Efficient Non-Parametric Function Induction in Semi-Supervised Learning

Olivier Delalleau, Yoshua Bengio and Nicolas Le Roux
Dept. IRO, Université de Montréal
P.O. Box 6128, Succ. Centre-Ville, Montreal, H3C 3J7, Qc, Canada
{delallea,bengioy,lerouxni}@iro.umontreal.ca

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. First, it extends them to function induction algorithms that minimize a regularization criterion applied to an out-of-sample example, and happen to have the form of Parzen windows regressors. This allows to predict test labels without solving again a linear system of dimension n (the number of unlabeled and labeled training examples), which can cost $O(n^3)$. Second, this function induction procedure gives rise to an efficient approximation of the training process, reducing the linear system to be solved to $m \ll n$ unknowns, using only a subset of m examples. An improvement of $O(n^2/m^2)$ in time can thus be obtained. Comparative experiments are presented, showing the good performance of the induction formula and approximation algorithm.

1 INTRODUCTION

Several non-parametric approaches to semi-supervised learning (see (Seeger, 2001) for a review of semi-supervised learning) have been recently introduced, e.g. in (Szummer & Jaakkola, 2002; Chapelle et al., 2003; Belkin & Niyogi, 2003; Zhu et al., 2003a; Zhu et al., 2003b; Zhou et al., 2004). They rely on weak implicit assumptions on the generating data distribution, e.g. smoothness of the target function with respect to a given notion of similarity between examples[1]. For classification tasks this amounts to assuming that the target function is constant within the region of input space (or "cluster" (Chapelle et al., 2003)) associated with a particular class. 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 between two nodes are associated with the value of a similarity function applied on the corresponding two 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 general they have been designed for the transductive setting, in which 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 data. 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. In (Zhu et al., 2003b) it is proposed to assign to the test case the label (or inferred label) of the nearest neighbor (NN) from the training set (labeled or unlabeled). In this paper we derive from the training criterion an inductive formula that turns out to have the form of a Parzen windows predictor, for a computational cost that is $O(n)$. Besides being smoother than the NN-algorithm, this induction formula is consistent with the predicted labels on the unlabeled training data.

In addition to providing a relatively cheap way of 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, 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 m variables and equations.

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

2 NON-PARAMETRIC SMOOTHNESS CRITERION

In the mathematical formulations, we only consider here the case of binary classification. Each labeled example x_k $(1 \leq k \leq l)$ is associated with a label $y_k \in \{-1, 1\}$, and we turn the classification task into a regression one by looking for the values of a function f on both labeled and unlabeled examples x_i $(1 \leq i \leq n)$, such that $f(x_i) \in [-1, 1]$. The predicted class of x_i is thus $sign(f(x_i))$. Note however that all algorithms proposed extend naturally to multiclass problems, using the usual one vs. rest trick.

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 training examples x_i:

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

where U is the unlabeled set, L the labeled set, x_i the i-th example, y_i the target label for $i \in L$, $W(\cdot, \cdot)$ 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. Three methods using a criterion of this form have already been proposed: (Zhu et al., 2003a), (Zhou et al., 2004) (where an additional regularization term is added to the cost, equal to $\lambda \sum_{i \in U} f(x_i)^2$), and (Belkin et al., 2004) (where for the purpose of theoretical analysis, they add the constraint $\sum_i f(x_i) = 0$). To obtain a quadratic form in $f(x_i)$ one typically chooses D and D' to be quadratic, e.g. the Euclidean distance. This criterion can then 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 $W(\cdot, \cdot)$ is sparse.

A quadratic dissimilarity function makes a lot of sense in regression problems but has also been used successfully in classification problems, by looking for a *continuous* labeling function f. The first term of eq. 1 indeed enforces the smoothness of f. The second term makes f consistent with the given labels. The hyperparameter λ controls the trade-off between those two costs. 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 et al., 2003a) consider forcing $f(x_i) = y_i$, which corresponds to $\lambda = +\infty$).

