} \\ &+ \frac{\gamma}{2}\sum_{i=l+1}^{l+u}\sum_{j=l+1}^{l+u} S_{ij}e^{F_t(\vec{x}_i)-F_t(\vec{x}_j)}(e^{2w_{t+1}f_{t+1}(\vec{x}_i)} + e^{-2w_{t+1}f_{t+1}(\vec{x}_j)}).\end{aligned} \tag{A.3}$$

Let $\delta(a, b) = 1$ if $a = b$ and 0 otherwise, and let the first double summation be represented by T_1 and the second by T_2.

$$T_1 = \sum_{j=l+1}^{l+u}\sum_{i=1}^{l} S_{ij}e^{-2F_t(\vec{x}_j)}e^{-2w_{t+1}f_{t+1}(\vec{x}_j)}\delta(y_i, 1) + \sum_{j=l+1}^{l+u}\sum_{i=1}^{l} S_{ij}e^{2F_t(\vec{x}_j)}e^{2w_{t+1}f_{t+1}(\vec{x}_j)}\delta(y_i, -1) \tag{A.4}$$

$$T_1 = \sum_{j=l+1}^{l+u} e^{-2w_{t+1}f_{t+1}(\vec{x}_j)} \sum_{i=1}^{l} S_{ij} e^{-2F_t(\vec{x}_j)} \delta(y_i, 1) + \sum_{j=l+1}^{l+u} e^{2w_{t+1}f_{t+1}(\vec{x}_j)} \sum_{i=1}^{l} S_{ij} e^{2F_t(\vec{x}_j)} \delta(y_i, -1) \tag{A.5}$$

$$T_2 = \frac{\gamma}{2} \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_i)-F_t(\vec{x}_j)} e^{2w_{t+1}f_{t+1}(\vec{x}_i)} + \frac{\gamma}{2} \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_i)-F_t(\vec{x}_j)} e^{-2w_{t+1}f_{t+1}(\vec{x}_j)}. \tag{A.6}$$

By the symmetry of the double summation we can re-index the first term:

$$T_2 = \frac{\gamma}{2} \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j)-F_t(\vec{x}_i)} e^{2w_{t+1}f_{t+1}(\vec{x}_j)} + \frac{\gamma}{2} \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_i)-F_t(\vec{x}_j)} e^{-2w_{t+1}f_{t+1}(\vec{x}_j)} \tag{A.7}$$

$$T_2 = \frac{\gamma}{2} \sum_{j=l+1}^{l+u} e^{2w_{t+1}f_{t+1}(\vec{x}_j)} \sum_{i=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j)-F_t(\vec{x}_i)} + \frac{\gamma}{2} \sum_{j=l+1}^{l+u} e^{-2w_{t+1}f_{t+1}(\vec{x}_j)} \sum_{i=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_i)-F_t(\vec{x}_j)}. \tag{A.8}$$

Adding T_1 and T_2 and regrouping by common terms we would get two new terms, say T_1' and T_1', as below:

$$T_1' = \sum_{j=l+1}^{l+u} e^{-2w_{t+1}f_{t+1}(\vec{x}_j)} \left[\sum_{i=1}^{l} S_{ij} e^{-2F_t(\vec{x}_j)} \delta(y_i, 1) + \frac{\gamma}{2} \sum_{i=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_i)-F_t(\vec{x}_j)} \right] \tag{A.9}$$

$$T_2' = \sum_{j=l+1}^{l+u} e^{2w_{t+1}f_{t+1}(\vec{x}_j)} \left[\sum_{i=1}^{l} S_{ij} e^{2F_t(\vec{x}_j)} \delta(y_i, -1) + \frac{\gamma}{2} \sum_{i=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j)-F_t(\vec{x}_i)} \right]. \tag{A.10}$$

Finally, this leads to

$$\mathcal{E}(\vec{y}, S) \leq \sum_{j=l+1}^{l+u} e^{-2w_{t+1}f_{t+1}(\vec{x}_j)} p_j + e^{2w_{t+1}f_{t+1}(\vec{x}_j)} q_j \tag{A.11}$$

where

$$p_j = \sum_{i=1}^{l} S_{ij} e^{-2F_t(\vec{x}_j)} \delta(y_i, 1) + \frac{\gamma}{2} \sum_{i=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_i)-F_t(\vec{x}_j)} \tag{A.12}$$

$$q_j = \sum_{i=1}^{l} S_{ij} e^{2F_t(\vec{x}_j)} \delta(y_i, -1) + \frac{\gamma}{2} \sum_{i=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j) - F_t(\vec{x}_i)}. \tag{A.13}$$

■

A.2 Proposition 1.5.2

We rewrite Eq. (3.33) using the inequality $e^{ax} \leq e^{a} + e^{-a} - 1 + ax, \ \forall x \in [-1, 1]$

$$\mathcal{E}'(\vec{y}, S) = \sum_{i=l+1}^{l+u} e^{-2w_{t+1} f_{t+1}(\vec{x}_i)} p_i + e^{2w_{t+1} f_{t+1}(\vec{x}_i)} q_i \tag{A.14}$$

$$\mathcal{E}'(\vec{y}, S) \leq \sum_{i=l+1}^{l+u} (e^{-2w_{t+1}} + e^{2w_{t+1}} - 1 - 2w_{t+1} f_{t+1}(\vec{x}_i)) p_i + (e^{2w_{t+1}} + e^{-2w_{t+1}} - 1 + 2w_{t+1} f_{t+1}(\vec{x}_i)) q_i \tag{A.15}$$

$$\mathcal{E}'(\vec{y}, S) \leq \sum_{i=l+1}^{l+u} e^{-2w_{t+1}}(p_i + q_i) + e^{2w_{t+1}}(p_i + q_i) - (p_i + q_i) - 2w_{t+1} f_{t+1}(\vec{x}_i)(p_i - q_i) \tag{A.16}$$

$$\mathcal{E}'(\vec{y}, S) \leq \sum_{i=l+1}^{l+u} (p_i + q_i)(e^{2w_{t+1}} + e^{-2w_{t+1}} - 1) - \sum_{i=l+1}^{l+u} 2w_{t+1} f_{t+1}(\vec{x}_i)(p_i - q_i). \tag{A.17}$$

