SemiSupervised: Scalable Semi-Supervised Routines for Real Data Problems

Mark Vere Culp
West Virginia University

Kenneth Joseph Ryan
West Virginia University

May 9, 2018

Abstract

Machine learning approaches optimize predictive performance by achieving a desirable compromise between bias and variance. Many supervised software packages produce a strong performing universal prediction rule independent of the testing data. In practice, one often wants to achieve the best possible prediction performance on a specific testing data set. For example, suppose the feature information in the testing data are a set of extrapolations away from the mean of the training data. In this case, more aggressive shrinking may be beneficial. One could view this contrast in terms of a sports metaphor. It is often better to prepare for your specific opponent (semi-supervised) than to prepare for a generic opponent (supervised). We present the R package **SemiSupervised**. This package efficiently implements a family of semi-supervised techniques and can directly incorporate the feature data corresponding to predictions of interest in training. The efficacy of this package on real arbitrary data follows from its use of recent advances including *safe* enhancements, anchor graph approximations, linear in n training, optimized tuning via cross-validation, and other heuristics.

Note: Code to generate all tables is available in the document_code subdirectory for this package.

Keywords:*machine learning, fast parameter estimation, R*

1 Introduction

Semi-supervised learning addresses the topic of incorporating unlabeled (testing) data into a learning prediction framework [3]. Precisely, semi-supervised approaches differ from supervised in that they use information in both labeled (response is observed) and unlabeled (response is not observed) data to form better predictions than using the labeled cases only for training (i.e., supervised learning) [13]. The literature is replete with semi-supervised learning techniques including greedy graph cut approaches [31], logistic tree based approaches [7], manifold regularization [2], co-training [33], harmonic function approaches [6], and covariate shift methods [11, 26]. The focus has been on understanding circumstances where these approaches are expected to work [14, 6] or expected to fail [29, 18].

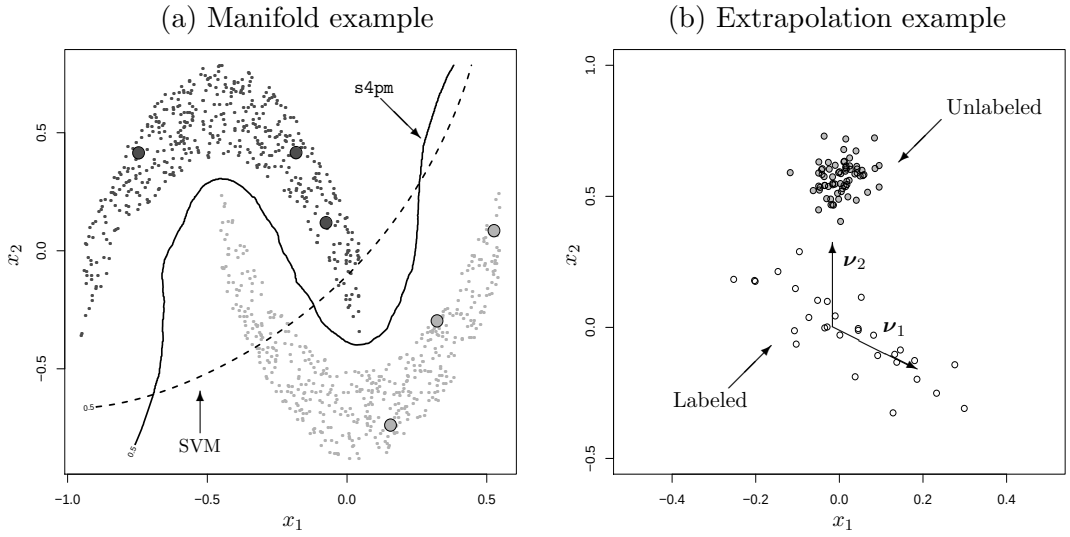


Figure 1: (a) A $p = 2$ classification example. The 6 labeled observations are plotted as big gray ($y = 0$) versus black ($y = 1$) dots. The additional 200 unlabeled observations uncover 2 crescent-shaped manifolds, which are only accounted for by the classification border. (b) The feature information for a $p = 2$ regression example. While the continuous response is not displayed, white points are "labeled" and gray points are "unlabeled." The first principal component of the labeled data is close to $\boldsymbol{\nu}_1$, and $\boldsymbol{\nu}_2$ is in the direction of the unlabeled mean.

Much of this existing work is hindered by the following issues. Training often involves inverting large matrices with dimensions equal to the full data size, so techniques don't scale to big data problems. Tuning parameters are rarely estimated in an efficient way and often require direct user specification. Many techniques are for classification only and do not have a natural extension to regression. Techniques were typically defined or tuned to perform well on some ad hoc synthetic data sets, but performance tends to be much worse than even basic supervised techniques in noisy/realistic prediction problems. We speculate that these shortcomings tend to impede the practical use of semi-supervised learning, even though it is not too hard to motivate the practical advantage of using unlabeled data in training.

To address this, the proposed **SemiSupervised** package features a *safe* semi-supervised semi-parametric model (`s4pm`) and a fast anchor graph approximation to the `s4pm` referred to as `agraph`. These methods offer the following important characteristics: (i) safeguards to make the semi-supervised predictions competitive with (if not better than) state-of-the-art machine learning approaches on real data sets, (ii) fitting times on both small and large data sets that do not prevent widespread use and that scale as a linear function of the full data size, and (iii) internal on-the-fly estimation of all tuning parameters. The approach builds off of an established semi-supervised joint optimization criterion and generalizes several popular semi-supervised graph-based techniques. Our overall goal in this effort is to present a software package that implements high performing semi-supervised techniques that tend to perform as good or better than their supervised counterparts in linear time.

The principles behind using unlabeled data in training relates to the bias and variance trade-

off of the prediction error rate [26]. A judicious use of the unlabeled data can boost performance by reducing bias (variance) if there is a modest increase in variance (bias), but the component of the trade-off to be reduced depends on the structure of the full data as illustrated by Figure 1. In circumstances such as Figure 1(a), a high performing semi-supervised technique incorporates unlabeled data to detect manifolds within the data to exploit the trade-off by dramatically decreasing prediction bias while increasing variance [6]. The feature data are represented by two moon manifolds, and the semi-supervised fit passes between them. The supervised fit is only able to train from the 6 labeled cases and misses the pattern.

In contrast, consider Figure 1(b). It is now more desirable for a semi-supervised technique to shrink along unlabeled data extrapolation directions in an effort to dramatically decrease prediction variance at the expense of an increase in prediction bias [26]. The goal would be to aggressively shrink in direction $\boldsymbol{\nu}_2$ while emphasizing prediction strength in direction $\boldsymbol{\nu}_1$. For example, suppose $\widehat{\boldsymbol{\beta}}$ is a linear semi-supervised estimator and that $\boldsymbol{x}_0$ is an unlabeled prediction vector. To decrease prediction variance, the semi-supervised prediction would ideally satisfy $\boldsymbol{x}_0^\top \widehat{\boldsymbol{\beta}} \approx a_1 \boldsymbol{\nu}_1$. On the other hand, the performance of a supervised technique may suffer as a need to shrink aggressively along direction $\boldsymbol{\nu}_2$ would be missed because such techniques by definition do not use the unlabeled data in training.

The semi-supervised concepts from Figure 1 were each developed to be *safe* in their application. That is, `s4pm` and `agraph` use the unlabeled data based on manifold in Figure 1(a) and/or extrapolation in Figure 1(b) requirements, and these safe techniques can also adjust back towards a supervised prediction rule to handle real noisy data problems when semi-supervised learning does not provide an advantage. The end-goal in-terms of performance is for the technique to work as good or better than its supervised counterpart on real data. This is not guaranteed to always happen (in a strict sense on a particular data set), but empirical results and theoretical work provide evidence in favor of this *safe* goal [11, 19]. Nearly, all existing semi-supervised approaches are not safe and provide no such considerations to our knowledge. Indeed, it is well-known that many semi-supervised techniques break-down in practice often performing much worse than even the simplest supervised approaches. In terms of implementation, the combination of incorporating unlabeled data in training and employing safety enhancements does indeed come at cost compared to supervised learning, but this package was carefully written to substantially decrease this cost without sacrificing performance.