In the following we study the case where D and D' are the Euclidean distance. We also assume samples are sorted so that $L = \{1, \ldots, l\}$ and $U = \{l + 1, \ldots, n\}$. The minimization of the criterion w.r.t. all the $f(x_i)$ for $i \in L \cup U$ then gives rise to the linear system

$$A\vec{f} = \lambda \vec{y} \tag{2}$$

with

$$\vec{y} = (y_1, \ldots, y_l, 0, \ldots, 0)^T \tag{3}$$

$$\vec{f} = (f(x_1), \ldots, f(x_n))^T$$

and, using the matrix notation $W_{ij} = W(x_i, x_j)$, the matrix A written as follows:

$$A = \lambda \Delta_L + Diag(W\mathbf{1}_n) - W \tag{4}$$

where $Diag(v)$ is the matrix whose diagonal is the vector v, $\mathbf{1}_n$ is the vector of n ones, and Δ_L $(n \times n)$ is

$$(\Delta_L)_{ij} = \delta_{ij} \delta_{i \in L}. \tag{5}$$

This solution has the disadvantage of providing no obvious prediction for new examples, but the method is generally used transductively (the test examples are included in the unlabeled set). To obtain function induction without having to solve the linear system for each new test point, one alternative would be to parameterize f with a flexible form such as a neural network or a linear combination of non-linear bases (see also (Belkin & Niyogi, 2003)). Another is the induction formula proposed below.

3 FUNCTION INDUCTION FORMULA

In order to transform the above transductive algorithms into function induction algorithms we will do two things: (i) consider the same type of smoothness criterion as in eq. 1, but including a test example x, and (ii) as in ordinary function induction (by opposition to transduction), require that the value of $f(x_i)$ on training examples x_i remain fixed even after x has been added[2].

The second point is motivated by the prohibitive cost of solving again the linear system, and the reasonable assumption that the value of the function over the unlabeled examples will not change much with the addition of a new point. This is clearly true asymptotically (when $n \to \infty$). In the non-asymptotic case we should expect transduction to perform better than induction (again, assuming test samples are drawn from the same distribution as the training data), but as shown in our experiments, the loss is typically very small, and comparable to the variability due to the selection of training examples.

Adding terms for a new unlabeled point x in eq. 1 and keeping the value of f fixed on the training points x_j

[2] Here we assume x to be drawn from the same distribution as the training samples: if it is not the case, this provides another justification for keeping the $f(x_i)$ fixed.

leads to the minimization of the modified criterion

$$C^*_{W,D}(f(x)) = \sum_{j \in U \cup L} W(x, x_j) D(f(x), f(x_j)). \quad (6)$$

Taking for D the usual Euclidean distance, $C^*_{W,D}$ is convex in $f(x)$ and is minimized when

$$f(x) = \frac{\sum_{j \in U \cup L} W(x, x_j) f(x_j)}{\sum_{j \in U \cup L} W(x, x_j)} = \tilde{f}(x). \quad (7)$$

Interestingly, this is exactly the formula for Parzen windows or Nadaraya-Watson non-parametric regression (Nadaraya, 1964; Watson, 1964) when W is the Gaussian kernel and the estimated $f(x_i)$ on the training set are considered as desired values.

One may want to see what happens when we apply $\tilde{f}$ on a point x_i of the training set. For $i \in U$, we obtain that $\tilde{f}(x_i) = f(x_i)$. But for $i \in L$,

$$\tilde{f}(x_i) \quad = \quad f(x_i) + \frac{\lambda(f(x_i) - y_i)}{\sum_{j \in U \cup L} W(x_i, x_j)}.$$

Thus the induction formula (eq. 7) gives the same result as the transduction formula (implicitely defined by eq. 2) over unlabeled points, but on labeled examples it chooses a value that is "smoother" than $f(x_i)$ (not as close to y_i). This may lead to classification errors on the labeled set, but generalization error may improve by allowing non-zero training error on labeled samples. This remark is also valid in the special case where $\lambda = +\infty$, where we fix $f(x_i) = y_i$ for $i \in L$, because the value of $\tilde{f}(x_i)$ given by eq. 7 may be different from y_i (though experiments showed such label changes were very unlikely in practice).

The proposed algorithm for semi-supervised learning is summarized in algorithm 1, where we use eq. 2 for training and eq. 7 for testing.

Algorithm 1 Semi-supervised induction

(1) Training phase
Compute $A = \lambda\Delta_L + Diag(W\mathbf{1}_n) - W$ (eq. 4)
Solve the linear system $A\vec{f} = \lambda\vec{y}$ (eq. 2) to obtain $f(x_i) = \vec{f}_i$
(2) Testing phase
For a new point x, compute its label $\tilde{f}(x)$ by eq. 7

4 SPEEDING UP THE TRAINING PHASE

A simple way to reduce the cubic computational requirement and quadratic memory requirement for 'training' the non-parametric semi-supervised algorithms of section 2 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, e.g. for Gaussian processes (Williams & Seeger, 2001).