■

A.3 Proposition 1.5.4

We begin rewriting Eq. (3.33)

$$\mathcal{E}'(\vec{y}, S) = \sum_{i=l+1}^{l+u} e^{-2w_{t+1} f_{t+1}(\vec{x}_i)} p_i + e^{2w_{t+1} f_{t+1}(\vec{x}_i)} q_i \tag{A.18}$$

$$\begin{aligned} \mathcal{E}'(\vec{y}, S) = & \sum_{i=l+1}^{l+u} e^{-2w_{t+1}} p_i \delta(f_{t+1}(\vec{x}_i), 1) + e^{2w_{t+1}} p_i \delta(f_{t+1}(\vec{x}_i), -1) \\ & + \sum_{i=l+1}^{l+u} e^{2w_{t+1}} q_i \delta(f_{t+1}(\vec{x}_i), 1) + e^{-2w_{t+1}} q_i \delta(f_{t+1}(\vec{x}_i), -1) \end{aligned} \tag{A.19}$$

$$\mathcal{E}'(\vec{y}, S) = \sum_{i=l+1}^{l+u} e^{-2w_{t+1}}(p_i\delta(f_{t+1}(\vec{x}_i), 1) + q_i\delta(f_{t+1}(\vec{x}_i), -1)) + \sum_{i=l+1}^{l+u} e^{2w_{t+1}}(p_i\delta(f_{t+1}(\vec{x}_i), -1) + q_i\delta(f_{t+1}(\vec{x}_i), 1)) \quad (A.20)$$

$$\mathcal{E}'(\vec{y}, S) = e^{2w_{t+1}}k + e^{-2w_{t+1}}k' \quad (A.21)$$

where

$$k = \sum_{i=l+1}^{l+u} p_i\delta(f_{t+1}(\vec{x}_i), -1) + q_i\delta(f_{t+1}(\vec{x}_i), 1) \quad (A.22)$$

$$k' = \sum_{i=l+1}^{l+u} p_i\delta(f_{t+1}(\vec{x}_i), 1) + q_i\delta(f_{t+1}(\vec{x}_i), -1). \quad (A.23)$$

We can simplify k' as

$$k' = \sum_{i=l+1}^{l+u} p_i(1 - \delta(f_{t+1}(\vec{x}_i), -1)) + q_i(1 - \delta(f_{t+1}(\vec{x}_i), 1)) \quad (A.24)$$

$$k' = \sum_{i=l+1}^{l+u} (p_i + q_i) - \sum_{i=l+1}^{l+u} p_i\delta(f_{t+1}(\vec{x}_i), -1) + q_i\delta(f_{t+1}(\vec{x}_i), 1) \quad (A.25)$$

$$k' = \eta - k \quad (A.26)$$

where $\eta = \sum_{i=l+1}^{l+u} (p_i + q_i)$. This leads to

$$\mathcal{E}'(\vec{y}, S) = e^{2w_{t+1}}k + e^{-2w_{t+1}}(\eta - k) \quad (A.27)$$

$$\mathcal{E}'(\vec{y}, S) = e^{2w_{t+1}}\eta\left(\frac{k}{\eta}\right) + e^{-2w_{t+1}}\eta\left(1 - \frac{k}{\eta}\right). \quad (A.28)$$

The value $\epsilon = (k/\eta)$ is the weighted error of the classifier f_{t+1}, leading to the final form of the function $\mathcal{E}'$:

$$\mathcal{E}'(\vec{y}, S) = e^{2w_{t+1}}\eta\epsilon + e^{-2w_{t+1}}\eta(1 - \epsilon) \quad (A.29)$$

Deriving Eq. (A.29) with respect to w_{t+1} and equating it to 0 we get

$$\frac{d\mathcal{E}'}{dw_{t+1}} = 2\eta\epsilon e^{2w_{t+1}} - 2\eta(1 - \epsilon)e^{-2w_{t+1}} = 0 \quad (A.30)$$

$$e^{4w_{t+1}} = \frac{1-\epsilon}{\epsilon} \tag{A.31}$$

$$w_{t+1} = \frac{1}{4}\ln\left(\frac{1-\epsilon}{\epsilon}\right) \tag{A.32}$$

or without ϵ

$$w_{t+1} = \frac{1}{4}\ln\left(\frac{\sum_{i=l+1}^{l+u} p_i\delta(f_{t+1}(\vec{x}_i),1) + q_i\delta(f_{t+1}(\vec{x}_i),-1)}{\sum_{i=l+1}^{l+u} p_i\delta(f_{t+1}(\vec{x}_i),-1) + q_i\delta(f_{t+1}(\vec{x}_i),1)}\right). \tag{A.33}$$

■

A.4 Proposition 1.5.5

Taking the upper error function at the $(t+1)$th iteration

$$\mathcal{E}'_{t+1}(\vec{y},S) = \sum_{i=l+1}^{l+u} e^{-2w_{t+1}f_{t+1}(\vec{x}_i)}p_i + e^{2w_{t+1}f_{t+1}(\vec{x}_i)}q_i \tag{A.34}$$

and replacing w_{t+1} with its optimal value

$$w_{t+1} = \frac{1}{4}\ln\left(\frac{\sum_{i=l+1}^{l+u} p_i\delta(f_{t+1}(\vec{x}_i),1) + q_i\delta(f_{t+1}(\vec{x}_i),-1)}{\sum_{i=l+1}^{l+u} p_i\delta(f_{t+1}(\vec{x}_i),-1) + q_i\delta(f_{t+1}(\vec{x}_i),1)}\right) \tag{A.35}$$

$$w_{t+1} = \frac{1}{4}\ln\left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right) \tag{A.36}$$

we get

$$\mathcal{E}'_{t+1}(\vec{y},S) = \sum_{i=l+1}^{l+u} p_i\left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{-\frac{f_{t+1}(\vec{x}_i)}{2}} + q_i\left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{\frac{f_{t+1}(\vec{x}_i)}{2}} \tag{A.37}$$

$$\begin{aligned}\mathcal{E}'_{t+1}(\vec{y},S) = {} & \sum_{i=l+1}^{l+u} p_i\left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{-1/2}\delta(f_{t+1}(\vec{x}_i),1) + p_i\left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{1/2}\delta(f_{t+1}(\vec{x}_i),-1) \\ & + \sum_{i=l+1}^{l+u} q_i\left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{1/2}\delta(f_{t+1}(\vec{x}_i),1) + q_i\left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{-1/2}\delta(f_{t+1}(\vec{x}_i),-1)\end{aligned} \tag{A.38}$$

$$\mathcal{E}'_{t+1}(\vec{y}, S) = \left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{-1/2} \sum_{i=l+1}^{l+u} (p_i\delta(f_{t+1}(\vec{x}_i), 1) + q_i\delta(f_{t+1}(\vec{x}_i), -1)) + \left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{1/2} \sum_{i=l+1}^{l+u} (p_i\delta(f_{t+1}(\vec{x}_i), -1) + q_i\delta(f_{t+1}(\vec{x}_i), 1)) \quad (A.39)$$

$$\mathcal{E}'_{t+1}(\vec{y}, S) = \left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{-1/2} \sum_{i=l+1}^{l+u} (p_i(1-\delta(f_{t+1}(\vec{x}_i), -1)) + q_i(1-\delta(f_{t+1}(\vec{x}_i), 1))) + \left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{1/2} \sum_{i=l+1}^{l+u} (p_i\delta(f_{t+1}(\vec{x}_i), -1) + q_i\delta(f_{t+1}(\vec{x}_i), 1)) \quad (A.40)$$