The entire fitting process of each technique was optimized using LAPACK/BLAS routines in a customized manner, and all tuning parameters are optimized with custom-designed k-fold cross-validation code. In cross-validation (CV), our goal is to determine the best tuning parameter choices for prediction of unlabeled cases. In practice, many optimal tuning parameter combinations tend to be on the boundary between a stable and unstable solution. This is difficult from a computational perspective, since many grid searches will encounter unstable values which force the underlying LAPACK routines to cycle through large matrices just to produce a terrible error rate. In other words, the vast majority of CV time is spent rejecting bad tuning parameter options as opposed to determining good ones. The LAPACK routines were carefully used to optimize error processing so as to reject bad choices as fast as possible. In this effort, special care was given to not only improve CV estimation but to also estimate all important internal parameters in real time. The **SemiSupervised** package integrates these optimized routines in R with a user-friendly front-end S4 generic structure.

1.1 Semi-supervised software

There are some existing semi-supervised software packages in R. The **Rmixmod** R package provides supervised, semi-supervised, and unsupervised clustering for classification problems [15]. The **upclass** R package implements an EM style semi-supervised routine building off of model-based clustering [25]. The **HCsnip** R package provides a semi-supervised hierarchical clustering method [23]. The **spa** R package is the most similar to the proposed approach and has been redesigned to work better with the enhancements developed for the **SemiSupervised** package [5].

Most semi-supervised routines are implemented as MATLAB scripts with fixed tuning settings to reproduce the technique being proposed as part of a publication. The closest to our work is the **SemiL** package which implements diffusion kernels and Gaussian harmonic fields primary in classification [9]. Kernel-based classification software include **SGTlight**, **SVMlight**, and **SVMlin** [10, 28]. The vast majority of other semi-supervised packages that we are aware of are primarily designed for specific text/web related applications, not large scale regression or classification applications.

The proposed software distinguishes itself from these existing packages in some key ways. The main difference is that this package is designed for arbitrary regression or classification problems. As a result, the package is designed to perform all necessary CV for tuning parameter estimates in real time. In addition, the novel semi-parametric *safe* term provides additional robustness to real world data problems. Final contributions include anchor graph approximations, linear in n training, an S4-generic front end, and some heuristics fined tuned for big data problems.

2 Package metrics and highlights

Semi-supervised regression or classification on big or small data is made easy with the **SemiSupervised** package. This section highlights the performance of the `s4pm` and `agraph` functions on a wide range of real data sets. The benchmarks are listed in Table 1. An internal CV approach was used to estimate all tuning parameters when calling `s4pm` or `agraph` at default settings throughout this section, so run-times reported include CV. The inner workings of the `s4pm` and `agraph` are deferred to Section 3.

Since `agraph` is an approximation to the `s4pm`, a comparison of these should bring into focus a familiar performance versus speed trade-off: (a) get the best performance by optimizing a computationally intense problem (e.g., the `s4pm`) versus (b) get (hopefully) comparable performance results faster by optimizing a problem requiring substantially less computation (e.g., the `agraph`). An additional comparison is also necessary to establish the *safe* component of this software. The natural supervised comparison to this end is the efficient `glmnet` [8]. The `glmnet` has 2 Lagrangian tuning parameters (λ, α) that were both optimized using CV for this comparison.

Let the index set $\{1, 2, \ldots, n\}$ of all n observations be partitioned into the labeled L and unlabeled U sets. For now, assume that an $n \times p$ model matrix $\boldsymbol{X}$ and a vector $\boldsymbol{Y}_L$ of $m = |L|$ responses are available in training, whereas the vector $\boldsymbol{Y}_U$ of $n - m = |U|$ responses is missing at the time of training and forecasting.

To simulate this commonly occurring scenario during benchmarking, the data were randomly

	Data	n	p	Response	Reference
1.	Ethanol	589	1,037	Ethanol	[26]
2.*	Breast	699	9	Cancer	[20]
3.*	Image	2,310	18	Picture	[20]
4.*	Sol	5,631	72	Solubility	[26]
5.	Power	9,568	4	Usage	[20]
6.	Navy	11,933	16	GT Decay	[20]
7.	House	20,640	13	ln(Value)	[24]
8.	CASP	45,730	9	Residue Size	[20]
9.	Song	515,345	90	Release Year	[20]
10.	NYC	1,073,473	44	Property Tax	[34]

Table 1: Data sets used for benchmarking. Asterisks indicate classification problems with a binary response. The other problems are regression with a continuous response.

partitioned into labeled and unlabeled sets such that $m = \lceil 0.15 * n \rceil \mathcal{I}_{\{n \leq 1000\}} + 200 \mathcal{I}_{\{n > 1000\}}$, and the unlabeled responses were withheld from training. This experiment was repeated 100 times for each data set. To assist with comparisons in the regression problems, a scaled unlabeled root mean squared error metric

$$\text{sRMSE}_U = \sqrt{\frac{m}{n-m} \frac{\sum_{i \in U} (\boldsymbol{Y}_i - \boldsymbol{f}_i)^2}{\sum_{i \in L} (\boldsymbol{Y}_i - \bar{\boldsymbol{Y}}_L)^2}}$$

was used, where $\bar{\boldsymbol{Y}}_L$ is the arithmetic sample mean of the labeled responses $\boldsymbol{Y}_L$ and the length n vector $\boldsymbol{f}$ is comprised of the predictions from some technique (supervised or semi-supervised). In classification, the performance metric reported is the unlabeled classification error. The results for data sets 1-10 from Table 1 are listed in Table 2. Interestingly, the results came out better than expected for the `agraph` with 7 out of 10 wins in comparison to the `s4pm`. However, it is important not to read too much into this with real data (as opposed to simulated data from a probabilistic model in a more tightly controlled simulation study). In any case, the performance of either semi-supervised approach was quite strong overall in terms of comparisons with the `glmnet`, justifying the *safe* semi-supervised component of the software.

In terms of CPU time, the original techniques that this code is based off of involved cubic in n inverses which would not be realistic for larger data sets. Much of our work in this software development was to reduce the additional computational burden of training with the unlabeled data; see Section 4.5. The log - log plot of the Average Times are provided against n in Figure 2 using the data set number from Table 1 as the plotting symbol. For either the `s4pm` or the `agraph`, the least-squares line of the regression examples is roughly $\log(\text{Time}) = 0.6 \log(n) - 3$, which indicates a slightly better than linear fit. It is noteworthy that the supervised `glmnet` was faster than the proposed approach especially on the larger data sets. This makes senses, since the supervised technique trains $\widehat{\boldsymbol{\beta}}$ using only the labeled data and then predicts the unlabeled case as $\boldsymbol{x}_0^\top \widehat{\boldsymbol{\beta}}$. In other words, it is always faster to ignore the unlabeled data than to use it in

Data	n	p	s4pm	agraph	glmnet
Ethanol	589	1,037	0.563 ± 0.032	0.573 ± 0.030	**0.445** ± 0.034
Breast	699	9	0.054 ± 0.002	**0.033** ± 0.001	0.045 ± 0.002
Image	2,310	18	0.356 ± 0.020	**0.077** ± 0.005	0.102 ± 0.003
Sol	5,631	72	0.378 ± 0.002	**0.301** ± 0.004	0.306 ± 0.004
Power	9,568	4	0.280 ± 0.002	**0.277** ± 0.002	0.286 ± 0.004
Navy	11,933	16	0.731 ± 0.069	**0.118** ± 0.033	0.456 ± 0.005
House	20,640	8	**0.627** ± 0.008	0.641 ± 0.009	0.656 ± 0.011
CASP	45,730	9	**0.866** ± 0.007	0.873 ± 0.007	0.895 ± 0.008
Song	515,345	90	1.002 ± 0.021	**0.993** ± 0.020	1.009 ± 0.021
NYC	1,073,473	44	7.465 ± 1.263	**6.926** ± 1.038	10.297 ± 1.519

Table 2: Performance results for the Table 1 benchmarks, i.e., average performance metric ± 2 standard errors.

