Efficient Non-Parametric Function Induction in Semi-Supervised Learning

Yoshua Bengio, Olivier Delalleau and Nicolas Le Roux
Département d'Informatique et Recherche Opérationnelle
Université de Montréal
Montréal, Québec, Canada, H3C 3J7
{bengioy,delallea,lerouxni}@iro.umontreal.ca

Technical Report 1247,
Département d'Informatique et Recherche Opérationnelle

May 6, 2004

Abstract

There has been an increase of interest for semi-supervised learning recently, because of the many datasets with large amounts of unlabeled examples and only a few labeled ones. This paper follows up on proposed non-parametric algorithms which provide an estimated continuous label for the given unlabeled examples. It extends them to function induction algorithms that correspond to the minimization of a regularization criterion applied to an out-of-sample example, and happens to have the form of a Parzen windows regressor. The advantage of the extension is that it allows predicting the label for a new example without having to solve again a linear system of dimension n (the number of unlabeled and labeled training examples), which can cost $O(n^3)$. Experiments show that the extension works well, in the sense of predicting a label close to the one that would have been obtained if the test example had been included in the unlabeled set. This relatively efficient function induction procedure can also be used when n is large to approximate the solution by writing it only in terms of a kernel expansion with $m \ll n$ terms, and reducing the linear system to m equations in m unknowns.

1 Introduction

In many applications of machine learning, the labeled examples only represent a small fraction of all the available data. This situation has created a spur of research activity in the area of semi-supervised learning algorithms, that can take advantage of both types of examples. We refer to (Seeger, 2001) for a good overview of the issues involved. Most approaches to semi-supervised learning make some implicit or explicit assumption about the joint distribution of the input and output variables. An exception

is the set of regularization methods (e.g. as in (Schuurmans and Southey, 2002)) that use the unlabeled data to discover overfitting. Among the other approaches one would traditionally characterize the methods as parametric or non-parametric, and as using an explicit generative model (e.g. considering the labels as hidden variables in a graphical model, see (McCallum and Nigam, 1998)) or not. Interestingly, with explicit parametric assumptions of the class-conditional input distribution (Cozman, Cohen and Cirelo, 2003), one can show that these assumptions (if not perfectly valid) yield both a decrease in variance and an increase in bias, and the more so when the relative amount of unlabeled data increases. To us this strongly suggests that good results could be obtained with non-parametric methods when the amount of unlabeled data is large and little prior information on the input distribution is available.

Fortunately, in the very recent past, several non-parametric approaches to semi-supervised learning have been introduced, e.g. in (Szummer and Jaakkola, 2002; Chapelle, Weston and Scholkopf, 2003; Belkin and Niyogi, 2003; Zhu, Ghahramani and Lafferty, 2003; Zhu, Lafferty and Ghahramani, 2003; Zhou et al., 2004). They rely on weak implicit assumptions on the generating data ditribution, e.g. smoothness of the target function with respect to a given notion of similarity between examples (represented by a kernel function)[1]. For classification tasks this amounts to assuming that the target function is constant within the region of input space (or "cluster" (Chapelle, Weston and Scholkopf, 2003)) associated with a particular class. All of these previous non-parametric approaches exploit the idea of building and smoothing a graph in which each example is associated with a node, and arcs betweeen two nodes are associated with a similarity function applied on the input part of the corresponding two examples. For some of these methods one first builds a representation (e.g. (Belkin and Niyogi, 2003)) or a kernel (Chapelle, Weston and Scholkopf, 2003) using the input part of the data (of both labeled and unlabeled cases), and then trains a supervised learning algorithm with the labeled examples but relying on the representation or kernel induced using the unlabeled examples. In the other methods (Szummer and Jaakkola, 2002; Zhu, Ghahramani and Lafferty, 2003; Zhu, Lafferty and Ghahramani, 2003; Zhou et al., 2004) one solves an optimization problem in which both the labeled and unlabeled cases intervene: the idea is to propagate the label information from the labeled cases in the graph, with stronger propagation between similar examples.

It is not always clear with these graph-based kernel methods for semi-supervised learning how to generalize to previously unseen test examples. In (Zhu, Lafferty and Ghahramani, 2003) it is proposed to assign to the test case the label (or inferred label) of the nearest neighbor from the training set (labeled or unlabeled). In this paper we derive from the training criterion an inductive formula that is in fact a cousin of the nearest neighbor solution. In general the above graph-based approaches have been designed for the transductive setting, in which the input part of the test examples must be provided before doing the expensive part of training. This typically requires solving a linear system with n equations and n parameters, where n is the number of labeled and unlabeled examples. In a truly inductive setting where new examples are given one after the other and a prediction must be given after each example, it can be very computationally costly to solve such a system anew for each of these test examples.