$$\mathcal{E}'_{t+1}(\vec{y}, S) = \left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{-1/2} (\eta_{t+1} - \eta_{t+1}\epsilon_{t+1}) + \left(\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}\right)^{1/2} \epsilon_{t+1}\eta_{t+1} \quad (A.41)$$

$$\mathcal{E}'_{t+1}(\vec{y}, S) = (1-\epsilon_{t+1})^{1/2}\epsilon_{t+1}^{1/2}\eta_{t+1} + (1-\epsilon_{t+1})^{1/2}\epsilon_{t+1}^{1/2}\eta_{t+1} \quad (A.42)$$

$$\mathcal{E}'_{t+1}(\vec{y}, S) = 2\eta_{t+1}\epsilon_{t+1}\sqrt{\frac{1-\epsilon_{t+1}}{\epsilon_{t+1}}} \quad (A.43)$$

using *Lemma 1.5.6*

$$\mathcal{E}'_{t+1}(\vec{y}, S) = 2\mathcal{E}_t(\vec{y}, S)\epsilon_{t+1}e^{2w_{t+1}}. \quad (A.44)$$

Isolating ϵ_{t+1} in Eq. (A.36)

$$\epsilon_{t+1} = \frac{1}{1+e^{4w_{t+1}}} \quad (A.45)$$

and substituting it in Eq. (A.44)

$$\mathcal{E}'_{t+1}(\vec{y}, S) = \mathcal{E}_t(\vec{y}, S)\frac{2e^{2w_{t+1}}}{1+e^{4w_{t+1}}} = \mathcal{E}_t(\vec{y}, S)\frac{1}{1/2(e^{-2w_{t+1}} + e^{2w_{t+1}})} \quad (A.46)$$

$$\mathcal{E}'_{t+1}(\vec{y}, S) = \frac{\mathcal{E}_t(\vec{y}, S)}{cosh(2w_{t+1})}. \quad (A.47)$$

Since $\mathcal{E}_t(\vec{y}, S)$ is bounded by $\mathcal{E}'_t(\vec{y}, S)$

$$\mathcal{E}'_{t+1}(\vec{y}, S) \leq \frac{\mathcal{E}'_t(\vec{y}, S)}{cosh(2w_{t+1})} \quad (A.48)$$

and since $cosh(x) \leq e^x$ for $x > 0$

$$\mathcal{E}'_{t+1}(\vec{y}, S) \leq \frac{\mathcal{E}'_t(\vec{y}, S)}{e^{2w_{t+1}}} \tag{A.49}$$

$$\mathcal{E}'_{t+1}(\vec{y}, S) \leq \mathcal{E}'_t(\vec{y}, S) e^{-2w_{t+1}}. \tag{A.50}$$

If we repeatedly substitute the upper error on the right hand-side we get

$$\mathcal{E}'_{t+1}(\vec{y}, S) \leq \mathcal{E}'_0(\vec{y}, S) e^{-\sum_{i=1}^{t+1} 2w_i} \tag{A.51}$$

where

$$\mathcal{E}'_0(\vec{y}, S) = \sum_{i=l+1}^{l+u} \left[\sum_{j=1}^{l} S_{ij} + \gamma \sum_{j=l+1}^{l+u} S_{ij} \right]. \tag{A.52}$$

■

A.5 Lemma 1.5.6

Lemma 1.5.6: Let

$$\eta_{t+1} = \sum_{i=l+1}^{l+u} p_i + q_i \tag{A.53}$$

where

$$p_i = \sum_{j=1}^{l} S_{ij} e^{-2F_t(\vec{x}_i)} \delta(y_j, 1) + \frac{\gamma}{2} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j) - F_t(\vec{x}_i)} \tag{A.54}$$

$$q_i = \sum_{j=1}^{l} S_{ij} e^{2F_t(\vec{x}_i)} \delta(y_j, -1) + \frac{\gamma}{2} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_i) - F_t(\vec{x}_j)} \tag{A.55}$$

then $\eta_{t+1} = \mathcal{E}_t(\vec{y}, S)$.

Proof: We begin calculating $p_i + q_i$ for a generic point $\vec{x}_i$. Let $p_i + q_i = A_i + B_i$ such that A_i is the sum of the first term of p_i with the first term of q_i, and B_i is the sum of the second term of p_i with the second term of q_i

$$\sum_{i=l+1}^{l+u} p_i + q_i = \sum_{i=l+1}^{l+u} (\sum_{j=1}^{l} S_{ij} e^{-2F_t(\vec{x}_i)} \delta(y_j, 1) + \frac{\gamma}{2} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j) - F_t(\vec{x}_i)}$$
$$+ \sum_{j=1}^{l} S_{ij} e^{2F_t(\vec{x}_i)} \delta(y_j, -1) + \frac{\gamma}{2} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_i) - F_t(\vec{x}_j)}) \tag{A.56}$$

$$\sum_{i=l+1}^{l+u} p_i + q_i = \sum_{i=l+1}^{l+u} \sum_{j=1}^{l} \left[S_{ij} e^{-2F_t(\vec{x}_i)} \delta(y_j, 1) + S_{ij} e^{2F_t(\vec{x}_i)} \delta(y_j, -1) \right] + \frac{\gamma}{2} \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} \left[S_{ij} e^{F_t(\vec{x}_j) - F_t(\vec{x}_i)} + S_{ij} e^{F_t(\vec{x}_i) - F_t(\vec{x}_j)} \right] \tag{A.57}$$

$$\sum_{i=l+1}^{l+u} p_i + q_i = \sum_{i=l+1}^{l+u} \sum_{j=1}^{l} S_{ij} e^{-2F_t(\vec{x}_i) y_j} + \frac{\gamma}{2} \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j) - F_t(\vec{x}_i)} + \frac{\gamma}{2} \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_i) - F_t(\vec{x}_j)}. \tag{A.58}$$

Since $F_t(\vec{x}_j) = y_j$ for a labeled example $\vec{x}_j$, and using the symmetry of the double summation to re-index the third term

$$\sum_{i=l+1}^{l+u} p_i + q_i = \sum_{i=l+1}^{l+u} \sum_{j=1}^{l} S_{ij} e^{-2y_i y_j} + \frac{\gamma}{2} \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j) - F_t(\vec{x}_i)} + \frac{\gamma}{2} \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j) - F_t(\vec{x}_i)} \tag{A.59}$$

$$\sum_{i=l+1}^{l+u} p_i + q_i = \sum_{i=l+1}^{l+u} \sum_{j=1}^{l} S_{ij} e^{-2y_i y_j} + \gamma \sum_{i=l+1}^{l+u} \sum_{j=l+1}^{l+u} S_{ij} e^{F_t(\vec{x}_j) - F_t(\vec{x}_i)}. \tag{A.60}$$