	Mac OS X		Linux Ubuntu		Windows	
	Ref.	VecLib	Ref.	MKL	Ref.	MKL
agraph	34.44	15.39	36.03	9.11	54.70	15.64
s4pm	47.18	22.91	25.34	13.79	33.81	23.64

Table 3: CPU time in seconds to execute the CASP data set on different OS's using different BLAS libraries on each system. The median out of ten runs is reported.

training. The use of semi-supervised learning is justified by potential performance improvements from incorporating this information in training.

The proposed software uses underlying LAPACK routines which are optimized in different BLAS libraries. Multi-threaded libraries can be used to substantially improve the empirical speed of all matrix operations in practice. To demonstrate that the proposed software can take advantage of these enhancements, additional results are listed in Table 3. These results demonstrate the substantial speed improvements due to using the freely available VecLib with 6 cores or the Microsoft Kernel Library MKL with 8-cores against the default Reference (Ref.) libraries in R. As expected, the `agraph` has a greater impact over the `s4pm`. Each was fit on a top-of-the-line machine available in the Department of Statistics at West Virginia University.

A final benchmark metric for consideration is the effect as $m = |L|$ increases. Results in Table 4 demonstrate that the `agraph` approach is linear in m as well, while the `s4pm` is cubic in m. Performance is also provided as a function of labeled size indicating that the faster `agraph` does indeed perform slightly worse than the `s4pm`.

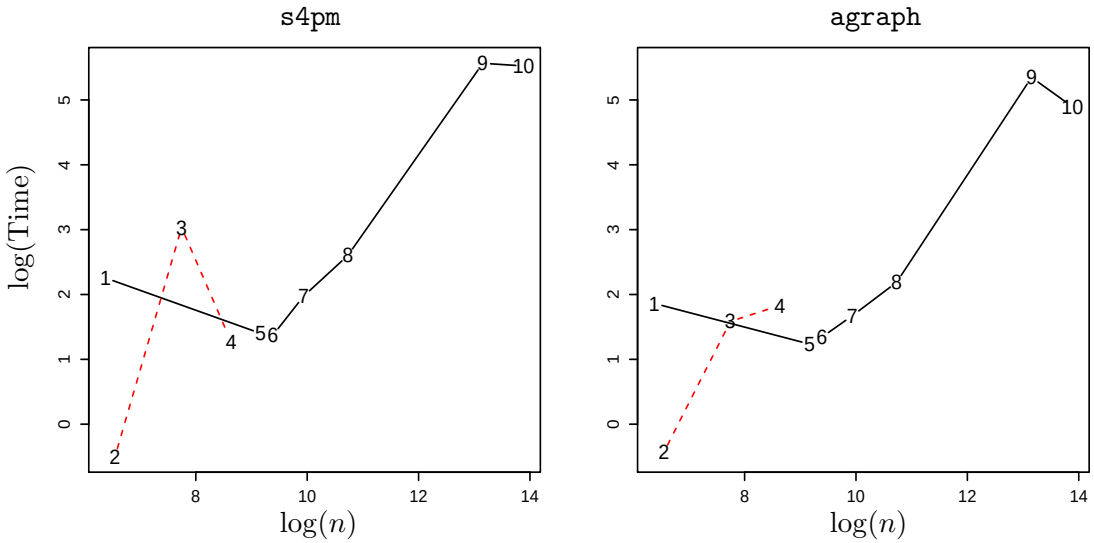


Figure 2: Plots of computation time in seconds versus n on log - log scales. The solid-black and dashed-red lines connect the outcomes for the regression and classification examples.

	m:	200	500	1,000	2,000	5,000	10,000
Time (sec.):	`agraph`	9.12	9.50	11.34	15.99	24.33	44.04
	`s4pm`	13.45	18.51	29.92	73.34	431.0	1,829
Performance:	`agraph`	0.871	0.834	0.816	0.804	0.787	0.776
	`s4pm`	0.860	0.823	0.793	0.773	0.731	0.700

Table 4: Labeled size m in terms of CPU time in seconds and performance is listed for the Casp data. The median out of ten runs is reported.

3 The SemiSupervised package

In this Section, the focus is on the general structure of **SemiSupervised**. The **SemiSupervised** R package implements three different semi-supervised routines: `s4pm`, `agraph`, and `jtharm`. In general, these functions are different variations of the semi-supervised joint training optimization problem. To set this up, let L and U again partition the index set $\{1, \ldots, n\}$ for the n observations into the sets of labeled and unlabeled observations. The technical setup requires that the $m = |L|$ labeled observations $(y_i, \boldsymbol{x}_i)$ for $i \in L$ are independent and identically distributed. An additional $n - m = |U|$ unlabeled observations $\boldsymbol{x}_i$ are also independent and identically distributed (and independent of the labeled data), but their responses y_i for $i \in U$ are not available for training. The full data are represented by an $n \times p$ matrix $\boldsymbol{X}$ with row-wise partitions $\boldsymbol{X}_L$ and $\boldsymbol{X}_U$. It is important to note that the length n response vector $\boldsymbol{Y}$ partitions into the m

observed responses $\boldsymbol{Y}_L$ and $n-m$ *latent* (or unobserved) variables $\boldsymbol{Y}_U$. Joint training

$$\min_{\boldsymbol{\beta},\boldsymbol{f},\boldsymbol{Y}_U} L\left(\boldsymbol{Y}\left(\boldsymbol{Y}_U\right), \boldsymbol{f}+\boldsymbol{X}\boldsymbol{\beta}\right) + \lambda_1 \boldsymbol{f}^\top \boldsymbol{B}\boldsymbol{f} + \lambda_2 \parallel \boldsymbol{\beta} \parallel_2^2 + \gamma \left\|\boldsymbol{Y}_U\right\|_2^2 \tag{1}$$

provides a general framework for arriving at suitable semi-supervised predictions given a loss function $L(\cdot,\cdot)$, a tuning parameter vector $(\lambda_1, \lambda_2, \gamma) \geq 0$, a stacked response vector $\boldsymbol{Y}(\boldsymbol{Y}_U)$ with $\boldsymbol{Y}_L$ on top of $\boldsymbol{Y}_U$, and a positive semi-definite penalty matrix $\boldsymbol{B}$.

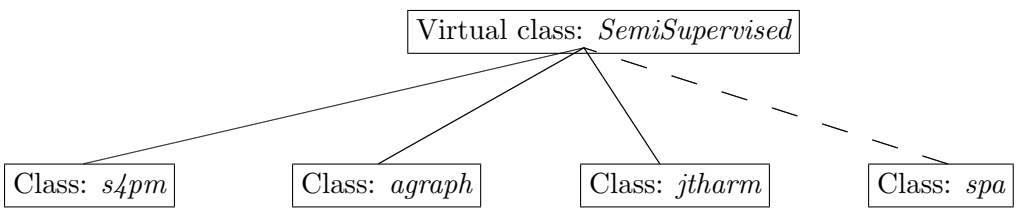
Briefly, the `s4pm` implements the solution $\widehat{\boldsymbol{\eta}} = \widehat{\boldsymbol{f}} + \boldsymbol{X}\widehat{\boldsymbol{\beta}}$ to this optimization problem for 2 choices of $\boldsymbol{B}$: (i) the Laplacian matrix and (ii) the normalized Laplacian matrix. The `agraph` approximates $\widehat{\boldsymbol{\eta}} \approx \boldsymbol{Z}\widehat{\gamma} + \boldsymbol{X}\widehat{\boldsymbol{\beta}}$ using a matrix $\boldsymbol{Z}$ that is constructed via an anchor graph penalty matrix. The `jtharm` optimizes (1) using a weighted loss and has no $\boldsymbol{X}\boldsymbol{\beta}$ term (i.e., $\lambda_2 = \infty$). The details for these techniques are discussed in Sections 4 and 5 with examples.