[1] See also (Kemp et al., 2004) for a hierarchically structured notion of a priori similarity.

This paper starts from this problem and proposes a natural generalization of the graph-based semi-supervised learning algorithms that allows one to cheaply perform function induction, i.e. for a computational cost that is $O(n)$. The main idea is to apply the same smoothness criterion that is behind the original semi-supervised algorithm, adding the terms corresponding to the new example, but keeping the predictions fixed for the training examples (both the labeled and unlabeled ones).

In addition to providing a cheap alternative for doing function induction, the proposed approach opens the door to efficient approximations even in the transductive setting. Since we know the analytic functional form of the prediction at a point x in terms of the predictions at a set of training points (it turns out to be a Parzen windows predictor) we can use it to express all the predictions in terms of a small subset of $m \ll n$ examples (i.e. a low-rank approximation) and solve a linear system with $O(m)$ variables and equations.

In the next section we formalize a family of smoothness criteria giving rise to already proposed non-parametric semi-supervised learning algorithms. In section 3 we justify and derive the function induction formula. In section 4.2.1 we show in experiments that the out-of-sample induction is very close to the transductive prediction. In section 4.2.2 we compare variants of the proposed semi-supervised algorithm with two very closely related previously proposed ones (Zhu, Ghahramani and Lafferty, 2003; Zhou et al., 2004). Finally, in section 5 we present an approximation method to reduce the computational time and the memory requirements by solving a smaller linear system.

2 Non-Parametric Smoothness Criteria

Among the previously proposed approaches, several can be cast as the minimization of a criterion (often a quadratic form) in terms of the function values $f(x_i)$ at the labeled and unlabeled examples x_i, as follows:

$$\begin{aligned} C_{K,D,D',\lambda}(f) &= \frac{1}{2} \sum_{i,j \in U \cup L} K(x_i, x_j) D(f(x_i), f(x_j)) \\ &+ \lambda \sum_{i \in L} D'(f(x_i), y_i) + R(f) \end{aligned} \tag{1}$$

where U is the unlabeled set, L the labeled set, x_i the input part of the i-th example, y_i the target label, $K(\cdot, \cdot)$ is a positive similarity function (e.g. a Gaussian kernel) applied on a pair of inputs, and $D(\cdot, \cdot)$ and $D'(\cdot, \cdot)$ are lower-bounded dissimilarity functions applied on a pair of output values. $R(f)$ is an optional additional regularization term of the values of f at x_i. In particular, in (Zhou et al., 2004), the proposed criterion amounts to $R(f) = \lambda \sum_{i \in U} f(x_i)^2$. To obtain a quadratic form in $f(x_i)$ one typically chooses D and D' to be quadratic, e.g.

$$D(y, y') = D'(y, y') = (y - y')^2 \stackrel{\text{def}}{=} D^*(y, y').$$

When D, D' and R are quadratic, this criterion can be minimized exactly for the n function values $f(x_i)$. In general this could cost $O(n^3)$ operations, possibly less if the

input similarity function $K(\cdot,\cdot)$ is sparse.

A quadratic dissimilarity function makes a lot of sense in regression problems but has also been used successfully in classification problems (where $f(\cdot)$ is not constrained to be discrete). In this paper, we only consider binary classification, but note that all the algorithms proposed extend naturally to multiclass problems. The first term of equation 1 says that we want to penalize the dissimilarity between $f(x_i)$ and $f(x_j)$ when x_i and x_j are similar. The second term says we want to penalize $f(x_i)$ for being far from the observed target value y_i (for $i \in L$). The hyperparameter λ controls the trade-off between the smoothness of f and achieving the observed values on the labeled cases. It should depend on the amount of noise in the observed values y_i, i.e. on the particular data distribution (although for example (Zhu, Ghahramani and Lafferty, 2003) consider forcing $f(x_i) = y_i$). The optional extra regularization term $R(f)$ controls how much we penalize the high values of f on the unlabeled data. Adding it may give better results when very few labels are available.