Here we will take advantage of the induction formula (eq. 7) to simplify the linear system to $m \ll n$ 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 to force $f(x_i)$ for $i \in R$ to be expressed as a linear combination of the $f(x_j)$ with $j \in S$:

$$\forall i \in R, f(x_i) = \frac{\sum_{j \in S} W(x_i, x_j) f(x_j)}{\sum_{j \in S} W(x_i, x_j)}. \quad (8)$$

Plugging this in eq. 1, we separate the cost in four terms $(C_{RR}, C_{RS}, C_{SS}, C_L)$:

$$\begin{aligned}
& \underbrace{\frac{1}{2} \sum_{i,j \in R} W(x_i, x_j) \left(f(x_i) - f(x_j)\right)^2}_{C_{RR}} \\
+ \quad & \underbrace{2 \times \frac{1}{2} \sum_{i \in R, j \in S} W(x_i, x_j) \left(f(x_i) - f(x_j)\right)^2}_{C_{RS}} \\
+ \quad & \underbrace{\frac{1}{2} \sum_{i,j \in S} W(x_i, x_j) \left(f(x_i) - f(x_j)\right)^2}_{C_{SS}} \\
+ \quad & \underbrace{\lambda \sum_{i \in L} (f(x_i) - y_i)^2}_{C_L}
\end{aligned}$$

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 notations, decompose W in the following sub-matrices:

$$W = \begin{pmatrix} W_{SS} & W'_{RS} \\ W_{RS} & W_{RR} \end{pmatrix}.$$

with W_{SS} of size $(m \times m)$, W_{RS} of size $((n-m) \times m)$ and W_{RR} of size $((n-m) \times (n-m))$. Also define $\overline{W}_{RS}$ the matrix of size $((n-m) \times m)$ with entries $\frac{W_{ij}}{\sum_{k \in S} W_{ik}}$, for $i \in R$ and $j \in S$.

Using these notations, the gradient of the above cost with respect to $\vec{f}$ can be written as follows:

$$\begin{aligned}
& \underbrace{\left[2\left(\overline{W}^T_{RS} \left(Diag(W_{RR}\mathbf{1}_r) - W_{RR}\right) \overline{W}_{RS}\right)\right] \vec{f}}_{\frac{\partial C_{RR}}{\partial \vec{f}}} \\
+ \quad & \underbrace{\left[2\left(Diag(W_{SR}\mathbf{1}_r) - \overline{W}^T_{RS} W_{RS}\right)\right] \vec{f}}_{\frac{\partial C_{RS}}{\partial \vec{f}}} \\
+ \quad & \underbrace{\left[2\left(Diag(W_{SS}\mathbf{1}_m) - W_{SS}\right)\right] \vec{f}}_{\frac{\partial C_{SS}}{\partial \vec{f}}} + \underbrace{2\lambda\Delta_L(\vec{f} - \vec{y})}_{\frac{\partial C_L}{\partial \vec{f}}}
\end{aligned}$$

where Δ_L is the same as in eq. 5, but is of size $(m \times m)$, and $\vec{y}$ is the vector of targets (eq. 3), of size m. The linear system $A\vec{f} = \lambda\vec{y}$ of eq. 2 is thus redefined with the following system matrix:

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

The main computational cost now comes from the computation of $\frac{\partial C_{RR}}{\partial \vec{f}}$. To avoid it, we simply choose to ignore C_{RR} in the total cost, so that 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 keeping C_{RR}. By doing so we lessen the smoothness constraint on f, since we do not take into account the part of the cost enforcing smoothness between the examples in R. However, this may have a beneficial effect. Indeed, the costs C_{RS} and C_{RR} can be seen as regularizers encouraging the smoothness of f on R. In particular, using C_{RR} may induce strong constraints on f that could be inappropriate when the approximation of eq. 8 is inexact (which especially happens when a point in R is far from all examples in S). This could constrain f too much, thus penalizing the classification performance. In this case, discarding C_{RR}, besides yielding a significant speed-up, also gives better results. Algorithm 2 summarizes this algorithm (not using C_{RR}).

Algorithm 2 Fast semi-supervised induction

Choose a subset $S \supseteq L$ (e.g. with algorithm 3)
$R \leftarrow U \setminus S$
(1) Training phase

$$\begin{aligned} A \leftarrow \quad & \lambda\Delta_L + Diag(W_{SR}\mathbf{1}_r) \\ - \quad & \overline{W}_{RS}^T W_{RS} + Diag(W_{SS}\mathbf{1}_m) - W_{SS} \end{aligned}$$

Solve the linear system $A\vec{f} = \lambda\vec{y}$ to obtain $f(x_i) = \vec{f}_i$ for $i \in S$
Use eq. 8 to obtain $f(x_i)$ for $i \in R$
(2) Testing phase
For a new point x, compute its label $\tilde{f}(x)$ by eq. 7