The $\boldsymbol{X}\widehat{\boldsymbol{\beta}}$ term in (1) provides the *safe* enhancement to the semi-supervised solution. In prior work, the connection between optimization (1) and several semi-supervised techniques proposed in the literature including regularized Laplacians [1], manifold regularization [2], label spreading [36], regularized anchor methods [21, 22], transductive SVM [3], S3VM [4], and ψ-learning [32] was established [6]. Most of these techniques are simplified versions of (1) usually with $\gamma = 0$ and $\lambda_2 = \infty$, i.e., $\widehat{\boldsymbol{Y}}_U = \widehat{\boldsymbol{f}}_U$ and $\widehat{\boldsymbol{\beta}} = \vec{0}$. Examples on how to fit these special cases using the `s4pm` are provided in Section 4.1.

3.1 S4 structure

The **SemiSupervised** utilizes R's object model S4 class structure which builds off the **methods** package. The virtual class *SemiSupervised* defines the base signature for a semi-supervised object in R. Every object in this class has the usual `dim`, `fit`, and `fitted` methods. In addition, new base methods for class *SemiSupervised* includes `type` (whether regression or classification), `xmatrix` (the scaled feature data used for training), `gmatrix` (graphical representation), `ymatrix` (the scaled response), and `measures` (the error information). All accessor and assignments in slots are accessed either through the access function or directly through the slot with the `@` operator.

The diagram below represents instances of this virtual class which are different versions of optimization (1). The similar `spa` function in the **spa** package was also re-written as an instance of virtual class *SemiSupervised* (refer to Section 5 for more on this connection).



Each class has a `standardGeneric` by the same name. The input methods for each class include `formula`, `NULL`, `matrix`, `vector`, and `data.frame`. The `agraph` also allows for an `anchor` graph object to be inputted. All input methods call a default method (i.e., `s4pm.default`) which in-turn invokes low-level fit (i.e., `s4pm.fit`) and CV (i.e., `s4pm.cv`) routines. These low-level

routines are not exported and require the `:::` scope operator for access in R. In addition, each routine offers a `show` method, and a `predict` method. This provides the standard base structures necessary to fit an instance of *SemiSupervised*.

4 Demonstration of semi-supervised estimators

A demonstration of the safe semi-supervised parametric model and the anchor graph approximation are given in this section. Each technique optimizes special instances of (1). This demonstration involves a data set from the **caret** package [12], so it is necessary to first install both packages with the R commands `install.packages("SemiSupervised")` and `install.packages("caret")`. After loading the **SemiSupervised** package in an R session with the command `library("SemiSupervised")`, all the R code presented in this section is conveniently executed with the command `demo("SemiSupervised")`. This demonstration begins by loading and preprocessing the Blood Brain data [12] for semi-supervised learning with the **SemiSupervised** package. The performance metric sRMSE_U for this type of regression problem with a continuous response is also defined.

```
R> library("SemiSupervised")
R> library("caret")

Loading required package: lattice
Loading required package: ggplot2

R> data("BloodBrain")
R> set.seed(100)
R> n <- nrow(bbbDescr)
R> L <- sample(1:n, ceiling(0.1 * n))
R> U <- setdiff(1:n, L)
R> y.fit <- rep(NA, n)
R> y.fit[L] <- logBBB[L]
R> msy <- sqrt(mean((logBBB[L] - mean(logBBB[L]))^2))
R> sRMSEU <- function(f, y, msy = 1.0){ sqrt(mean((f - y)^2)) / msy }
```

By default, the **SemiSupervised** package fits the *safe* semi-supervised technique (1) with the all-purpose `formula` call $y \sim .$, e.g., the details for constructing $\boldsymbol{B}$ from $\boldsymbol{X}$ are hidden. Adjustments to these settings are demonstrated in Sections 4.1 and 4.2, and some other possibilities are given in Sections 4.4 and 5. Also, the R help pages complement the Blood Brain regression example of Sections 3-4.2 with a classification example involving a Sonar data set [16]. Such classification examples optimize (1) with a hybrid general loss function, $L\left(\boldsymbol{Y}\left(\boldsymbol{Y}_U\right), \boldsymbol{f} + \boldsymbol{X}\boldsymbol{\beta}\right)$, as the logistic loss function for the labeled cases and squared error for the unlabeled observations.

4.1 `s4pm`: Regression with proximity graphs

The details for how to choose the penalty matrix $\boldsymbol{B}$ for the `s4pm` are discussed next. A distance between observations or rows of $\boldsymbol{X}$ must be specified to begin constructing this penalty

function. The **SemiSupervised** package implements both Cosine Distance (as default) and Euclidean Distance. The $n \times n$ distance matrix is a fully connected graph. To induce edge sparsity, a k-NN graph is fit, i.e., the number of closest points is specified. The default is $k = 6$. The package converts dissimilarity to similarities using the local kernel $\boldsymbol{W}_{ij} = \exp(-\boldsymbol{D}_{ij}/h)$ where $\boldsymbol{D}_{ij}$ is a dissimilarity measure between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Let $\text{diag}\left(\boldsymbol{W}\vec{1}\right)$ denote the diagonal row sum matrix of $\boldsymbol{W}$. The penalty matrix $\boldsymbol{B}$ in optimization (1) is either the combinatorial Laplacian $\boldsymbol{B} = \text{diag}\left(\boldsymbol{W}\vec{1}\right) - \boldsymbol{W}$ or the normalized Laplacian $\boldsymbol{B} = \text{diag}\left(\boldsymbol{W}\vec{1}\right)^{-1/2}\left(\text{diag}\left(\boldsymbol{W}\vec{1}\right) - \boldsymbol{W}\right)\text{diag}\left(\boldsymbol{W}\vec{1}\right)^{-1/2}$. The package defaults to the normalized Laplacian.

The default call to the `s4pm` solves optimization (1) directly with the normalized Laplacian used for $\boldsymbol{B}$ as shown below.

```
R> safe.spread <- s4pm(y ~ ., data = data.frame(y = y.fit, bbbDescr))
R> safe.spread

S4PM Fit with (n,|L|)=( 208 , 21 ) or  10 % labeled

Performance Estimates:
k-CV:  0.636  GCV:  0.036  DF:  12.572

Fit Estimates:
Graph Kernel h:  0.237  Lagrangians:  10   0.001  Safe-Lagrangian 10

R> sRMSEU(fitted(safe.spread)[U], logBBB[U], msy)
[1] 1.111641
```

The 3-fold CV estimates of $(h, \lambda_1, \lambda_2, \gamma) = (0.237, 10.0, 10.0, 0.001)$ and their optimal CV value of 0.636 are displayed. The Generalized Cross-Validation (GCV) and degrees of freedom performance estimates corresponding to this fit are printed out as well.

The function `SemiSupervised.control` gives the user the flexibility to switch to the combinatorial Laplacian matrix.

```
R> ctrl<-SemiSupervised.control(normalize = FALSE)
R> safe.lap<-s4pm(y ~ ., data = data.frame(y = y.fit, bbbDescr),
R+                control = ctrl)
R> safe.lap

S4PM Fit with (n,|L|)=( 208 , 21 ) or  10 % labeled

Performance Estimates:
k-CV:  0.639  GCV:  0.036  DF:  14.102

Fit Estimates:
Graph Kernel h:  0.48  Lagrangians:  10   0.001  Safe-Lagrangian 10
```

```
R> sRMSEU(fitted(safe.lap)[U], logBBB[U], msy)

[1] 1.10838
```

Notice that the scaled unlabeled RMSE improves slightly with the combinatorial Laplacian in this particular example. In practice, allowing a large enough grid and scaling the data seems to trivialize the difference between the two graph operators.