The right hand-side of the Eq. (A.60) is precisely the error function in (3.32b) but at iteration t. Therefore

$$\sum_{i=l+1}^{l+u} p_i + q_i = \mathcal{E}_t(\vec{y}, S). \tag{A.61}$$

■

A.6 Proposition 1.7.1

If $|\mathcal{L}_{i,t-1}| < |\mathcal{L}_{i,t}|$ it's obvious that the numerator on the left-hand side of the inequality is smaller than the numerator on the right-hand side, that is

$$|\mathcal{L}_{i,t-1}| < |\mathcal{L}_{i,t}| \implies \sqrt{\frac{c}{|\mathcal{L} \cup \mathcal{L}_{i,t}|}} < \sqrt{\frac{c}{|\mathcal{L} \cup \mathcal{L}_{i,t-1}|}}. \tag{A.62}$$

Similarly, we want the negative term on the denominator to be smaller on the left-hand side, making the denominator bigger and the fraction smaller. In other words, we want

$$\frac{\eta_L|\mathcal{L}| + e_{i,t}|\mathcal{L}_{i,t}|}{|\mathcal{L} \cup \mathcal{L}_{i,t}|} < \frac{\eta_L|\mathcal{L}| + e_{i,t-1}|\mathcal{L}_{i,t-1}|}{|\mathcal{L} \cup \mathcal{L}_{i,t-1}|}. \tag{A.63}$$

Since the denominator on the left-hand side of inequality (A.63) is already bigger than the denominator on the right-hand side, we just want a smaller numerator on the left so the fraction on the left will be smaller and the inequality (A.63) is satisfied. Mathematically, we want

$$\eta_L|\mathcal{L}| + e_{i,t}|\mathcal{L}_{i,t}| < \eta_L|\mathcal{L}| + e_{i,t-1}|\mathcal{L}_{i,t-1}| \tag{A.64}$$

$$e_{i,t}|\mathcal{L}_{i,t}| < e_{i,t-1}|\mathcal{L}_{i,t-1}|. \tag{A.65}$$

Therefore, if $|\mathcal{L}_{i,t-1}| < |\mathcal{L}_{i,t}|$ and $e_{i,t}|\mathcal{L}_{i,t}| < e_{i,t-1}|\mathcal{L}_{i,t-1}|$ the numerator of the left-hand side of inequality (3.54) will be smaller, while the denominator will be bigger than the equivalent right-hand side, satisfying the inequality.

■

A.7 Proposition 1.7.2

Substituting the new size of $\mathcal{L}_{i,t}$ in condition 1:

$$|\mathcal{L}_{i,t-1}| < *\frac{e_{i,t-1}|\mathcal{L}_{i,t-1}|}{e_{i,t}} - 1. \tag{A.66}$$

We can strengthen the condition by requiring that $|\mathcal{L}_{i,t-1}|$ be smaller than the right-hand side without the ceiling function

$$|\mathcal{L}_{i,t-1}| < \frac{e_{i,t-1}|\mathcal{L}_{i,t-1}|}{e_{i,t}} - 1 < *\frac{e_{i,t-1}|\mathcal{L}_{i,t-1}|}{e_{i,t}} - 1. \tag{A.67}$$

Isolating $|\mathcal{L}_{i,t-1}|$ on the first inequality

$$|\mathcal{L}_{i,t-1}|e_{i,t} < e_{i,t-1}|\mathcal{L}_{i,t-1}| - e_{i,t} \tag{A.68}$$

$$|\mathcal{L}_{i,t-1}|(e_{i,t} - e_{i,t-1}) < -e_{i,t} \tag{A.69}$$

$$|\mathcal{L}_{i,t-1}|(e_{i,t-1} - e_{i,t}) > e_{i,t} \tag{A.70}$$

$$|\mathcal{L}_{i,t-1}| > \frac{e_{i,t}}{e_{i,t-1} - e_{i,t}}. \tag{A.71}$$

Therefore, if Eq. (A.71) is satisfied, condition 1 is also satisfied.

■

A.8 Proposition 1.8.1

First, we prove that the proposed label swapping results in a decrease in the cost function. Notice that since the slack variable ξ_i represents the distance between a point $\vec{x}_i$ and its assigned label's margin, when we swap the labels there are two options: the point falls at or before the new label's margin at a distance $2 - \xi_i \geq 0$, or the point falls after the margin, in which case the slack variable is not necessary and is set to zero. This can be represented by the expression $\xi_i' = \max\{0, 2 - \xi_i\}$, where ξ_i' is the new slack variable.

Let m and n be values that satisfy the inner loop conditions, and without loss of generality let $y_m = 1$ and $y_n = -1$. The cost function presented in (3.59) can be rewritten as

$$\frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{\ell}\xi_i + C_+'\sum_{\substack{y_i=1\\ i\neq m}}\xi_i + C_-'\sum_{\substack{y_i=-1\\ i\neq n}}\xi_i + C_+'\xi_m + C_-'\xi_n. \tag{A.72}$$

However, by the loop condition $\xi_n + \xi_m > 2$, we know that

$$C_+'\xi_m + C_-'\xi_n > C_+'(2 - \xi_n) + C_-'(2 - \xi_m) \tag{A.73}$$

and since they satisfy the loop conditions, we know that

$$C_+'\xi_m + C_-'\xi_n > C_+' \cdot 0 + C_-' \cdot 0. \tag{A.74}$$

By expressions (A.73), (A.74) and the identity $\xi_i' = \max\{0, 2 - \xi_i\}$, we can write

$$C_+'\xi_m + C_-'\xi_n > C_+'\xi_m' + C_-'\xi_n'. \tag{A.75}$$

This shows that the new cost function is smaller after the label swapping.
Assuming that the minimization problem (3.59) is deterministic, and each permutation of labels always yields the same finite cost, there are u points and 2 possible labels for each, resulting in at most 2^u possible values for the cost function, at least one of which is bound to be the minimum. Since with every iteration we have only two options: we fail to find pairs of points that satisfy the conditions and the loop breaks, or we find them and decrease the cost function to a lower value among the 2^u possibilities, bringing it closer to the overall minimum. This makes it clear that the loop will break early if no

improvement can be made, or it will continue until it reaches the minimum value among the 2^u possibilities, where the next loop will fail to find adequate points or we would have a contradiction.