In general, training using only a subset of $m \ll n$ samples will not perform as well as using the whole dataset. Thus, it can be important to choose the examples in the subset carefully to get better results than a random selection. Our criterion to choose those examples is based on eq. 8, that shows $f(x_i)$ for $i \notin S$ should be well approximated by the value of f at the neighbors of x_i in S (the notion of neighborhood being defined by W). Thus, in particular, x_i for $i \notin S$ should not be too far from the examples in S. This is also important because when discarding the part C_{RR} of the cost, we must be careful to cover the whole manifold with S, or we may leave "gaps" where the smoothness of f would not be enforced. This suggests to start with $S = \emptyset$ and $R = U$, then add samples x_i iteratively by choosing the point farthest from the current subset, i.e. the one that minimizes $\sum_{j \in L \cup S} W(x_i, x_j)$. Note that adding a sample that is far from all other examples in the dataset will not help, thus we discard an added point if this is the case (x_j being "far" is defined by a threshold on $\sum_{i \in R \setminus \{j\}} W(x_i, x_j)$). In the worst case, this could make the algorithm in $O(n^2)$, but assuming only few examples are far from all others, it scales as $O(mn)$. Once this first subset is selected, we refine it by training the algorithm presented in section 2 on the subset S, in order to get an approximation of the $f(x_i)$ for $i \in S$, and by using the induction formula of section 3 (eq. 7) to get an approximation of the $\tilde{f}(x_j)$ for $j \in R$. We then discard samples in S for which the confidence in their labels is high[3], and replace them with samples in R for which the confidence is low (samples near the decision surface). One should be careful when removing samples, though: we make sure we do not leave "empty" regions (i.e. $\sum_{i \in L \cup S} W(x_i, x_j)$ must stay above some threshold for all $j \in R$). Finally, labeled samples are added to S. Overall, the cost of this selection phase is on the order of $O(mn + m^3)$. Experiments showing its effectiveness are presented in section 5.3. The subset selection algorithm[4] is summarized in algorithm 3.

5 EXPERIMENTS

5.1 FUNCTION INDUCTION

Here, we want to validate our induction formula (eq. 7): the goal is to show that it gives results close to what would have been obtained if the test points had been included in the training set (transduction). Indeed, we expect that the more unlabeled points the better, but how much better? Experiments have been performed on the "Letter Image Recognition" dataset of the UCI Machine Learning repository (UCI MLR). There are 26 handwritten characters classes, to be discriminated using 16 geometric features. However, to keep things simple, we reduce to a binary problem by considering only the class formed by the characters 'I' and 'O' and the class formed by 'J' and 'Q' (the choice of these letters makes the problem harder than a basic two-character classification task). This yields a dataset of 3038 samples. We use for $W(x, y)$ the Gaussian kernel with bandwidth 1: $W(x, y) = e^{-||x-y||^2}$.

First, we analyze how the labels can vary between in-

[3] In a binary classification task, the confidence is given by $|f(x_i)|$. In the multi-class case, it is the difference between the weights of the two classes with highest weights.

[4] Note that it would be interesting to investigate the use of such an algorithm in cases where one can obtain labels, but at a cost, and needs to select which samples to label.

Algorithm 3 Subset selection

δ is a small threshold, e.g. $\delta = 10^{-10}$
(1) Greedy selection
$S \leftarrow \emptyset$ {The subset we are going to build}
$R \leftarrow U$ {The rest of the unlabeled data}
while $|S| + |L| < m$ **do**
  Find $j \in R$ s.t. $\sum_{i \in R \setminus \{j\}} W(x_i, x_j) \geq \delta$ and $\sum_{i \in L \cup S} W(x_i, x_j)$ is minimum
  $S \leftarrow S \cup \{j\}$
  $R \leftarrow R \setminus \{j\}$
(2) Improving the decision surface
Compute an approximate of $f(x_i)$, $i \in S$ and $\tilde{f}(x_j)$, $j \in R$, by applying algorithm 1 with the labeled set L and the unlabeled set S and using eq. 7 on R
$S_H \leftarrow$ the points in S with highest confidence (see footnote 3)
$R_L \leftarrow$ the points in R with lowest confidence
for all $j \in S_H$ **do**
  if $\min_{i \in R} \sum_{k \in L \cup S \setminus \{j\}} W(x_i, x_k) \geq \delta$ **then**
    $k^* \leftarrow \text{argmin}_{k \in R_L} \sum_{i \in L \cup S} W(x_k, x_i)$
    $S \leftarrow (S \setminus \{j\}) \cup \{k^*\}$ {Replace j by k^* in S}
    $R \leftarrow (R \setminus \{k^*\}) \cup \{j\}$ {Replace k^* by j in R}
$S \leftarrow S \cup L$ {Add the labeled data to the subset}

duction and transduction when the test set is large (section 5.1.1), then we study how this variation compares to the intrinsic variability due to the choice of training data (section 5.1.2).