Next, 2 additional flexibilities of the function `s4pm` are demonstrated. (i) Its inputs can be adjusted to directly fit some popular semi-supervised special cases of (1). (ii) It can be used for variable selection where the unlabeled data influences the important variables to be selected. As an example of (i), consider the basic Regularized Laplacian approach that solves

$$\min_{\boldsymbol{f}} \|\boldsymbol{Y}_L - \boldsymbol{f}_L\|_2^2 + \lambda_1 \boldsymbol{f}^\top \left(\text{diag}\left(\boldsymbol{W}\vec{1}\right) - \boldsymbol{W}\right) \boldsymbol{f}. \tag{2}$$

Optimization (2) is a special case of optimization (1) with $\lambda_2 = \infty$ and $\gamma = 0$. In this case, the `formula` command must be modified to use a *dissimilarity weighted graph* with the function `dG` in the `formula` call. In order to fit this, one should first scale the data according to the labeled means and variances using the `x.scaleL` function. Note that the `formula` $y \sim .$ used in the general *safe* case above to fit optimization (1) is equivalent to $y \sim .+$`dG(x.scaleL(.,`L`),metric=`"cosine"). At any rate, the `s4pm` function in **SemiSupervised** solves (2) with the following code.

```
R> xscale <- x.scaleL(bbbDescr, L)
R> rlap <- s4pm(y.fit ~ dG(xscale, metric = "cosine"), gam = 0.0,
R+             control = ctrl)
R> rlap

S4PM Fit with (n,|L|)=( 208 , 21 ) or  10 % labeled

Performance Estimates:
k-CV:  0.676  GCV:  0.038  DF:  0.171

Fit Estimates:
Graph Kernel h:  0.12  Lagrangians:  0.01   0

R> sRMSEU(fitted(rlap)[U], logBBB[U], msy)

[1] 1.342094
```

As before, the performance estimates list the optimal 3-fold CV, corresponding GCV, and degrees of freedom. The tuning parameter estimates are $(h, \lambda_1, \gamma) = (0.12, 0.01, 0.0)$, and parameter λ_2 is not shown because the $\boldsymbol{X}\boldsymbol{\beta}$ term from (1) is not fit in optimization (2).

The diffusion kernel is another popular semi-supervised optimization alternative that can be fit with the `s4pm` by solving optimization

$$\min_{\boldsymbol{f}} \|\boldsymbol{Y}_L - \boldsymbol{f}_L\|_2^2 + \lambda_1 \boldsymbol{f}^\top \left(\text{diag}\left(\boldsymbol{W}\vec{1}\right)^{-1/2} \left(\text{diag}\left(\boldsymbol{W}\vec{1}\right) - \boldsymbol{W}\right) \text{diag}\left(\boldsymbol{W}\vec{1}\right)^{-1/2}\right) \boldsymbol{f}$$

with the following R code.

```
R> diffkern <- s4pm(y.fit ~ dG(xscale, metric = "cosine"), gam = 0.0)
R> sRMSEU(fitted(diffkern)[U], logBBB[U], msy)

[1] 1.32251
```

Lastly, label spreading is yet another alternative optimizing

$$\min_{\boldsymbol{f}} \|\boldsymbol{Y}^* - \boldsymbol{f}\|_2^2 + \lambda_1 \boldsymbol{f}^\top \left(\text{diag}\left(\boldsymbol{W}\vec{1}\right)^{-1/2} \left(\text{diag}\left(\boldsymbol{W}\vec{1}\right) - \boldsymbol{W} \right) \text{diag}\left(\boldsymbol{W}\vec{1}\right)^{-1/2} \right) \boldsymbol{f},$$

where $\boldsymbol{Y}_L^* = \boldsymbol{Y}_L$ and $\boldsymbol{Y}_U^* = \vec{0}$, and is directly fit as follows.

```
R> lspread <- s4pm(y.fit ~ dG(xscale, metric = "cosine"), gam = Inf)
R> sRMSEU(fitted(lspread)[U], logBBB[U], msy)

[1] 1.414956
```

In these Blood Brain examples, the joint optimization (1) with the safety term $\boldsymbol{X}\boldsymbol{\beta}$ provides substantially improved performance over these special cases. The joint optimization (1), e.g., the default settings for function `s4pm`, is recommended for general usage.

As mentioned previously, the `s4pm` can also perform variable selection. The following code shows how the S4-generic object functions underlying this software can be easily manipulated to determine important variables similar to R's `drop1` style command [30]. This code takes about 13 seconds to run.

```
R> drp1 <- (do.call("c", lapply(names(bbbDescr), function(i){
R+   form <- as.formula(paste("y ~ . - ", i, sep = ""))
R+   g1 <- s4pm(form,data = data.frame(y = y.fit, bbbDescr),
R+              hs = hparm(safe.spread), gams = gparm(safe.spread),
R+              lams = lparm(safe.spread))
R+   measures(g1)[2]
R+ })) / measures(safe.spread)[2] - 1) * 100
R> attr(drp1, "names") <- names(bbbDescr)
R> round(sort(drp1, decreasing = TRUE)[1:5], 4)

      fpsa3   peoe_vsa.5 dipole_moment      vsa_pol          pol
     2.9492       2.7100       2.5221       1.6540       1.3797
```

The top 5 variables using %-Degradation in GCV are given, i.e., there is approximately a 3% degradation in GCV performance due to the exclusion of variable 'fpsa3'. This variable selection approach is unique compared to corresponding supervised alternatives since the bias/variance trade-off on the unlabeled data influences the selected variables.

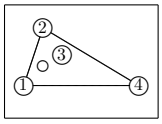
4.2 agraph: Regression with anchor graphs

The agraph function implements a semi-supervised technique using an anchor point method for graph construction. To start, a clustering algorithm is applied to $\boldsymbol{X}$. Let $\boldsymbol{M}$ denote the $a \times p$ set of cluster centers. The main idea is define a length a vector $\boldsymbol{z}_i$ so that each data row $\boldsymbol{x}_i$ is close to $\boldsymbol{z}_i\boldsymbol{M}$. Specifically, $\boldsymbol{z}_i$ is the vector of probability weights used to project $\boldsymbol{x}_i$ onto the convex polytope consisting of the s closest anchor points in $\boldsymbol{M}$ (refer below for examples). Repeating this for each row, yields a dimension reduced $n \times a$ matrix $\boldsymbol{Z}$. The Laplacian graph is then computed using adjacency $\boldsymbol{W} = \boldsymbol{Z}\text{diag}(\boldsymbol{Z}^\top\vec{1})^{-1}\boldsymbol{Z}^\top$ with $\boldsymbol{Z}\vec{1} = \vec{1}$. Optimization (1) is now used but with $\boldsymbol{f} = \boldsymbol{Z}\boldsymbol{\gamma}$, i.e., $\boldsymbol{f}$ is restricted to the column-space of $\boldsymbol{Z}$. From this, the anchor graph implementation approximates optimization (1) with

$$\min_{\boldsymbol{\beta},\boldsymbol{\gamma},\boldsymbol{Y}_U} L\left(\boldsymbol{Y}_L, \boldsymbol{Z}_L\boldsymbol{\gamma} + \boldsymbol{X}_L\boldsymbol{\beta}\right) + \|\boldsymbol{Y}_U - \boldsymbol{Z}_U\boldsymbol{\gamma} - \boldsymbol{X}_U\boldsymbol{\beta}\|_2^2 + \lambda_1\boldsymbol{\gamma}^\top\boldsymbol{B}^\star\boldsymbol{\gamma} + \lambda_2 \parallel \boldsymbol{\beta} \parallel_2^2 + \gamma \|\boldsymbol{Y}_U\|_2^2$$

where $\boldsymbol{B}^\star = \boldsymbol{Z}^\top\boldsymbol{Z} - \boldsymbol{Z}^\top\boldsymbol{Z}\text{diag}(\boldsymbol{Z}^\top\vec{1})^{-1}\boldsymbol{Z}^\top\boldsymbol{Z}$ is the so-called *reduced Laplacian* [21].

The AnchorGraph method in **SemiSupervised** is used in R to get $\boldsymbol{Z}$. The details for the actual projection of a vector $\boldsymbol{x}$ onto the convex polytope of the centroids from a clustering algorithm using AnchorGraph are now provided. Consider the $p = 2$ and $s = 4$ example to the right. The data point $\boldsymbol{x} = (2, 3)$ denoted by an empty circle is contained in the convex polygon with anchors labeled 1-4 having Cartesian coordinates $(1, 2), (2, 5), (3, 4), (7, 2)$ and is fit in R below.