Two methods using a criterion of the form given by eq. 1 have already been proposed. In (Zhu, Ghahramani and Lafferty, 2003), $\lambda = \infty$, $R(f) = 0$ and $D = D' = D^*$. In (Zhou et al., 2004), the cost function can be represented as in eq. 1 with $\lambda > 0$, $D' = D^*$ but

$$D(f(x_i), f(x_j)) = \left(\frac{f(x_i)}{\sqrt{a_i}} - \frac{f(x_j)}{\sqrt{a_j}} \right)^2 \tag{2}$$

where

$$a_i \stackrel{\text{def}}{=} \sum_{j \in U \cup L, j \neq i} K(x_i, x_j). \tag{3}$$

In this paper we mostly study the case in which $D = D' = D^*$ and $R(f) = 0$. The minimization of the criterion with respect to all the $f(x_i)$ for $i \in L \cup U$ gives rise to the following linear system:

$$A\vec{f} = b \tag{4}$$

where $\vec{f_i} = f(x_i)$, whose first l elements are in L and the remaining $n - l$ in U. Using the matrix notation $W_{ij} = K(x_i, x_j)$, the system matrix A can be written as follows:

$$A = \lambda \Delta_L + Diag(W\mathbf{1}_n) - W \tag{5}$$

where Δ_L ($n \times n$) has entries $\Delta_{L,ij} = \delta_{ij}\delta_{i \in L}$, $Diag(v)$ is the matrix containing the vector v in its diagonal, and $\mathbf{1}_n$ is the vector of n ones. The right hand side vector b is as follows:

$$b_i = \delta_{i \in L} \lambda y_i. \tag{6}$$

In the non-parametric setting, which is the one studied by the above authors, the criterion is directly optimized with respect to the function values. This has the disadvantage of providing no obvious prediction for new examples, and the method is therefore used in the transductive setting (the test examples are included in the unlabeled set). To obtain function induction from the transductive learner, one can of course add the test point x to the unlabeled set and solve again the system, but it is a costly procedure (e.g. $O(n^3)$ for solving a linear system when D and D' are quadratic). One alternative – which should be further explored – is to parameterize $f(x)$ with a flexible form such

as a neural network or a linear combination of non-linear bases (see also (Belkin and Niyogi, 2003)). Another is the induction formula proposed below. As we will see in section 5 the two alternatives can be combined to yield an efficient approximation to the non-parametric semi-supervised learning problem.

3 Function Induction Formula

In order to transform the above transductive algorithms (for different choices of K, D and D' in eq. 1) into function induction algorithms we will do two things:

- apply the same type of smoothness constraint as in eq. 1 to the new example x,
- as in ordinary function induction (by opposition to transduction), require that the $f(x_i)$ remain fixed even though the knowledge of x has been added.

Therefore we will minimize the criterion of eq. 1 when we add terms for the new unlabeled example x, but only with respect to $f(x)$. Since the terms of the criterion depending on $f(x)$ are

$$\sum_{j \in U \cup L} K(x, x_j) D(f(x), f(x_j))$$

by minimizing them (taking $D = D' = D^*$) we readily obtain the solution

$$\tilde{f}(x) = \frac{\sum_{j=1}^{n} K(x, x_j) f(x_j)}{\sum_{j=1}^{n} K(x, x_j)}. \tag{7}$$

Interestingly this is exactly the formula for Parzen windows or Nadaraya-Watson non-parametric regression (Nadaraya, 1964; Watson, 1964) when K is the Gaussian kernel.

An interesting question to consider is whether $\tilde{f}(x_i)$ as defined above approaches $f(x_i)$ (the solution of eq. 4) for $i \in L \cup U$. Plugging $x = x_i$ in the above formula and applying the linear equations of eq. 4 yields

$$\tilde{f}(x_i) = f(x_i) - \delta_{i \in L} \frac{\lambda (y_i - f(x_i))}{\sum_{j \in L \cup U} K(x_i, x_j)}.$$

Hence $\tilde{f}(x_i) = f(x_i)$ for $i \in U$, but on labeled examples the induction formula chooses a value $\tilde{f}(x_i)$ that is "smoother" than $f(x_i)$, i.e. not as close to y_i.

4 Experiments

In our experiments, we have used both the Gaussian kernel

$$G_\sigma(x, y) = e^{-\frac{||x-y||^2}{\sigma^2}}$$

and the spectral clustering adaptive kernel (Ng, Jordan and Weiss, 2002), obtained from G_σ by the normalization

$$H_\sigma(x_i, x_j) = \frac{G_\sigma(x_i, x_j)}{\sqrt{a_i a_j}} \tag{8}$$

with the a_i defined as in eq. 3 with $K = G_\sigma$. As we will see, this kernel tends to yield better results than the fixed Gaussian kernel.

We denote by $SS(\lambda, K)$ the algorithm that consists in minimizing the criterion $C_{K,D*,D*,\lambda}$ of eq. 1. The algorithm proposed in (Zhu, Ghahramani and Lafferty, 2003) can thus be written $SS(+\infty, G_\sigma)$. Similarly, we note $SS'(\alpha, \sigma)$ the algorithm that minimizes the regularized version of the criterion of eq. 1, with the kernel G_σ, $\lambda = \frac{1}{2}(1-\alpha)\alpha^{-1}$ $(0 < \alpha < 1)$, $R(f) = \lambda \sum_{i \in U} f(x_i)^2$, $D' = D^*$, and the distance D defined in eq. 2: this is the algorithm from (Zhou et al., 2004).