5.1.1 Induction vs. Transduction

When we add new points to the training set, two questions arise. First, do the $f(x_i)$ change significantly? Second, how important is the difference between induction and transduction over a large amount of new points, in terms of classification performance?

The experiments shown in fig. 1 have been made considering three training sets, $T1000$, $T2000$ and $T3038$, containing respectively 1000, 2000 and 3038 samples (the results plotted are averaged on 10 runs with randomly selected $T1000$ and $T2000$). The first two curves show the percentage of unlabeled data in $T1000$ and $T2000$ whose label has changed compared to the labels obtained when training over $T3038$ (the whole dataset). This validates our hypothesis that the $f(x_i)$ do not change much when adding new training points.

The next three curves show the classification error for the unlabeled data respectively on $T3038 \setminus T2000$, $T3038 \setminus T1000$ and $T3038 \setminus T1000$, for the algorithm trained respectively on $T2000$, $T1000$ and $T3038$. This allows us to see that the induction's performance is close to that of transduction (the average relative increase in classification error compared to transduction is about 20% for $T1000$ and 10% for $T2000$). In addi-

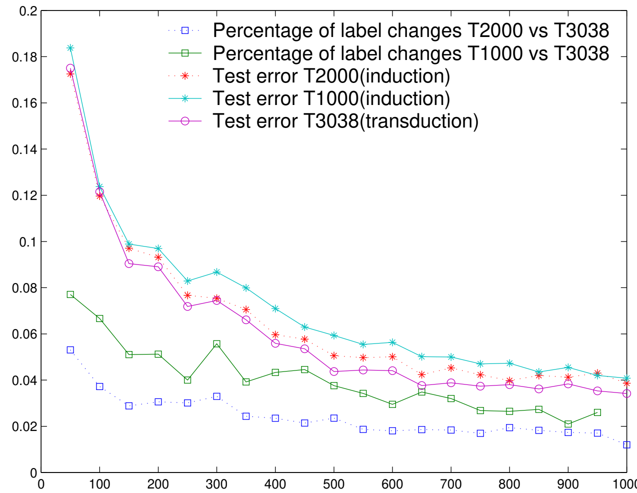

Figure 1: Percentage of unlabeled training data whose label has changed when test points were added to the training set, and classification error in induction and transduction. Horizontal axis: number of labeled data.

tion, the difference is very small for large amounts of labeled data as well as for very small amounts. This can be explained in the first case by the fact that enough information is available in the labeled data to get close to optimal classification, and in the second case, that there are too few labeled data to ensure an efficient prediction, either transductive or inductive.

5.1.2 Varying The Test Set Size

The previous experiments have shown that when the test set is large in comparison with the training set, the induction formula will not be as efficient as transduction. It is thus interesting to see how evolves the difference between induction and transduction as the test set size varies in proportion with the training set size. In particular, for which size of the test set is that difference comparable to the sensitivity of the algorithm with respect to the choice of training set?

To answer this question, we need a large enough dataset to be able to choose random training sets. The whole Letters dataset is thus used here, and the binary classification problem is to discriminate the letters 'A' to 'M' from the letters 'N' to 'Z'. We take a fixed test set of size 1000. We repeat 10 times the experiments that consists in: (i) choosing a random base training set of 2000 samples (with 10% labeled), and (ii) computing the average error on test points in *transduction* by adding a fraction of them to this base training set and solving the linear system (eq. 2), repeating this so as to compute the error on all test points.

The results are shown in fig. 2, when we vary the number of test points added to the training set. Adding 0 test points is slightly different, since it corresponds to

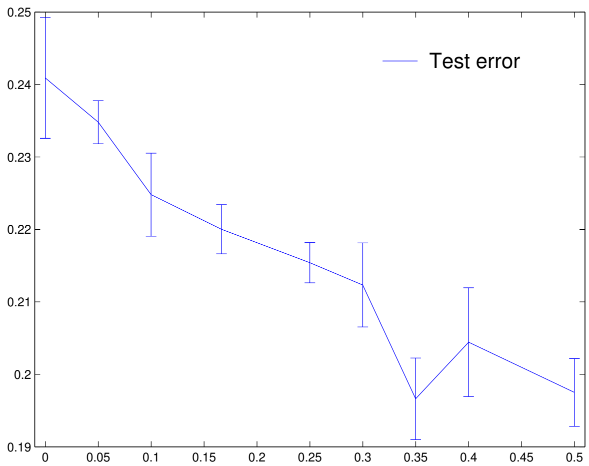

Figure 2: Horizontal axis: number of test points added in proportion with the size of the training set. Vertical axis: test error (in transduction for a proportion > 0, and in induction for the proportion 0).