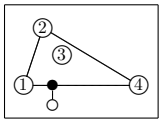



```
R> ctrl.agr <- SemiSupervised.control(cn = 50)
R> anchors <- rbind(c(1, 2), c(2, 5), c(3, 4), c(7, 2))
R> z <- AnchorGraph(c(2, 3), anchor = anchors, control = ctrl.agr)$Z
R> as.vector(z)

[1] 0.53276500 0.19673762 0.20492595 0.06557143

R> as.vector(z %*% anchors)

[1] 2.000018 3.000065
```

In other words, the vector $\boldsymbol{z}$ is chosen as probability weights so that

$$\begin{aligned}\begin{pmatrix}2.000018\\3.000065\end{pmatrix} &= 0.53276500\begin{pmatrix}1\\2\end{pmatrix} + 0.19673762\begin{pmatrix}2\\5\end{pmatrix} + 0.20492595\begin{pmatrix}3\\4\end{pmatrix} + 0.06557143\begin{pmatrix}7\\2\end{pmatrix}\\ &\approx \begin{pmatrix}2\\3\end{pmatrix} = \boldsymbol{x}.\end{aligned}$$

The parameter c_n controls the threshold for number of iterations internally in this algorithm, and larger values approximate the point better. In practice, close approximation to row $\boldsymbol{x}$ is not all that critical in terms of performance, so this parameter is defaulted quite low to $c_n = 4$.



Perhaps more astounding in practice are circumstances where the observation $\boldsymbol{x}$ is outside the convex polygon of the s closest anchor points. Such a circumstance is depicted in the example to the right with the same anchors as above, but now $\boldsymbol{x} = (2.5, 1.0)$. In this case, a Local Anchor Embedding algorithm [21] is implemented in C++ to project the row back onto the convex polygon consisting of the s closest anchor points in $\boldsymbol{M}$. The vector $\boldsymbol{z}$ is displayed in R below and defines the row probabilities necessary for this projection.

```
R> z <- AnchorGraph(c(2.5, 1), anchor = anchors, control = ctrl.agr)$Z
R > as.vector(z)

[1] 0.7500015 0.0000000 0.0000000 0.2499985
```

This example illustrates some important properties of the anchor graph approximation. First, the approximation $\boldsymbol{z}\boldsymbol{M} \rightarrow (2.5, 2)$ for $\boldsymbol{x}$ will always miss the mark. Second, $\boldsymbol{z}$ only involves anchors 1 and 4, but anchor 4 is the furthest away from $\boldsymbol{x}$ while the much closer anchors 2 and 3 are ignored. The aggregate reliance on anchors in this manner tends to lead to more jagged classification borders or regression fits in practice than compared to the `s4pm` method. This jaggedness tends to be more pronounced as p increases.

While the above anchor graph fitting details are hidden from the user by default, the function `agraph` uses them below.

```
R> safe.agraph <- agraph(y ~ ., data = data.frame(y = y.fit, bbbDescr))
R> safe.agraph

Anchor Graph Laplacian (agraph) with (n,|L|)=( 208 , 21 ) or  10 % labeled

Performance Estimates:
k-CV:  0.653  GCV:  0.025  DF:  6.148

Fit Estimates:
Lagrangians:  1   0.001  Safe-Lagrangian 2

R> sRMSEU(fitted(safe.agraph)[U], logBBB[U], msy)

[1] 1.105722
```

The output produces estimates $(\lambda_1, \lambda_2, \gamma) = (1.0, 2.0, 0.001)$. Notice that there is no h parameter which makes sense, since this method does not rely on a local kernel. The `agraph` approach performs slightly better than the `s4pm` as well. As with the `s4pm`, the following code shows that one can issue this command without the safety term $\boldsymbol{X}\boldsymbol{\beta}$ using the `aG` function.

```
R> agr<-agraph(y.fit ~ aG(AnchorGraph(xscale)))
R> sRMSEU(fitted(agr)[U], logBBB[U], msy)

[1] 1.298504
```

4.3 Out-of-sample prediction

The **SemiSupervised** package offers a prediction routine for either a single observation $\boldsymbol{x}_0$ or group of observations. The prediction rule for the `agraph` is trivial. Given vector $\boldsymbol{x}_0$ or a group of predictions, it just builds the `AnchorGraph` with this as input and and then linearly predicts using the trained $\widehat{\boldsymbol{\beta}}$ from optimization (3). However, unlike `agraph` the `s4pm` procedure does not lend itself to an obvious out-of-sample prediction rule, but out-of-sample prediction in its most basic sense is always possible with continuous feature data by interpolating the predictions $\widehat{\boldsymbol{\eta}}$. To this end, recall that $\boldsymbol{W}$ is the Gram matrix of local kernel $K_h(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and let $\boldsymbol{W}_{i0} = K_h(\boldsymbol{x}_i, \boldsymbol{x}_0)$ for $i \in L \cup U$. The interpolation function

$$\widehat{\boldsymbol{\eta}}(\boldsymbol{x}_0) = \boldsymbol{x}_0^\top \widehat{\boldsymbol{\beta}} + \frac{\sum_i a_i \widehat{\boldsymbol{f}}_i}{\sum_i a_i} \text{ with } a_i = \frac{\boldsymbol{W}_{i0}}{1 - \boldsymbol{W}_{i0}}$$

is implemented for out-of-sample predictions.

The following code randomly perturbs the first 5 observations of the Blood Brain data and then predicts them with the safe approaches above.

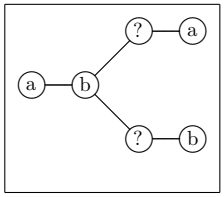
```
R> set.seed(10)
R> xnew <- as.data.frame(jitter(as.matrix(bbbDescr)[1:5, ]))
R> round(predict(safe.lap, xnew = xnew), 3)

[1] -0.937  0.759 -3.934  2.438  1.129

R> round(predict(safe.spread, xnew = xnew), 3)

[1] -0.824  0.634 -3.552  2.303  1.058

R> round(predict(safe.agraph, xnew = xnew), 3)

[1] -1.450  1.195 -4.987  3.358  1.396
```

Prediction with the other routines is more complicated because the graph must be constructed from the feature data in each case. The following code presents this case for the regularized Laplacian and the anchor graph only approaches above.

```
R> sxnew <- scale(xnew,center = attr(xscale, "scaled:center"),
R+                      scale = attr(xscale, "scaled:scale"))
R> gnew <- kgraph.predict(cosineDist(sxnew, xscale))
R> round(predict(rlap, gnew = gnew), 3)

[1] 0.219 0.032 0.109 0.437 0.216

R> round(predict(agr,gnew = AnchorGraph(sxnew, fit.g = gmatrix(agr))), 3)

[1]  0.007 -0.116  0.075  0.553  0.077
```

In practice, the safe versions tend to out-perform the non-safe versions on real data and the above stability improvement of these methods is expected.

4.4 Observed graph example



Some real data examples involve an *observed graph* that was not computed from some data matrix $\boldsymbol{X}$. Such examples can occur in classification or regression and include protein interaction networks [35], terrorist networks [27], and shopping networks [17]. In this case, $\boldsymbol{W}$ is an $n \times n$ adjacency matrix defined on both the labeled and unlabeled cases. Optimization (1) is fit with $\boldsymbol{\beta} = \vec{0}$ and $\boldsymbol{B}$ as either the Laplacian or normalized Laplacian. The **SemiSupervised** package provides an internal `formula` function term `sG` for handling such an observed *similarity graph* adjacency matrix as input. Consider the observed graph to the right. The objective is to classify the nodes of the graph as "a" or "b." There are two unlabeled observations marked by "?." The following code fits this observed graph in R with the `s4pm`.

```
R> onePos <- list(1:2, 1:4, c(2:3, 6), c(2, 4, 5), c(4, 5), c(3, 6))
R> (W <- do.call("rbind", lapply(onePos,
R+                       function(i){v = rep(0, 6); v[i] = 1; v})))

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    1    0    0    0    0
[2,]    1    1    1    1    0    0
[3,]    0    1    1    0    0    1
[4,]    0    1    0    1    1    0
[5,]    0    0    0    1    1    0
[6,]    0    0    1    0    0    1

R> y <- as.factor(c("a", "b", NA, NA, "a", "b"))
R> graph.ex <- s4pm(y ~ sG(W))
R> graph.ex

S4PM Fit with (n,|L|)=( 6 , 4 ) or  67 % labeled

Performance Estimates:
k-CV:  0  GCV:  0.002  DF:  0

Fit Estimates:
Lagrangians:  0.01   0
```

In this type of observed similarity graph example, there is no feature data matrix $\boldsymbol{X}$ or kernel function, so the estimated Lagrangian parameters are $(\lambda_1, \gamma_1) = (0.01, 0.0)$. The fit is saturated and optimal in terms of CV. The following code outputs the node predictions and their corresponding probabilities.

```
R> fitted(graph.ex)

[1] a b b a a b
Levels: a b
```

```
R> round(fitted(graph.ex, type = "prob"), 2)

[1] 0.12 0.88 0.90 0.40 0.11 0.89
```

The returned probability estimates of $P(\boldsymbol{y} = \text{“b”}|\boldsymbol{W})$ are consistent with what one might expect given the location of the missing cases on the observed graph.