4.1 Toy data

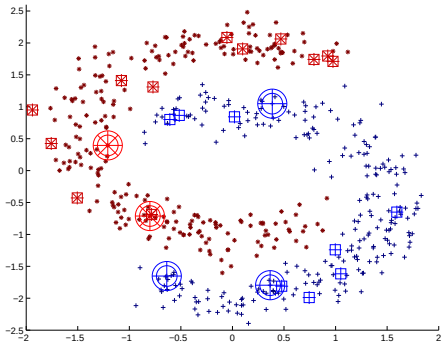

Figure 1: *Toy example of the classification obtained from* 5 *labeled examples (in circles), and* 495 *unlabeled. In squares are a few test points, whose class (symbol or color) is obtained by the induction formula of eq. 7.*

We first validate our generalization formula on a toy dataset, the classical two-moon problem. Figure 1 shows an example where 500 unlabeled examples are classified from only 5 labeled ones. Equation 7 is then used to predict the class of 10000 new test examples sampled from the same density, yielding an error rate of $1,76\%$. Note that, obviously, any supervised algorithm taking into account only the labeled examples would have achieved a much worse performance.

This error rate is similar to the one obtained when we add each test point x to the training set, then minimize the cost of eq. 1 to get the class of x, repeating this operation for each x in the test set: by doing so, we obtain a $1,7\%$ classification error. In both cases, we use the same learning algorithm $SS(\lambda = 5, K = G_{0.1})$.

4.2 Handwritten character recognition

We evaluate our method on real data, using the Letters dataset of the UCI Machine Learning repository. There are 26 handwritten characters classes, to be discriminated using 16 geometric features. However, in our experiments, we reduced to a binary problem by considering only the class formed by the characters 'I' and 'O' and the one formed by 'J' and 'Q' (to make the problem harder than a simple two-character classification). This yields a dataset of 3038 samples, that we divide into a train set $\mathcal{D}$ and a test set $\mathcal{T}$, with respectively 2038 and 1000 samples.

4.2.1 Generalization performance

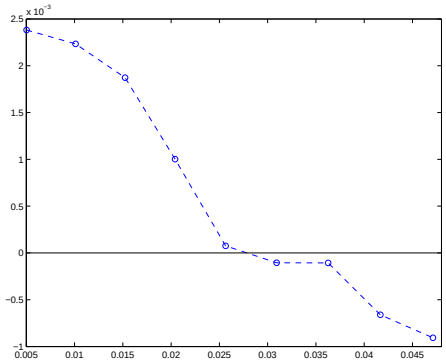

Figure 2: *The horizontal axis is the fraction of substituted data in the training set, which induces a variance in the transductive predictions. The vertical axis gives the difference (δ in eq. 9) between the average difference between the transductive and inductive predictions and the variance in transductive predictions. The graph shows that around 3% perturbation in the training set induces as much variance in the predictions as the difference between transductive and inductive predictions.*

To assess the validity of our generalization formula, we compare $\tilde{f}(x)$ given by eq. 7 with its value $f^*(x)$ that would have been found if x had been included in the training set. We also study the intrinsic stability of $f^*(x)$ when samples from the training set are substituted: this gives us an idea of the uncertainty around $f^*(x)$.

More precisely, the following experiment is made. We take V = the first m samples from our training set $\mathcal{D}$, and split the remaining $n-m$ samples randomly into two sets of equal size, V_1 and V_2. We train our model on $V \cup V_1$, and use eq. 7 to compute $\tilde{f}(x)$ for all $x \in \mathcal{T}$. We also compute $f_1^*(x)$ by minimizing the criterion (eq. 1) over $V \cup V_1 \cup \{x\}$, and $f_2^*(x)$ by doing it over $V \cup V_2 \cup \{x\}$. We then compare the difference between f and f_1^* to the difference between f_1^* and f_2^*:

$$\delta = \frac{1}{|T|} \sum_{x \in T} \left((\tilde{f}(x) - f_1^*(x))^2 - (f_1^*(x) - f_2^*(x))^2 \right). \tag{9}$$

We average δ over 10 random splits of $\mathcal{D} \backslash V$ for V_1 and V_2, which yields the value plotted on figure 2 for different sizes of V. It appears the average error we make by not including x in the training set is of the same order of magnitude as the perturbation induced by a small substitution of about 3% of the training samples. Here, the algorithm $SS(\lambda = 1, K = G_1)$ is used, and the first 10% examples of our dataset $\mathcal{D}$ are labeled.