the induction setting, that we plot here for comparison purpose. We see that adding a fraction of test examples corresponding to less than 5% of the training set does not yield a significant decrease in the test error compared to induction, given the intrinsic variability due to the choice of training set. It could be interesting to compare induction with the limit case where we add only 1 test point at step (ii). We did not do it because of the computational costs, but one would expect the difference with induction to be smaller than for the 5% fraction.

5.2 COMPARISON WITH EXISTING ALGORITHM

We compare our proposed algorithm (alg. 1) to the semi-supervised *Laplacian* algorithm from (Belkin & Niyogi, 2003), for which classification accuracy on the MNIST database of handwritten digits is available. Benchmarking our induction algorithm against the *Laplacian* algorithm is interesting because the latter does not fall into the general framework of section 2.

In order to obtain the best performance, a few refinements are necessary. First, it is better to use a sparse weighting function, which allows to get rid of the noise introduced by far-away examples, and also makes computations faster. The simplest way to do this is to combine the original weighting function (the Gaussian kernel) with k-nearest-neighbors. We define a new weighting function W_k by $W_k(x_i, x_j) = W(x_i, x_j)$ if x_i is a k-nearest-neighbor of x_j or vice-versa, and 0 otherwise. Second, W_k is normalized as in Spectral Clustering (Ng et al., 2002), i.e.

$$\overline{W}_k(x_i, x_j) = \frac{W_k(x_i, x_j)}{\sqrt{\frac{1}{n}\sum_{r\neq i} W_k(x_i, x_r)\sum_{r\neq j} W_k(x_r, x_k)}}.$$

Table 1: Comparative Classification Error of the *Laplacian* Algorithm (Belkin & Niyogi, 2003), $WholeSet$ in Transduction and $WholeSet$ in Induction on the MNIST Database. On the horizontal axis is the number of labeled examples and we use two different sizes of training sets (1000 and 10000 examples).

Labeled	**50**	**100**	**500**	**1000**	**5000**
Total: **1000**					
Laplacian	29.3	19.6	11.5		
$WholeSet_{trans}$	25.4	17.3	9.5		
$WholeSet_{ind}$	26.3	18.8	11.3		
Total: **10000**					
Laplacian	25.5	10.7	6.2	5.7	4.2
$WholeSet_{trans}$	25.1	11.3	5.3	5.2	3.5
$WholeSet_{ind}$	25.1	11.3	5.7	5.1	4.2

Finally, the dataset is systematically preprocessed by a Principal Component Analysis on the training part (labeled and unlabeled), to further reduce noise in the data (we keep the first 45 principal components).

Results are presented in table 1. Hyperparameters (number of nearest neighbors and kernel bandwidth) were optimized on a validation set of 20000 samples, while the experiments were done on the rest (40000 samples). The classification error is averaged over 100 runs, where the train (1000 or 10000 samples) and test sets (5000 samples) are randomly selected among those 40000 samples. Standard errors (not shown) are all under 2% of the error. The *Laplacian* algorithm was used in a transductive setting, while we separate the results for our algorithm into $WholeSet_{trans}$ (error on the training data) and $WholeSet_{ind}$ (error on the test data, obtained thanks to the induction formula)[5]. On average, both $WholeSet_{trans}$ and $WholeSet_{ind}$ slightly outperform the *Laplacian* algorithm.

5.3 APPROXIMATION ALGORITHMS

The aim of this section is to compare the classification performance of various algorithms:

- $WholeSet$, the original algorithm presented in sections 2 and 3, where we make use of all unlabeled training data (same as $WholeSet_{ind}$ in the previous section),
- $RSub_{subOnly}$, the algorithm that consists in speeding-up training by using only a random subset of the unlabeled training samples (the rest is completely discarded),
- $RSub_{RR}$ and $RSub_{noRR}$, the approximation algorithms described in section 4, when the subset is selected randomly (the second algorithm discards the part C_{RR} of the cost for faster training),

[5]See section 5.3 for the origin of the name $WholeSet$.