In some cases, $\boldsymbol{X}$ data from a different view is available in conjunction with an observed similarity graph $\boldsymbol{W}$. Possibly in the protein example mentioned earlier, $\boldsymbol{W}$ is an observed similarity graph of protein interactions where specifically the nodes of the graph are n individual proteins. In addition, each protein could also have an independent observation vector forming an $n \times p$ chemistry $\boldsymbol{X}$ data view. When such data $\boldsymbol{W}$ and $\boldsymbol{X}$ arise from 2 sources (and hence $\boldsymbol{X}$ is not used to compute $\boldsymbol{W}$), the function `s4pm` can still easily accommodate this type of data analysis as well.

4.5 Package heuristics for big data

Much work on computational efficiency was done to fit the semi-supervised technique quickly in memory. Since this approach is based on symmetric matrices, a Cholesky decomposition and efficient storage of matrices in conjunction with optimized LAPACK/ATLAS C-routines boosted speed substantially. An even larger speedup was due to the following, *Stagewise Cross Validation* (SCV) scheme. For the `s4pm`, adjacencies in the graph

$$\boldsymbol{W}_{LL} + \boldsymbol{B}_{LU}\boldsymbol{B}_{UU}^{-1}\boldsymbol{B}_{UL} \tag{3}$$

on L were computed to quantify labeled-to-labeled connectives through labeled and unlabeled networks in the larger graph $\boldsymbol{W}$ on $L \cup U$ [6] with $\boldsymbol{B}$ as either the Laplacian or normalized Laplacian. Four parameters $h, \lambda_1, \lambda_2, \gamma$ were then estimated with 3-fold CV from graph (3) and data $\boldsymbol{X}_L$. (While $\boldsymbol{X}_L$ played the role of $\boldsymbol{X}$ during the fitting of a given fold, the responses in the other 2 folds were omitted and played the role of $\boldsymbol{Y}_U$.) Lastly, parameter γ was re-estimated with respect to the full graph $\boldsymbol{W}$ with fixed, optimal values for h, λ_1, λ_2 to stabilize the predictions. The `agraph` uses a very similar adjustment for stage-wise cross-validation where $\lambda_1, \lambda_2, \gamma$ are estimated on data $(\boldsymbol{Z}_L, \boldsymbol{X}_L)$ while the semi-supervised parameter γ was then reestimated on $(\boldsymbol{Z}, \boldsymbol{X})$ with the other parameters fixed. The *cv.type='scv'* control parameter performs SCV by default.

For large data sets, the semi-supervised method is adjusted to use anchor points by default. In this adjustment, the k-means algorithm is run to obtain a centroids (anchors). Optimization (1) is then recast as follows. All unlabeled cases are treated as the anchor points. This is automated through the control parameters *U.as.anchor* and *U.as.anchor.thresh*. Specifically, if the data set has $n >$ *U.as.anchor.thresh* and *U.as.anchor=TRUE* then this substantial time saving is employed for any model fit by the **SemiSupervised** package. Note *U.as.anchor.thresh* is the number of centroids in k-means and defaults to 600. During development, we exhaustively tested using the anchor points as unlabeled cases versus using the unlabeled data directly and found no appreciable degradation in performance.

5 Harmonic functions and the spa package

Harmonic functions in semi-supervised learning have a long history in this up and coming field. Let $\boldsymbol{S}$ =diag$(\boldsymbol{W}\vec{1})^{-1}\boldsymbol{W}$. A function is harmonic if it satisfies the averaging property

$$\boldsymbol{f}_U = (\boldsymbol{I} - \boldsymbol{S}_{UU})^{-1} \boldsymbol{S}_{UL}\boldsymbol{f}_L.$$

It is now understood that optimization (1) with $\gamma = 0$ and $\boldsymbol{B}$ as the combinatorial Laplacian results in a harmonic function approach independent of the loss function [6]. This work also defined the *joint harmonic function*

$$\min_{\boldsymbol{f},\boldsymbol{Y}_U} (\boldsymbol{Y} - \boldsymbol{f})^\top \boldsymbol{W} (\boldsymbol{Y} - \boldsymbol{f}) + \lambda \boldsymbol{f}^\top \left(\text{diag}\left(\boldsymbol{W}\vec{1}\right) - \boldsymbol{W}\right) \boldsymbol{f} + \gamma \left\|\boldsymbol{Y}_U\right\|_2^2, \tag{4}$$

which is a harmonic function when $\gamma = 0$. Non-zero γ does not result in a harmonic function, but does often improve performance and stabilizes the numerical analysis. Interestingly, if $\boldsymbol{W}$ is generated from a local kernel function with $\lambda = 1$, then the solution to optimization (4) converges to a shrunken supervised local kernel function as $\gamma \to \infty$. Specifically, it was proven that in classification the class labels are exactly equivalent to the supervised class assignments for kernel regression. Modifying λ has a minor effect on this convergence result. This is a safety feature since a supervised technique results as a special case, but this connection only occurs in classification problems. In our experience, solutions to (4) tend to perform poorly in regression problems.

The **SemiSupervised** package fits the joint harmonic function approach using its `jtharm` function. The observed graph from Section 4.4 is now used to fit this technique with the R code below.

```
R> safe.jtharm <- jtharm(y ~ sG(W))
R> safe.jtharm

Joint Harmonic Fit with (n,|L|)=( 6 , 4 ) or  67 % labeled

Performance Estimates:
k-CV:  0  GCV:  7.589  DF:  0.824

Fit Estimates:
Lagrangian:  0.01     Safe-Lagrangian:  0.001
```

The Lagrangian parameter vector $(\lambda, \gamma) = (0.01, 0.001)$ was estimated for this joint harmonic function approach.

Another approach to fit harmonic functions was the heuristic sequential predictions algorithm [5]. This algorithm simultaneously fits a function over the nodes of a graph and a linear term $\boldsymbol{X}\boldsymbol{\beta}$ where specifically the node on the graph corresponds to a row in the $\boldsymbol{X}$ data. In some cases, the graph can be constructed from the feature data, and a *safe* version would be possible to fit. The **spa** package was recently rewritten to inherit the S4-generic class structure and `formula` interface of the **SemiSupervised** package. In addition, the CV and internal fitting was also recently rewritten in C++, so this package is quite a bit faster than its original version. The following code gives the fit from `spa` to the Blood Brain data.

```
R> library("spa")
R> safe.spa <- spa(y ~ ., data = data.frame(y = y.fit, bbbDescr))
R> sRMSEU(fitted(safe.spa)[U], logBBB[U], msy)

[1] 1.220487
```

The performance is slightly worse than the `s4pm`, but is still competitive. The technical differences between the original **spa** version 1.0 and the updated **spa** version 3.0 are described in its vignette available in version 3.0.