4.2.2 Comparison with existing algorithms

Here, we compare the classification performance of various algorithms on the unlabeled part U of the dataset $\mathcal{D}$, when we vary the fraction p_l of labeled data L in $\mathcal{D}$. The algorithms tested are $SS(\lambda = +\infty, K = G_\sigma)$, from (Zhu, Ghahramani and Lafferty,

2003), $SS'(\alpha, \sigma)$, from (Zhou et al., 2004), an alternative $SS(\lambda = +\infty, K = H_\sigma)$ derived from our general framework, and, as a baseline, a Parzen windows classifier trained only the labeled examples, using a Gaussian kernel G_σ (algorithm written $PW(\sigma)$ in the figures).

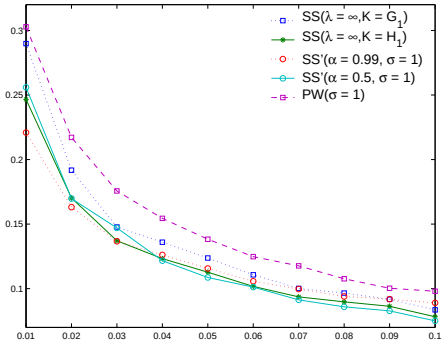

Figure 3: *Classification error on the unlabeled training data for* $0.01 \leq p_l \leq 0.1$.

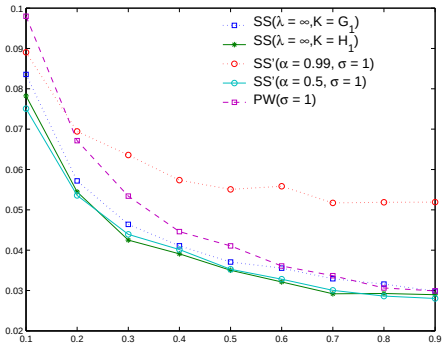

Figure 4: *Classification error on the unlabeled training data for* $0.1 \leq p_l \leq 0.9$.

Throughout the experiments, we keep $\sigma = 1$, which seems to be an adequate value for this dataset. We compute the classification error of each of the algorithms on 100 random shuffles of the whole dataset $\mathcal{D} \cup \mathcal{T}$ (thus using each time a different $\mathcal{D}$ and $\mathcal{T}$, while keeping their size fixed). The average error is plotted on figure 3 for the fraction of labeled samples $p_l \leq 0.1$, and on figure 4 for $p_l \geq 0.1$. We do not show results of the algorithm SS for $\lambda < +\infty$ because they are, on average, worse than those obtained with an infinite λ (for this dataset). Note however that, when using $K = G_\sigma$, for a given $\mathcal{D}$ and $\mathcal{T}$, a lower λ may sometimes give better results when p_l is low (5% or less). This is probably because when less labels are available, the smoothness of the solution becomes more important.

The best overall results are obtained by $SS(\lambda = +\infty, K = H_\sigma)$ and $SS'(\alpha = 0.5, \sigma)$. The reason why they outperform $SS(\lambda = +\infty, K = G_\sigma)$ probably resides in

the normalization of eq. 8 (though SS' does not use exactly the same normalization, but a very close one given by eq. 2). This normalization tends to give higher weights to isolated points, which enforces the smoothness of the function in the regions where the data is sparser, compared to the unnormalized kernel G_σ. As illustrated on the plots, this property has a very positive effect on the overall performance of the algorithm.

Note that, for $p_l \leq 3\%$, $SS'(\alpha = 0.99)$ yields the best performance, but this value of α also penalizes the algorithm for higher values of p_l. The same phenomenon was pointed out in (Zhou et al., 2004), where it was noted that $SS'(\alpha = 0.99)$ was less efficient than $SS(\lambda = +\infty, K = G_\sigma)$ when p_l grew. If we remember that a value of α close to 1 means a low λ in eq. 1, this is explained by the fact the given label values are not necessarily preserved when λ is low. As for the good results obtained by $SS'(\alpha = 0.99)$ for a very low p_l, they can be explained again by the importance of the smoothness of the solution when very few labels are given. Indeed, in SS', the regularization term $\lambda \sum_{i \in U} f(x_i)^2$ encourages this smoothness, and thus yields a better classification than SS, where there is no such regularization.

5 Efficient Approximation

5.1 Algorithm

A simple way to reduce the cubic computational requirement and quadratic memory requirement for 'training' the above non-parametric semi-supervised algorithms is to force the solutions to be expressed in terms of a **subset of the examples**. This idea has already been exploited successfully in a different form for other kernel algorithms, in particular for Gaussian processes (Williams and Seeger, 2001).