Appendix B

Auxiliary Algorithms

B.1 Select

Algorithm: 15 Select

Input:

$\overline{\mathcal{U}} \subset \mathcal{X}_A \times \mathcal{X}_B$	:: Unlabeled Buffer Dataset
$f_I : \mathcal{X} \to \mathbb{R}^{\lvert\mathcal{Y}\rvert}$	:: View I Classifier
$p \in \mathbb{Z}^+$	:: Number of Positively Labeled Examples to Select
$n \in \mathbb{Z}^+$	:: Number of Negatively Labeled Examples to Select

Output:

$\mathcal{S} \subset \overline{\mathcal{U}}$ :: Confident Unlabeled Examples

```
1. Let $V^+ \in \mathbb{R}^u \leftarrow V_i^+ = (f_I(\vec{x}_i))_1,\ \vec{x}_i \in \overline{\mathcal{U}}$
2. $V^+ \leftarrow$ **Order**$(V^+)$
3. Let $S_p \leftarrow \{\vec{x}_j \in \overline{\mathcal{U}} : (f_I(\vec{x}_j))_1 \in V_{1:p}^+\}$
4. $\overline{\mathcal{U}} \leftarrow \overline{\mathcal{U}} \setminus S_p$
5. Let $V^- \in \mathbb{R}^u \leftarrow V_i^- = (f_I(\vec{x}_i))_2,\ \vec{x}_i \in \overline{\mathcal{U}}$
6. $V^- \leftarrow$ **Order**$(V^-)$
7. Let $S_n \leftarrow \{\vec{x}_j \in \overline{\mathcal{U}} : (f_I(\vec{x}_j))_2 \in V_{1:n}^-\}$
8. $\overline{\mathcal{U}} \leftarrow \overline{\mathcal{U}} \setminus S_n$
9. **return** $S_p \bigcup S_n$
```

B.2 Select’

Algorithm: 16 Select’

Input:

$\overline{\mathcal{U}} \subset \mathcal{X}_A \times \mathcal{X}_B$	:: Unlabeled Buffer Dataset
$f_I : \mathcal{X} \rightarrow \mathbb{R}^{\lvert\mathcal{Y}\rvert}$	:: View I Classifier
$\vec{v} \in \mathbb{Z}^{\lvert\mathcal{Y}\rvert}$	:: Number of Examples To Select For Each Class

Output:

$\mathcal{S} \subset \overline{\mathcal{U}}$ :: Confident Unlabeled Examples

---

```
1. Let $S \leftarrow \emptyset$
2. **for** $k \leftarrow 1$ **to** $|\vec{v}|$ **do**:
3.         Let $V^{(k)} \in \mathbb{R}^u \leftarrow V_i^{(k)} = (f_I(\vec{x}_i))_k,\ \vec{x}_i \in \overline{\mathcal{U}}$
4.         $V^{(k)} \leftarrow$ **Order**$(V^{(k)})$
5.         Let $S_k \leftarrow \{\vec{x}_j \in \overline{\mathcal{U}} : (f_I(\vec{x}_j))_k \in V^{(k)}_{1:v_k}\}$
6.         $\overline{\mathcal{U}} \leftarrow \overline{\mathcal{U}} \setminus S_k$
7.         $S \leftarrow S \bigcup S_k$
8. **return** $S$
```

B.3 BootstrapSample

Algorithm: 17 BootstrapSample

Input:
$\mathcal{L} \subset \mathcal{X} \times \mathcal{Y}$ :: Labeled Dataset

Output:
$\mathcal{L}' \subset \mathcal{X} \times \mathcal{Y}$ :: Bootstrapped Dataset

```
1. let $\mathcal{L}' \leftarrow \emptyset$
2. for $n \leftarrow 1$ to $|\mathcal{L}|$ do:
3.     let $m \in \mathcal{L} \leftarrow$ Choose($\mathcal{L}$) :: Choose a single element
4.     $\mathcal{L}' \leftarrow \mathcal{L}' \cup \{m\}$
5. return $\mathcal{L}'$
```

B.4 Error

Algorithm: 18 Error

Input:
$f_{j,t} : \mathcal{X} \to \mathcal{Y}$ :: Classifier j at iteration t
$f_{k,t} : \mathcal{X} \to \mathcal{Y}$ :: Classifier k at iteration t
$\mathcal{L} \subset \mathcal{X} \times \mathcal{Y}$ :: Labeled dataset

Output:
$e_{i,t} \in \mathbb{R}$ :: Error rate of the combined classifiers $f_{j,t}$ and $f_{k,t}$ at iteration t

```
1. let $C \subset \mathcal{X} \times \mathcal{Y} \leftarrow \{(\vec{x}, y) : (\vec{x}, y) \in \mathcal{L}, f_{j,t}(\vec{x}) = f_{k,t}(\vec{x}) = y\}$
2. let $e_{i,t} \in \mathbb{R} \leftarrow \frac{|L| - |C|}{|L|}$
3. return $e_{i,t}$
```

B.5 Subsample

Algorithm: 19 Subsample

Input:

$\mathcal{L}_{i,t} \subset \mathcal{X} \times \mathcal{Y}$	:: Examples pseudo-labeled by classifiers $f_{j,t}$ and $f_{k,t}$
$\ell'_{i,t} \in \mathbb{N}$	:: Final size of set $\mathcal{L}_{i,t}$

Output:

$\mathcal{L}'_{i,t} \subset \mathcal{L}_{i,t}$ :: New set $\mathcal{L}_{i,t}$

```
1. for $n \leftarrow 1$ to $(|\mathcal{L}_{i,t}| - \ell'_{i,t})$ do:
2.     let $m \in \mathcal{L}_{i,t} \leftarrow$ Choose($\mathcal{L}_{i,t}$) :: Choose a single element
3.     $\mathcal{L}_{i,t} \leftarrow \mathcal{L}_{i,t} \setminus \{m\}$
4. return $\mathcal{L}_{i,t}$
```

Appendix C

Results and Tables

C.1 Binary Positive Accuracy

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	0.244	0.019	1.000	0.652	1.000	1.000
LabelS	1.000	-	0.230	0.244	0.003	1.000	0.006	0.652	0.807
LabelP	1.000	0.230	-	0.652	0.041	0.428	0.169	1.000	0.323
LapSVM	0.244	0.244	0.652	-	1.000	1.000	0.471	0.244	0.079
SemiB	0.019	0.003	0.041	1.000	-	0.428	0.169	0.240	0.003
SelfT	1.000	1.000	0.528	1.000	0.428	-	0.138	1.000	0.112
CoT	0.652	0.006	0.169	0.471	0.169	0.138	-	1.000	0.001
TriT	1.000	0.652	1.000	0.244	0.240	1.000	1.000	-	0.003
SSGMM	1.000	0.807	0.323	0.079	0.003	0.112	0.001	0.003	-