Table 2: Comparative Computational Requirements (n = number of training data, m = subset size)

	Time	**Memory**
$WholeSet$	$O(n^3)$	$O(n^2)$
$RSub_{subOnly}$	$O(m^3)$	$O(m^2)$
$RSub_{RR}$ $SSub_{RR}$	$O(m(n-m)^2)$	$O(m(n-m))$
$RSub_{noRR}$ $SSub_{noRR}$	$O(m^2(n-m))$	$O(m^2)$

- $SSub_{RR}$ and $SSub_{noRR}$, which are similar to those above, except that the subset is now selected as in algorithm 3.

Table 2 summarizes time and memory requirements for these algorithms: in particular, the approximation method described in section 4, when we discard the part C_{RR} of the cost ($RSub_{noRR}$ and $SSub_{noRR}$), improves the computation time and memory usage by a factor approximately $(n/m)^2$.
The classification performance of these algorithms was compared on three multi-class problems: **LETTERS** is the "Letter Image Recognition" dataset from the UCI MLR. (26 classes, dimension 16), **MNIST** contains the first 20000 samples of the MNIST database of handwritten digits (10 classes, dimension 784), and **COVTYPE** contains the first 20000 samples of the normalized[6] "Forest CoverType" dataset from the UCI MLR. (7 classes, dimension 54).

We repeat 50 times the experiment that consists in choosing randomly 10000 samples as training data and the rest as the test set, and computing the test error (using the induction formula) for the different algorithms. The average classification error on the test set (with standard error) is presented in table 3 for $WholeSet$, $RSub_{subOnly}$, $RSub_{noRR}$ and $SSub_{noRR}$. For each dataset, results for a labeled fraction of 1%, 5% and 10% of the training data are presented. In algorithms using only a subset of the unlabeled data (i.e. all but $WholeSet$), the subset contains only 10% of the unlabeled set. Hyperparameters have been roughly estimated and remain fixed on each dataset. In particular, λ (in eq. 1) is set to 100 for all datasets, and the bandwidth of the Gaussian kernel used is set to 1 for LETTERS, 1.4 for MNIST and 1.5 for COVTYPE.

The approximation algorithms using the part C_{RR} of the cost ($RSub_{RR}$ and $SSub_{RR}$) are not shown in the results, because it turns out that using C_{RR} does not necessarily improve the classification accuracy, as argued in section 4. Additionally, discarding C_{RR} makes training significantly faster. Compared to $WholeSet$, typical training times with these specific settings show that $RSub_{subOnly}$ is about 150 times faster, $RSub_{noRR}$ about 15 times, and $SSub_{noRR}$ about 10 times. Note however that these factors increase very fast with the size of the dataset (10000 samples is still "small").

[6] Scaled so that each feature has standard deviation 1.

Table 3: Comparative Classification Error (Induction) of $WholeSet$, $RSub_{subOnly}$, $RSub_{noRR}$ and $SSub_{noRR}$, for Various Fractions of Labeled Data.

% labeled	LETTERS	MNIST	COVTYPE
1%			
$WholeSet$	56.0 ± 0.4	35.8 ± 1.0	47.3 ± 1.1
$RSub_{subOnly}$	59.8 ± 0.3	29.6 ± 0.4	$\mathbf{44.8} \pm 0.4$
$RSub_{noRR}$	57.4 ± 0.4	27.7 ± 0.6	75.7 ± 2.5
$SSub_{noRR}$	$\mathbf{55.8} \pm 0.3$	$\mathbf{24.4} \pm 0.3$	45.0 ± 0.4
5%			
$WholeSet$	$\mathbf{27.1} \pm 0.4$	12.8 ± 0.2	37.1 ± 0.2
$RSub_{subOnly}$	32.1 ± 0.2	14.9 ± 0.1	$\mathbf{35.4} \pm 0.2$
$RSub_{noRR}$	29.1 ± 0.2	12.6 ± 0.1	70.6 ± 3.2
$SSub_{noRR}$	28.5 ± 0.2	$\mathbf{12.3} \pm 0.1$	35.8 ± 0.2
10%			
$WholeSet$	$\mathbf{18.8} \pm 0.3$	$\mathbf{9.5} \pm 0.1$	34.7 ± 0.1
$RSub_{subOnly}$	22.5 ± 0.1	11.4 ± 0.1	$\mathbf{32.4} \pm 0.1$
$RSub_{noRR}$	20.3 ± 0.1	9.7 ± 0.1	64.7 ± 3.6
$SSub_{noRR}$	19.8 ± 0.1	$\mathbf{9.5} \pm 0.1$	33.4 ± 0.1