Here we will take advantage of the induction formula (eq. 7) to simplify the linear system to $m < n$ equations and variables, where m is the size of a subset of examples that will form a basis for expressing all the other function values. Let $S \subset L \cup U$ with $L \subset S$ be such a subset, with $|S| = m$. Define $R = U \backslash S$. The idea is thus to force $f(x_i)$ for $i \in R$ to be expressed as a linear combination of the $f(x_j)$ with $j \in S$:

$$\forall i \in R, f(x_i) = \frac{\sum_{j \in S} K(x_i, x_j) f(x_j)}{\sum_{j \in S} K(x_i, x_j)}.$$

Plugging this formula in eq. 1 (using the simple squared difference for D and D'), the total cost can be separated in four terms, $C_{LL} + C_{SS} + C_{RR} + C_{RS}$:

- the labeled data error:

$$C_{LL} = \lambda \sum_{i \in L} (f(x_i) - y_i)^2$$

- the smoothness within the selected subset S:

$$C_{SS} = \frac{1}{2} \sum_{i,j \in S} K(x_i, x_j) \left(f(x_i) - f(x_j)\right)^2$$

- the smoothness within the rest of the unlabeled examples R:

$$C_{RR} = \frac{1}{2} \sum_{i,j \in R} K(x_i, x_j) \left(f(x_i) - f(x_j)\right)^2$$

- the two cross-terms between elements of S and elements of R:

$$C_{RS} = 2\frac{1}{2} \sum_{i \in R, j \in S} K(x_i, x_j) \left(f(x_i) - f(x_j)\right)^2 .$$

Let $\vec{f}$ denote now the vector with entries $f(x_i)$, only for $i \in S$ (they are the values to identify). To simplify the notation, also define W the matrix with entries $K(x_i, x_j)$, and sub-matrices W_{SS} when $i, j \in S$, W_{RS} when $i \in R$ and $j \in S$, W_{RR} when $i, j \in R$. Define $\overline{W}$ the matrix with entries $\frac{W_{ij}}{\sum_{k \in S} W_{ik}}$, and the corresponding submatrix $\overline{W}_{RS}$ with entries (i, j) such that $i \in R$ and $j \in S$.

Using this notation, the gradients with respect to the above cost components can be written as follows:

$$\frac{\partial C_{LL}}{\partial \vec{f}} = 2\lambda \Delta_L (\vec{f} - y)$$

where Δ_L is the same as in eq. 5, except it is of size $(m \times m)$, and y is a vector with elements y_i for $i \in L$ and arbitrary elsewhere (these values are multiplied by zeros in Δ_L). The other gradients are as follows:

$$\frac{\partial C_{RR}}{\partial \vec{f}} = \left[2 \left(\overline{W}'_{RS} \left(Diag(W_{RR}\mathbf{1}_r) - W_{RR}\right) \overline{W}_{RS}\right)\right] \vec{f}$$

$$\frac{\partial C_{RS}}{\partial \vec{f}} = \left[2 \left(Diag(W_{SR}\mathbf{1}_r) - \overline{W}'_{RS} W_{RS}\right)\right] \vec{f}$$

$$\frac{\partial C_{SS}}{\partial \vec{f}} = \left[2 \left(Diag(W_{SS}\mathbf{1}_m) - W_{SS}\right)\right] \vec{f}$$

The linear system to be solved can thus be written:

$$A\vec{f} = b$$

with

$$\begin{aligned} A = \lambda \Delta_L \quad &+ \quad \overline{W}'_{RS} \left(Diag(W_{RR}\mathbf{1}_r) - W_{RR}\right) \overline{W}_{RS} \\ &+ \quad Diag(W_{SR}\mathbf{1}_r) - \overline{W}'_{RS} W_{RS} \\ &+ \quad Diag(W_{SS}\mathbf{1}_m) - W_{SS} \end{aligned}$$

and b defined as before in eq. 6 (but with size m). We denote the above algorithm $\widehat{SS}_{RR}$ because it is an approximation based on a subset vs subset linear system (the RR comes from the use of C_{RR}, which will be discussed below).

5.2 Variants

Further improvement can be obtained by observing that the main computational cost comes from $\frac{\partial C_{RR}}{\partial \vec{f}}$. If we choose to ignore C_{RR} in the total cost, then the matrix A can be computed in $O(m^2(n-m))$ time, using only $O(m^2)$ memory (instead of respectively $O(m(n-m)^2)$ time and $O(m(n-m))$ memory when using C_{RR}.). Of course, by doing so we lessen the smoothness constraint on f, since we do not take into account the part of the cost enforcing the smoothness on the rest R of the unlabeled set. However, this may have a beneficial effect when the dataset is large. Indeed, the costs C_{RS} and C_{RR} can be seen as regularizers encouraging the smoothness of $\vec{f}$. In particular, when R is very large, the regularization induced by C_{RR} (containing $(n-m)^2$ terms) can constrain $\vec{f}$ too much, thus penalizing the classification performance. In this case, discarding C_{RR}, in addition to speeding up the computation significantly, also yields better results. We denote the variant of $\widehat{SS}_{RR}$ that does not take C_{RR} into account $\widehat{SS}$.