Table C.1: Pairwise comparison p-values (25% Inductive binary positive accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	0.016	0.027	1.000	0.050	0.095	1.000
LabelS	1.000	-	1.000	0.037	0.015	1.000	0.004	0.319	0.367
LabelP	1.000	1.000	-	0.008	0.015	1.000	0.095	1.000	0.738
LapSVM	0.016	0.037	0.008	-	0.095	0.405	0.405	0.249	0.008
SemiB	0.027	0.015	0.015	0.095	-	0.540	1.000	1.000	0.012
SelfT	1.000	1.000	1.000	0.405	0.540	-	1.000	1.000	0.031
CoT	0.050	0.004	0.095	0.405	1.000	1.000	-	1.000	0.023
TriT	0.095	0.319	1.000	0.249	1.000	1.000	1.000	-	0.249
SSGMM	1.000	0.367	0.738	0.008	0.012	0.031	0.023	0.249	-

Table C.2: Pairwise comparison p-values (50% Inductive binary positive accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	0.006	1.000	1.000	0.272	0.808	0.982
LabelS	1.000	-	1.000	0.038	1.000	1.000	0.229	1.000	0.172
LabelP	1.000	1.000	-	0.006	1.000	1.000	0.808	1.000	0.491
LapSVM	0.006	0.038	0.006	-	0.014	0.505	0.006	0.006	0.003
SemiB	1.000	1.000	1.000	0.014	-	1.000	0.505	1.000	0.307
SelfT	1.000	1.000	1.000	0.505	1.000	-	0.691	1.000	0.017
CoT	0.272	0.229	0.808	0.006	0.505	0.691	-	0.626	0.001
TriT	0.808	1.000	1.000	0.006	1.000	1.000	0.626	-	0.010
SSGMM	0.982	0.172	0.491	0.003	0.307	0.017	0.001	0.010	-

Table C.3: Pairwise comparison p-values (75% Inductive binary positive accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	0.006	1.000	0.835	0.253	0.182	0.583
LabelS	1.000	-	1.000	0.034	1.000	1.000	0.253	1.000	0.126
LabelP	1.000	1.000	-	0.006	1.000	1.000	0.120	0.678	0.236
LapSVM	0.006	0.034	0.006	-	0.019	0.302	0.040	0.046	0.002
SemiB	1.000	1.000	1.000	0.019	-	1.000	0.336	1.000	0.253
SelfT	0.835	1.000	1.000	0.302	1.000	-	1.000	1.000	0.007
CoT	0.253	0.253	0.120	0.040	0.336	1.000	-	1.000	0.023
TriT	0.182	1.000	0.678	0.046	1.000	1.000	1.000	-	0.032
SSGMM	0.583	0.126	0.236	0.002	0.253	0.007	0.023	0.032	-

Table C.4: Pairwise comparison p-values (90% Inductive binary positive accuracy).

C.2 Binary Negative Accuracy

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.040
LabelS	1.000	-	1.000	1.000	1.000	1.000	1.000	1.000	0.009
LabelP	1.000	1.000	-	1.000	1.000	1.000	1.000	1.000	0.022
LapSVM	1.000	1.000	1.000	-	1.000	1.000	0.961	1.000	1.000
SemiB	1.000	1.000	1.000	1.000	-	1.000	0.804	0.804	0.601
SelfT	1.000	1.000	1.000	1.000	1.000	-	1.000	1.000	0.000
CoT	1.000	1.000	1.000	0.961	0.804	1.000	-	1.000	0.000
TriT	1.000	1.000	1.000	1.000	0.804	1.000	1.000	-	0.000
SSGMM	0.040	0.009	0.022	1.000	0.601	0.000	0.000	0.000	-

Table C.5: Pairwise comparison p-values (25% Inductive binary negative accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.103
LabelS	1.000	-	1.000	1.000	1.000	0.069	1.000	1.000	0.019
LabelP	1.000	1.000	-	1.000	1.000	0.479	1.000	1.000	0.103
LapSVM	1.000	1.000	1.000	-	1.000	1.000	1.000	1.000	1.000
SemiB	1.000	1.000	1.000	1.000	-	1.000	1.000	1.000	0.479
SelfT	1.000	0.069	0.479	1.000	1.000	-	0.179	0.091	0.001
CoT	1.000	1.000	1.000	1.000	1.000	0.179	-	1.000	0.237
TriT	1.000	1.000	1.000	1.000	1.000	0.091	1.000	-	0.238
SSGMM	0.103	0.019	0.103	1.000	0.479	0.001	0.237	0.238	-

Table C.6: Pairwise comparison p-values (50% Inductive binary negative accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.236
LabelS	1.000	-	1.000	1.000	0.904	0.096	1.000	1.000	0.113
LabelP	1.000	1.000	-	1.000	1.000	0.320	1.000	1.000	0.256
LapSVM	1.000	1.000	1.000	-	1.000	1.000	0.085	0.340	1.000
SemiB	1.000	0.904	1.000	1.000	-	1.000	1.000	1.000	0.033
SelfT	1.000	0.096	0.320	1.000	1.000	-	1.000	1.000	0.007
CoT	1.000	1.000	1.000	0.085	1.000	1.000	-	1.000	0.013
TriT	1.000	1.000	1.000	0.340	1.000	1.000	1.000	-	0.005
SSGMM	0.236	0.113	0.256	1.000	0.033	0.007	0.013	0.005	-

Table C.7: Pairwise comparison p-values (75% Inductive binary negative accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.196
LabelS	1.000	-	1.000	1.000	1.000	0.221	1.000	1.000	0.103
LabelP	1.000	1.000	-	1.000	1.000	0.253	1.000	1.000	0.298
LapSVM	1.000	1.000	1.000	-	1.000	1.000	1.000	1.000	1.000
SemiB	1.000	1.000	1.000	1.000	-	1.000	1.000	1.000	0.050
SelfT	1.000	0.221	0.253	1.000	1.000	-	0.103	0.103	0.004
CoT	1.000	1.000	1.000	1.000	1.000	0.103	-	1.000	1.000
TriT	1.000	1.000	1.000	1.000	1.000	0.103	1.000	-	0.540
SSGMM	0.196	0.103	0.298	1.000	0.050	0.004	1.000	0.540	-

Table C.8: Pairwise comparison p-values (90% Inductive binary negative accuracy).