The first observation that can be made from table 3 is that $SSub_{noRR}$ consistently outperforms (or does about the same as) $RSub_{noRR}$, which validates our subset selection step (alg. 3). However, rather surprisingly, $RSub_{subOnly}$ can yield better performance than $WholeSet$ (on MNIST for 1% of labeled data, and systematically on COVTYPE): adding more unlabeled data actually harms the classification accuracy. There may be various reasons to this, the first one being that hyperparameters should be optimized separatly for each algorithm to get their best performance. In addition, for high-dimensional data without obvious clusters or low-dimensional representation, it is known that the inter-points distances tend to be all the same and meaningless (see e.g. (Beyer et al., 1999)). Thus, using a Gaussian kernel will force us to consider rather large neighborhoods, which prevents a sensible propagation of labels through the data during training. Nevertheless, a constatation that arises from those results is that $SSub_{noRR}$ never "breaks down", being always either the best or close to the best. It is able to take advantage of all the unlabeled data, while focussing the computations on a well chosen subset. The importance of the subset selection is made clear with COVTYPE, where choosing a random subset can be catastrophic: this is probably because the approximation made in eq. 8 is very poor for some of the points which are not in the subset, due to the low structure in the data.

Note that the goal here is not to obtain the best performance, but to compare the effectiveness of those algorithms under the same experimental settings. Indeed, further refinements of the weighting function (see section 5.2) can greatly improve classification accuracy.

Additional experiments were performed to asses the superiority of our subset selection algorithm over random selection. In the following, unless specified otherwise, datasets come from the UCI MLR, and were preprocessed with standard normalization. The kernel bandwidth was approximately chosen to optimize the performance of $RSub_{noRR}$, and λ was arbitrarily set to 100. The experiments consist in taking as training set 67% of the available data, 10% of which are labeled, and using the subset approximation methods $RSub_{noRR}$ and $SSub_{noRR}$ with a subset of size 10% of the available unlabeled training data. The classification error is then computed on the rest of the data (test set), and averaged over 50 runs. On average, on the 8 datasets tested, $SSub_{noRR}$ always gives better performance. The improvement was *not* found to be statistically significative for the following datasets: Mushroom (8124 examples $\times$ 21 variables), Statlog Landsat Satellite (6435×36) and Nursery (12960×8). $SSub_{noRR}$ performs significantly better than $RSub_{noRR}$ (with a relative decrease in classification error from 4.5 to 12%) on: Image (2310×19), Isolet (7797×617), PenDigits (10992×16), SpamBase (4601×57) and the USPS dataset (9298×256, not from UCI). Overall, our experiments show that random selection can sometimes be efficient enough (especially with large low-dimensional datasets), but smart subset selection is to be preferred, since it (almost always) gives better and more stable results.

6 CONCLUSION

The first 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 used to obtain the original algorithms in the first place.

The second contribution is the use of this induction formula to define new optimization algorithms speeding up the training phase. Those new algorithms are based on using of a small subset of the unlabeled data, while still keeping information from the rest of the available samples. This subset can be heuristically chosen to improve classification performance over random selection. Such algorithms yield important reductions in computational and memory complexity and, combined with the induction formula, they give predictions close to the (expensive) transductive predictions.

References

Belkin, M., Matveeva, I., & Niyogi, P. (2004). Regularization and semi-supervised learning on large graphs. *COLT'2004*. Springer.

Belkin, M., & Niyogi, P. (2003). Using manifold structure for partially labeled classification. *Advances in Neural Information Processing Systems 15*. Cambridge, MA: MIT Press.

Beyer, K. S., Goldstein, J., Ramakrishnan, R., & Shaft, U. (1999). When is "nearest neighbor" meaningful? *Proceeding of the 7th International Conference on Database Theory* (pp. 217–235). Springer-Verlag.

Chapelle, O., Weston, J., & Scholkopf, B. (2003). Cluster kernels for semi-supervised learning. *Advances in Neural Information Processing Systems 15*. Cambridge, MA: MIT Press.

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

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

Ng, A. Y., Jordan, M. I., & Weiss, Y. (2002). On spectral clustering: analysis and an algorithm. *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press.

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

Szummer, M., & Jaakkola, T. (2002). Partially labeled classification with markov random walks. *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press.

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

Williams, C. K. I., & Seeger, M. (2001). Using the Nyström method to speed up kernel machines. *Advances in Neural Information Processing Systems 13* (pp. 682–688). Cambridge, MA: MIT Press.

Zhou, D., Bousquet, O., Navin Lal, T., Weston, J., & Schölkopf, B. (2004). Learning with local and global consistency. *Advances in Neural Information Processing Systems 16*. Cambridge, MA: MIT Press.

Zhu, X., Ghahramani, Z., & Lafferty, J. (2003a). Semi-supervised learning using gaussian fields and harmonic functions. *ICML'2003*.

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