Training with only a subset S of size $m \ll n$, however, usually won't perform as well as training with the full unlabeled set (even when including C_{RR} and C_{RS}). In order to get closer to this "optimal" performance, one may use the following ensemble method:

- Choose k disjoint subsets $(S'_j)_{1 \leq j \leq k}$ of size $m - |L|$ in U
- For each $j \leq k$, get $f_k(x_i)$ for $i \in U$ by running the above algorithm with the subset $S_j = L \cup S'_j$ and the rest $R_j = U \backslash S_j$ (with or without taking into account C_{RR})
- $\forall i \in U$, take $f(x_i) = \frac{1}{k} \sum_{j=1}^{k} f_k(x_i)$.

As seen in the experiments, this procedure can help reducing the classification error, on the unlabeled training examples as well as on new test data (using the induction formula). Note that even though the computational time requirements are multiplied by a factor k, they may be taken back to their original value if one can perform parallel computation, in order to train simultaneously on the different subsets. We denote by $\overline{SS}_{k,RR}$ and $\overline{SS}_k$ the above algorithm, with and without the C_{RR} cost.

In table 1, we summarize the time and memory requirements of the various algorithms presented in this paper. The approximation methods described in this section improve the computation time and memory consumption by a factor approximately n/m for $\widehat{SS}_{RR}$ and $(n/m)^2$ for $\widehat{SS}$, and with parallelization, the same improvements can be maintained while averaging the results of k experiments on k different subsets.

5.3 Experiments

Here, we apply the $\overline{SS}_{k,RR}$ and $\overline{SS}_k$ algorithms described above to the handwritten character recognition problem of section 4.2. We take $k = 10$, and for each fraction p_l of labeled examples, we take 10 subsets S'_j so that $U = \bigcup_{1 \leq j \leq 10} S'_j$. For comparison purpose, we denote by SS_k the algorithm that consists in running SS on each $L \cup S'_j$, then averaging the outputs of the resulting regressors when computing the output for a

Table 1: *Comparative computational requirements of the original algorithm SS (or SS'), the approximation $\widehat{SS}_{RR}$ using C_{RR}, its variant $\widehat{SS}$ discarding C_{RR}, and the ensemble variant over k subsets, with or without C_{RR} in the cost, denoted respectively by $\overline{SS}_{k,RR}$ and $\overline{SS}_k$. The brackets around the factor k mean it can be taken out if the computation is parallelized.*

	TIME	MEMORY
SS OR SS'	$O(n^3)$	$O(n^2)$
$\widehat{SS}_{RR}$	$O(m(n-m)^2)$	$O(m(n-m))$
$\widehat{SS}$	$O(m^2(n-m))$	$O(m^2)$
$\overline{SS}_{k,RR}$	$O((m(n-m)^2)[k])$	$O(m(n-m))$
$\overline{SS}_k$	$O((m^2(n-m))[k])$	$O(m^2)$

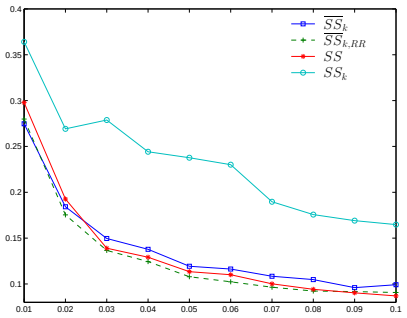

Figure 5: *Test set classifi cation error, for* $0.01 \leq p_l \leq 0.1$.

new point x. We compare the performance of $\overline{SS}_{k,RR}$ and $\overline{SS}_k$ with (i) the algorithm SS using the whole unlabeled set U, and (ii) the algorithm SS_k. This last algorithm is shown to demonstrate the gain induced by incorporating C_{RS} (and possibly C_{RR}) in the cost optimized. Figures 5 (for low p_l) and 6 (for high p_l) show the average classification error on the test set (the average is done over 10 fixed (train,test) pairs $(\mathcal{D}_i, \mathcal{T}_i)_{1 \leq i \leq 10}$ of respectively 2038 and 1000 characters, taken randomly from the original dataset). We take $K = G_1$, and $\lambda = +\infty$. Note that plotting the error on the unlabeled part of the training set would have given similar results. Also, the average errors of $\widehat{SS}_{RR}$ and $\widehat{SS}$ (the approximation algorithms without averaging), not shown on the figures, are slightly higher than the ones of respectively $\overline{SS}_{k,RR}$ and $\overline{SS}_k$, but still (far) below the one of SS_k.