C.3 Multi-class Global Accuracy

Ratio	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
0.25	3.00	3.35	4.69	2.75	4.36	2.86	3.00
0.50	3.82	3.82	4.06	3.00	4.21	2.86	2.53
0.75	3.27	4.00	4.25	2.94	4.07	2.86	2.47
0.90	3.09	4.24	4.06	2.75	4.07	3.00	2.40

Table C.9: Average rank of algorithms per ratio (Inductive multi-class global accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000
LabelS	1.000	-	0.535	1.000	1.000	1.000	1.000
LabelP	1.000	0.535	-	0.322	1.000	1.000	1.000
SelfT	1.000	1.000	0.322	-	1.000	1.000	1.000
CoT	1.000	1.000	1.000	1.000	-	0.275	1.000
TriT	1.000	1.000	1.000	1.000	0.275	-	1.000
SSGMM	1.000	1.000	1.000	1.000	1.000	1.000	-

Table C.10: Pairwise comparison p-values (25% Inductive multi-class global accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000
LabelS	1.000	-	1.000	1.000	1.000	1.000	1.000
LabelP	1.000	1.000	-	1.000	1.000	1.000	1.000
SelfT	1.000	1.000	1.000	-	1.000	1.000	1.000
CoT	1.000	1.000	1.000	1.000	-	1.000	1.000
TriT	1.000	1.000	1.000	1.000	1.000	-	1.000
SSGMM	1.000	1.000	1.000	1.000	1.000	1.000	-

Table C.11: Pairwise comparison p-values (50% Inductive multi-class global accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000
LabelS	1.000	-	1.000	1.000	1.000	1.000	0.733
LabelP	1.000	1.000	-	0.231	1.000	0.598	0.975
SelfT	1.000	1.000	0.231	-	1.000	1.000	1.000
CoT	1.000	1.000	1.000	1.000	-	1.000	1.000
TriT	1.000	1.000	0.598	1.000	1.000	-	1.000
SSGMM	1.000	0.733	0.975	1.000	1.000	1.000	-

Table C.12: Pairwise comparison p-values (75% Inductive multi-class global accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000
LabelS	1.000	-	1.000	0.097	1.000	1.000	0.844
LabelP	1.000	1.000	-	0.266	1.000	0.730	1.000
SelfT	1.000	0.097	0.266	-	1.000	1.000	1.000
CoT	1.000	1.000	1.000	1.000	-	1.000	1.000
TriT	1.000	1.000	0.730	1.000	1.000	-	1.000
SSGMM	1.000	0.844	1.000	1.000	1.000	1.000	-

Table C.13: Pairwise comparison p-values (90% Inductive multi-class global accuracy).

C.4 Multi-class Positive Accuracy

Ratio	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
0.25	2.27	2.88	3.56	2.69	3.50	2.29	2.07
0.50	2.55	2.76	3.44	2.62	3.57	2.57	2.20
0.75	2.45	2.88	3.56	2.75	3.50	2.57	1.60
0.90	2.73	2.88	3.38	2.81	3.36	2.50	1.40

Table C.14: Average rank of algorithms per ratio (Inductive multi-class positive accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000
LabelS	1.000	-	1.000	1.000	1.000	1.000	1.000
LabelP	1.000	1.000	-	1.000	1.000	1.000	0.995
SelfT	1.000	1.000	1.000	-	0.773	1.000	1.000
CoT	1.000	1.000	1.000	0.773	-	1.000	1.000
TriT	1.000	1.000	1.000	1.000	1.000	-	1.000
SSGMM	1.000	1.000	0.995	1.000	1.000	1.000	-

Table C.15: Pairwise comparison p-values (25% Inductive multi-class positive accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000
LabelS	1.000	-	1.000	1.000	1.000	1.000	1.000
LabelP	1.000	1.000	-	1.000	1.000	1.000	1.000
SelfT	1.000	1.000	1.000	-	1.000	1.000	1.000
CoT	1.000	1.000	1.000	1.000	-	1.000	1.000
TriT	1.000	1.000	1.000	1.000	1.000	-	1.000
SSGMM	1.000	1.000	1.000	1.000	1.000	1.000	-

Table C.16: Pairwise comparison p-values (50% Inductive multi-class positive accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	0.937
LabelS	1.000	-	1.000	1.000	1.000	1.000	0.268
LabelP	1.000	1.000	-	1.000	1.000	1.000	0.080
SelfT	1.000	1.000	1.000	-	1.000	1.000	0.786
CoT	1.000	1.000	1.000	1.000	-	1.000	1.000
TriT	1.000	1.000	1.000	1.000	1.000	-	1.000
SSGMM	0.937	0.268	0.080	0.786	1.000	1.000	-

Table C.17: Pairwise comparison p-values (75% Inductive multi-class positive accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	0.777
LabelS	1.000	-	1.000	1.000	1.000	1.000	0.155
LabelP	1.000	1.000	-	1.000	1.000	1.000	0.080
SelfT	1.000	1.000	1.000	-	1.000	1.000	0.298
CoT	1.000	1.000	1.000	1.000	-	1.000	0.524
TriT	1.000	1.000	1.000	1.000	1.000	-	0.328
SSGMM	0.777	0.155	0.080	0.298	0.524	0.328	-

Table C.18: Pairwise comparison p-values (90% Inductive multi-class positive accuracy).

C.5 Multi-class Negative Accuracy

Ratio	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
0.25	3.64	2.94	3.50	3.38	3.36	2.36	4.43
0.50	3.82	3.00	2.69	3.12	3.07	2.64	4.00
0.75	2.82	3.18	3.25	3.50	2.64	2.57	3.87
0.90	2.91	3.06	3.12	3.44	2.86	2.21	3.80

Table C.19: Average rank of algorithms per ratio (Inductive multi-class negative accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000
LabelS	1.000	-	1.000	1.000	1.000	1.000	1.000
LabelP	1.000	1.000	-	1.000	1.000	1.000	1.000
SelfT	1.000	1.000	1.000	-	1.000	1.000	1.000
CoT	1.000	1.000	1.000	1.000	-	1.000	0.334
TriT	1.000	1.000	1.000	1.000	1.000	-	1.000
SSGMM	1.000	1.000	1.000	1.000	0.334	1.000	-

Table C.20: Pairwise comparison p-values (25% Inductive multi-class negative accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000
LabelS	1.000	-	1.000	1.000	1.000	1.000	1.000
LabelP	1.000	1.000	-	1.000	1.000	1.000	1.000
SelfT	1.000	1.000	1.000	-	1.000	1.000	1.000
CoT	1.000	1.000	1.000	1.000	-	1.000	1.000
TriT	1.000	1.000	1.000	1.000	1.000	-	1.000
SSGMM	1.000	1.000	1.000	1.000	1.000	1.000	-

Table C.21: Pairwise comparison p-values (50% Inductive multi-class negative accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000
LabelS	1.000	-	1.000	1.000	1.000	1.000	1.000
LabelP	1.000	1.000	-	1.000	1.000	1.000	1.000
SelfT	1.000	1.000	1.000	-	1.000	1.000	1.000
CoT	1.000	1.000	1.000	1.000	-	1.000	1.000
TriT	1.000	1.000	1.000	1.000	1.000	-	1.000
SSGMM	1.000	1.000	1.000	1.000	1.000	1.000	-

Table C.22: Pairwise comparison p-values (75% Inductive multi-class negative accuracy).