6 Conclusions

The **SemiSupervised** package was presented for semi-supervised machine learning problems. The package was noted to scale linearly in n. In this capacity, it is important to draw a distinction between scaling issues associated with labeled (training) size m and the full data size n. Most machine learning techniques do not scale well with m, but have real promise for scaling with n. For example, take supervised learning; prediction of n observations is often trivial post training and is rarely discussed as a computational challenge. Semi-supervised learning is at a disadvantage in terms of time complexity because many techniques, especially the popular and high performing graph-based techniques, are cubic in n due to the required inversion of $n \times n$ matrices. The need for linear techniques has led to the recent work on anchor point methods together with the use of linear approximations to non-parametric semi-supervised techniques [21]. The **SemiSupervised** package builds on this work.

To summarize the important contributions of this work, we list the internal steps taken by the function `agraph` and simply note that they are indeed linear in n.

1. **Clustering step:** A clustering algorithm is applied to all n observations to produce a anchor points (i.e., centroids). The algorithm defaults to k-means clustering which has order npa with $a = 600$ by default.

2. **Graph construction step:** A *local anchor embedding gradient decent algorithm* is applied to the full n observations and a anchor points to get $\boldsymbol{Z}$. Upon completion, the graph is represented by an $n \times n$ adjacency matrix $\boldsymbol{W}$, but this is never actually computed thus circumventing a quadratic in n step (i.e., just compute $\boldsymbol{B}^\star$ directly from $\boldsymbol{Z}$ in the anchor graph optimization of Section 4.2).

3. **Fitting step:** The `agraph` function is applied to compute a vector of linear regression coefficient estimates $\widehat{\boldsymbol{\beta}}$, but this only requires the inverse of an $(a+p) \times (a+p)$ matrix. By fitting anchor points as unlabeled cases as opposed to the unlabeled cases directly, this step is constant as n increases.

4. **Prediction step:** The n predictions are produced linearly using $\widehat{\boldsymbol{\beta}}$.

The `agraph` is the linear approximation to the `s4pm`. Using this general algorithm and adapting the anchor points to the `s4pm` without approximating the non-parametric fit dramatically

improves the scalability of this method. However, it is still not as fast as the `agraph`, but theoretically should slightly out-preform the `agraph` method in practice.

The summary above also exposes a significant shortcoming of the **SemiSupervised** package and the work in semi-supervised learning in general. As p increases, these techniques have a terrible time scaling up. The need to address large p problems is, to our knowledge, entirely open in this field and is worthy of future study. This is not only an issue from a computational perspective but also from a theoretical perspective, e.g., it is not clear what effect allowing the unlabeled data to influence variable selection will have on variable selection in practice. In Section 4.1, we demonstrated that this is possible with the **SemiSupervised** package, but this is the only result that we are aware of that even considers this issue. Understanding this effect is an important practical problem that presents a fruitful future study.

In addition to having linear in n time complexity, the **SemiSupervised** package also offers internal CV routines and underlying fitting in C++, S4 generic consistency in R, and several practical heuristics to make this work on real arbitrary data. These advances provide an easy-to-use tool for real data problems.

Acknowledgments

The authors thank Prithish Banerjee and Michael Morehead for their help with this package. The work of Mark Vere Culp was supported in part by the NSF CAREER/DMS-1255045 grant. The opinions and views expressed in this paper are those of the authors and do not reflect the opinions or views at the NSF.

References

[1] M. Belkin, I. Matveeva, and P. Niyogi. Regularization and semi-supervised learning on large graphs. In *COLT*, pages 624–638, 2004.

[2] M Belkin, P Niyogi, and V Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

[3] O Chapelle, B Schölkopf, and A Zien. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[4] O Chapelle, V Sindhwani, and S Keerthi. Optimization techniques for semi-supervised support vector machines. *Journal of Machine Learning Research*, 9:203–233, 2008.

[5] MV Culp. **spa**: A semi-supervised R package for semi-parametric graph-based estimation. *Journal of Statistical Software*, 40(10):1–29, 2011.

[6] MV Culp and KJ Ryan. Joint harmonic functions and their supervised connections. *Journal of Machine Learning Research*, 14:3721–3752, 2013.

[7] N Fazakis, S Karlos, S Kotsiantis, and K Sgarbas. Self-trained lmt for semisupervised learning. *Computational Intelligence and Neuroscience*, 2016:1–13, 2016.

[8] J Friedman, T Hastie, and R Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

[9] TM Huang and V Kecman. ***SemiL**, Semi-Supervised Learning Software*, 2005. Version 1.0.

[10] T Joachims. Transductive learning via spectral graph partitioning. In T Fawcett and N Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 290–297, Washington, DC, USA, 2003. ACM.

[11] M Kawakita and T Jun'ichi. Safe semi-supervised learning based on weighted likelihood. *Neural Networks*, 53(1):146–164, 2014.

[12] M Kuhn. Building predictive models in R using the **caret** package. *Journal of Statistical Software*, 28(5):1–26, 2014.

[13] M Kuhn and K Johnson, editors. *Applied Predictive Modeling*. Springer-Verlag, New York, NY, 2013.

[14] J Lafferty and L Wasserman. Statistical analysis of semi-supervised regression. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 801–808. Curran Associates, Inc., 2008.

[15] F Langrognet, R Lebret, C Poli, and S Iovleff. ***Rmixmod**: Supervised, Unsupervised, Semi-Supervised Classification with MIXture MODelling (Interface of MIXMOD Software)*, 2016. R Package Version 2.1.1.

[16] Friedrich Leisch and Evgenia Dimitriadou. ***mlbench**: Machine Learning Benchmark Problems*, 2010. R Package Version 2.1-1.

[17] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[18] Y Li and Z Zhou. Towards making unlabeled data never hurt. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1081–1088, New York, NY, USA, 2011. ACM.

[19] YF Li, JT Kwok, and ZH Zhou. Towards safe semi-supervised learning for multivariate performance measures. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 1816–1822. AAAI Press, 2016.

[20] M Lichman. UCI machine learning repository, 2013.

[21] W Liu, J He, and S Chang. Large graph construction for scalable semi-supervised learning. In *Proceedings of the 27rd International Conference on Machine Learning*, pages 679–687, Haifa, Israel, 2010. ACM.

[22] W Liu, C Mu, S Kumar, and S Chang. Discrete graph hashing. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3419–3427. Curran Associates, Inc., 2014.

[23] Askar Obulkasim. ***HCsnip**: Semi-Supervised Adaptive-Height Snipping of the Hierarchical Clustering Tree*, 2016. R Package Version 1.14.0.

[24] RK Pace and R Barry. Sparse spatial autoregressions. *Statistics and Probability Letters*, 33(1):291–297, 1997.

[25] N Russell, L Cribbin, and TB Murphy. ***upclass**: Updated Classification Methods using Unlabeled Data*, 2014. R Package Version 2.0.

[26] KJ Ryan and MV Culp. On semi-supervised linear regression in covariate shift problems. *Journal of Machine Learning Research*, 16:3183–3217, 2015.

[27] F Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)*, 34(1):1–47, 2002.

[28] V Sindhwani and S Keerthi. Large scale semi-supervised linear svms. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 477–484, New York, NY, USA, 2006. ACM.

[29] A Singh, R Nowak, and X Zhu. Unlabeled data: Now it helps, now it doesn't. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1513–1520. Curran Associates, Inc., 2009.

[30] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[31] J Wang, T Jebara, and S Chang. Semi-supervised learning using greedy max-cut. *Journal of Machine Learning Research*, 14:771–800, 2013.

[32] J Wang, X Shen, and W Pan. On efficient large margin semisupervised learning: Method and theory. *Journal of Machine Learning Research*, 10:719–742, 2009.

[33] W Wang and Z Zhou. A new analysis of co-training. In *Proceedings of the 27rd International Conference on Machine Learning*, pages 1135–1142, Haifa, Israel, 2010. ACM.

[34] Chris Whong. Liberating data from nyc property tax bills. `http://chriswhong.com/open-data/liberating-data-from-nyc-property-tax-bills/`, 2016. Accessed: 2016-11-19.

[35] Y Yamanishi, J Vert, and M Kanehisa. Protein network inference from multiple genomic data: A supervised approach. *Bioinformatics*, 20:363–370, 2004.

[36] D Zhou, O Bousquet, TN Lal, J Weston, and B Schölkopf. Learning with local and global consistency. *Advances in Neural Information Processing Systems*, 16(16):321–328, 2004.