Interestingly, $\overline{SS}_{k,RR}$ and $\overline{SS}_k$ can outperform SS trained on the whole set U when p_l is low. This is coherent with our previous observations, which showed that encouraging the smoothness of the solution for such p_l could improve the robustness of the classifier.

Note that here, the dataset is still rather small, and including C_{RR} in the cost systematically helps. However, other experiments performed on the same handwritten

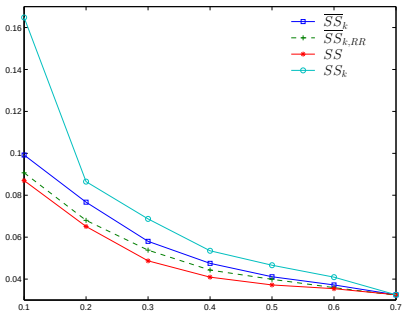

Figure 6: *Test set classifi cation error, for* $0.1 \leq p \leq 0.7$.

character database, but using a larger dataset (10000 training and test examples) have given better results with $\widehat{SS}$ than with $\widehat{SS}_{RR}$ (both still doing better than simply training over the subset S, discarding completely the rest R).

6 Conclusion

The main contribution of this paper is an extension of previously proposed non-parametric (graph-based) semi-supervised learning algorithms, that allows one to efficiently perform function induction (i.e. cheaply compute a prediction for a new example, in time $O(n)$ instead of $O(n^3)$). The extension is justified by the minimization of the same smoothness criterion that was used to obtain the original algorithms in the first place. We showed that the function induction formula is robust and yields predictions that are close to the transductive (expensive) predictions (in the sense that the difference is of the same order as the average change in transductive prediction when a small fraction of the training examples are substituted by others from the same distribution).

The paper compares empirically on real data a variety of semi-supervised learning algorithms that fit under this framework, helping to understand the effect of the different components of the training criterion.

Finally, the induction formula is used to define several approximation variants (either for transduction or for function induction) that yield important reductions in computational and memory complexity at a small cost in classification performance.

References

Belkin, M. and Niyogi, P. (2003). Using manifold structure for partially labeled classification. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, Cambridge, MA. MIT Press. 2, 5

Chapelle, O., Weston, J., and Scholkopf, B. (2003). Cluster kernels for semi-supervised learning. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, Cambridge, MA. MIT Press. 2

Cozman, F., Cohen, I., and Cirelo, M. (2003). Semi-supervised learning of mixture models. In *ICML'2003*. 2

Kemp, C., Griffiths, T., Stromsten, S., and Tenembaum, J. (2004). Semi-supervised learning with trees. In *Advances in Neural Information Processing Systems 16*, volume 16, Cambridge, MA. MIT Press. 2

McCallum, A. and Nigam, K. (1998). Employing EM and pool-based active learning for text classification. In *ICML'1998*. 2

Nadaraya, E. (1964). On estimating regression. *Theory of Probability and its Applications*, 9:141–142. 5

Ng, A. Y., Jordan, M. I., and Weiss, Y. (2002). On spectral clustering: analysis and an algorithm. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press. 5

Schuurmans, D. and Southey, F. (2002). Metric-based methods for adaptive model selection and regularization. *Machine Learning*, 48(1):51–84. 2

Seeger, M. (2001). Learning with labeled and unlabeled data. Technical report, Edinburgh University. 1

Szummer, M. and Jaakkola, T. (2002). Partially labeled classification with markov random walks. In Dietterich, T., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press. 2

Watson, G. (1964). Smooth regression analysis. *Sankhya - The Indian Journal of Statistics*, 26:359–372. 5

Williams, C. K. I. and Seeger, M. (2001). Using the Nyström method to speed up kernel machines. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems 13*, pages 682–688, Cambridge, MA. MIT Press. 9

Zhou, D., Bousquet, O., Navin Lal, T., Weston, J., and Schölkopf, B. (2004). Learning with local and global consistency. In Thrun, S., Saul, L., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA. MIT Press. 2, 3, 4, 6, 8, 9

Zhu, X., Ghahramani, Z., and Lafferty, J. (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *ICML'2003*. 2, 3, 4, 6, 8

Zhu, X., Lafferty, J., and Ghahramani, Z. (2003). Semi-supervised learning: From gaussian fields to gaussian processes. Technical Report CMU-CS-03-175, CMU. 2