-	ASB	LabelS	LabelP	SelfT	CoT	TriT	SSGMM
ASB	-	1.000	1.000	1.000	1.000	1.000	1.000
LabelS	1.000	-	1.000	1.000	1.000	1.000	1.000
LabelP	1.000	1.000	-	1.000	1.000	0.825	1.000
SelfT	1.000	1.000	1.000	-	1.000	1.000	1.000
CoT	1.000	1.000	1.000	1.000	-	1.000	1.000
TriT	1.000	1.000	0.825	1.000	1.000	-	1.000
SSGMM	1.000	1.000	1.000	1.000	1.000	1.000	-

Table C.23: Pairwise comparison p-values (90% Inductive multi-class negative accuracy).

C.6 Transductive Binary Positive Accuracy

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	1.000	0.953	0.001	0.000	1.000	1.000	0.047
LabelS	1.000	-	0.001	0.000	0.000	1.000	1.000	0.439
LabelP	0.953	0.001	-	0.001	0.000	0.741	0.465	0.012
LapSVM	0.001	0.000	0.001	-	1.000	0.000	0.000	0.000
SemiB	0.000	0.000	0.000	1.000	-	0.000	0.000	0.000
SelfT	1.000	1.000	0.741	0.000	0.000	-	1.000	0.074
SSGMM	1.000	1.000	0.465	0.00	0.000	1.000	-	1.000
TSVM	0.047	0.439	0.012	0.000	0.000	0.074	1.000	-

Table C.24: Pairwise comparison p-values (25% Transductive binary positive accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	1.000	1.000	0.000	0.000	1.000	0.600	0.001
LabelS	1.000	-	0.007	0.000	0.000	1.000	0.366	0.030
LabelP	1.000	0.007	-	0.000	0.000	1.000	0.218	0.003
LapSVM	0.000	0.000	0.000	-	0.127	0.000	0.000	0.000
SemiB	0.000	0.000	0.000	0.127	-	0.000	0.000	0.000
SelfT	1.000	1.000	1.000	0.000	0.000	-	0.366	0.001
SSGMM	0.600	0.366	0.218	0.000	0.000	0.366	-	1.000
TSVM	0.001	0.030	0.003	0.000	0.000	0.001	1.000	-

Table C.25: Pairwise comparison p-values (50% Transductive binary positive accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	0.832	0.658	0.001	0.025	0.439	0.658	0.040
LabelS	0.832	-	0.832	0.000	0.439	1.000	0.227	0.001
LabelP	0.658	0.832	-	0.000	0.815	1.000	0.227	0.001
LapSVM	0.001	0.000	0.000	-	0.000	0.000	0.000	0.000
SemiB	0.025	0.439	0.815	0.000	-	0.021	0.131	0.001
SelfT	0.439	1.000	1.000	0.000	0.021	-	0.439	0.001
SSGMM	0.658	0.227	0.227	0.000	0.131	0.439	-	1.000
TSVM	0.040	0.001	0.001	0.000	0.001	0.001	1.000	-

Table C.26: Pairwise comparison p-values (75% Transductive binary positive accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	1.000	1.000	0.001	0.307	0.777	1.000	0.180
LabelS	1.000	-	1.000	0.001	0.923	1.000	0.923	0.307
LabelP	1.000	1.000	-	0.001	1.000	1.000	0.870	0.109
LapSVM	0.001	0.001	0.001	-	0.002	0.002	0.001	0.001
SemiB	0.307	0.923	1.000	0.002	-	0.270	1.000	0.039
SelfT	0.777	1.000	1.000	0.002	0.270	-	0.964	0.072
SSGMM	1.000	0.923	0.870	0.001	1.000	0.964	-	1.000
TSVM	0.180	0.307	0.109	0.001	0.039	0.072	1.000	-

Table C.27: Pairwise comparison p-values (90% Transductive binary positive accuracy).

C.7 Transductive Binary Negative Accuracy

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	0.077	1.000	0.007	0.001	1.000	0.000	0.142
LabelS	0.077	-	0.027	0.001	0.000	0.007	0.000	1.000
LabelP	1.000	0.027	-	0.007	0.001	1.000	0.000	0.584
LapSVM	0.007	0.001	0.007	-	1.000	0.004	0.001	0.001
SemiB	0.001	0.000	0.001	1.000	-	0.000	0.000	0.000
SelfT	1.000	0.007	1.000	0.004	0.000	-	0.000	0.323
SSGMM	0.000	0.000	0.000	0.001	0.000	0.000	-	0.002
TSVM	0.142	1.000	0.584	0.001	0.000	0.323	0.002	-

Table C.28: Pairwise comparison p-values (25% Transductive binary negative accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	0.068	0.468	0.004	0.000	0.538	0.001	0.155
LabelS	0.068	-	0.210	0.002	0.001	0.005	0.000	0.538
LabelP	0.468	0.210	-	0.004	0.002	0.193	0.001	0.468
LapSVM	0.004	0.002	0.004	-	0.468	0.005	0.001	0.002
SemiB	0.000	0.001	0.002	0.468	-	0.000	0.000	0.002
SelfT	0.538	0.005	0.193	0.005	0.000	-	0.000	0.210
SSGMM	0.001	0.000	0.001	0.001	0.000	0.000	-	0.003
TSVM	0.155	0.538	0.468	0.002	0.002	0.210	0.003	-

Table C.29: Pairwise comparison p-values (50% Transductive binary negative accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	0.271	0.289	0.006	0.255	0.506	0.001	0.239
LabelS	0.271	-	0.255	0.004	0.051	0.068	0.001	0.306
LabelP	0.289	0.255	-	0.009	0.079	0.180	0.001	0.239
LapSVM	0.006	0.004	0.009	-	0.014	0.011	0.001	0.001
SemiB	0.255	0.051	0.079	0.014	-	0.062	0.000	0.066
SelfT	0.506	0.068	0.180	0.011	0.062	-	0.000	0.079
SSGMM	0.001	0.001	0.001	0.001	0.000	0.000	-	0.014
TSVM	0.239	0.306	0.239	0.001	0.066	0.079	0.014	-

Table C.30: Pairwise comparison p-values (75% Transductive binary negative accuracy).

-	ASB	LabelS	LabelP	LapSVM	SemiB	SelfT	SSGMM	TSVM
ASB	-	0.435	0.449	0.017	1.000	1.000	0.001	0.449
LabelS	0.435	-	1.000	0.015	0.115	0.115	0.000	0.419
LabelP	0.449	1.000	-	0.013	0.111	0.115	0.001	0.631
LapSVM	0.017	0.015	0.013	-	0.081	0.094	0.002	0.013
SemiB	1.000	0.115	0.111	0.081	-	1.000	0.001	0.419
SelfT	1.000	0.115	0.115	0.094	1.000	-	0.000	0.289
SSGMM	0.001	0.000	0.001	0.002	0.001	0.000	-	0.013
TSVM	0.449	0.419	0.631	0.013	0.419	0.289	0.013	-

Table C.31: Pairwise comparison p-values (90% Transductive binary negative accuracy).