

Olivier Delalleau
Yoshua Bengio
Nicolas Le Roux

In Chapter 11, it is shown how a number of graph-based semi-supervised learning algorithms can be seen as the minimization of a specific cost function, leading to a linear system with n equations and unknowns (with n the total number of labeled and unlabeled examples). Solving such a linear system will in general require on the order of $O(kn^2)$ time and $O(kn)$ memory (for a sparse graph where each data point has k neighbors), which can be prohibitive on large datasets (especially if $k = n$, i.e. the graph is dense). We present in this chapter a subset selection method that can be used to reduce the original system to one of size $m \ll n$. The idea is to solve for the labels of a subset $S \subset X$ of only m points, while still retaining information from the rest of the data by approximating their label with a linear combination of the labels in S (using the induction formula presented in Chapter 11). This leads to an algorithm whose computational requirements scale as $O(m^2n)$ and memory requirements as $O(m^2)$, thus allowing one to take advantage of significantly bigger unlabeled datasets than with the original algorithms.

18.1 Introduction

The graph-based semi-supervised algorithms presented in Chapter 11 do not scale well to very large datasets. In this chapter, we propose an approximation method that significantly reduces the computational and memory requirements of such algorithms. Notations will be the same as in Chapter 11, i.e:

- $Y = (Y_l, Y_u)$ is the set of “original” labels on labeled and unlabeled points (here, Y_u is filled with 0),
- $\hat{Y} = (\hat{Y}_l, \hat{Y}_u)$ is the set of estimated labels on labeled and unlabeled points,
- \hat{y} is the function to learn, which assigns a label to each point of the input space,

- $\hat{y}(x_i) = \hat{y}_i$ is the value of the function \hat{y} on training points (labeled and unlabeled).

In Chapter 11, we defined a quadratic cost (equation 11.11):

$$C(\hat{Y}) = \|\hat{Y}_l - Y_l\|^2 + \mu \hat{Y}^\top L \hat{Y} + \mu \epsilon \|\hat{Y}\|^2. \quad (18.1)$$

Minimizing this cost gives rise to the following linear system in \hat{Y} with regularization hyper-parameters μ and ϵ :

$$(\mathbf{S} + \mu L + \mu \epsilon \mathbf{I}) \hat{Y} = \mathbf{S} Y \quad (18.2)$$

where \mathbf{S} is the $(n \times n)$ diagonal matrix defined by $\mathbf{S}_{ii} = \delta_{i \leq l}$, and $L = \mathbf{D} - \mathbf{W}$ is the un-normalized graph Laplacian. This linear system can be solved to obtain the value of \hat{y}_i on the training points x_i . We can extend the formula to obtain the value of \hat{y} on every point x in the input space as shown in Section 11.4:

$$\hat{y} = \frac{\sum_j W_X(x, x_j) \hat{y}_j}{\sum_j W_X(x, x_j) + \epsilon} \quad (18.3)$$

where W_X is the symmetric data-dependent edge weighting function (e.g. a Gaussian kernel) such that $\mathbf{W}_{ij} = W_X(x_i, x_j)$. However, in case of very large training sets, solving the linear system (18.2) may be computationally prohibitive, even using iterative techniques such as those described in Section 11.2. In this chapter we consider how to approximate the cost using only a subset of the examples. Even though this will not yield an exact solution to the original problem, it will make the computation time much more reasonable.

18.2 Cost Approximations

18.2.1 Estimating the Cost from a Subset

reduced
parametrization
of solution

A simple way to reduce the $O(kn^2)$ computational requirement and $O(kn)$ memory requirement for training the non-parametric semi-supervised algorithms of Chapter 11 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 and Seeger [2001]) or spectral embedding algorithms (Ouibet and Bengio [2005]).

Here we will take advantage of the induction formula (eq. 18.3) 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 $S \subset \{1, \dots, n\}$ be a subset, with $|S| = m$ and $S \supset \{1, \dots, l\}$ (i.e. we take all labeled examples in the subset). Define $R = \{1, \dots, n\} \setminus S$ (the rest of the data). In the following, vector and matrices will be split into their S and R parts, e.g.

$$\hat{Y} = (\hat{Y}_S, \hat{Y}_R) \text{ and}$$

$$L = \begin{pmatrix} L_{SS} & L_{SR} \\ L_{RS} & L_{RR} \end{pmatrix}.$$

The idea is to force $\hat{y}_i \in \hat{Y}_R$ to be expressed as a linear combination of the $\hat{y}_j \in \hat{Y}_S$ following (18.3):

$$\forall i \in R, \hat{y}_i = \frac{\sum_{j \in S} \mathbf{W}_{ij} \hat{y}_j}{\sum_{j \in S} \mathbf{W}_{ij} + \epsilon} \quad (18.4)$$

or in matrix notation

$$\hat{Y}_R = \overline{\mathbf{W}}_{RS} \hat{Y}_S \quad (18.5)$$

with $\overline{\mathbf{W}}_{RS}$ the matrix of size $((n-m) \times m)$ with entries $\mathbf{W}_{ij}/(\epsilon + \sum_{k \in S} \mathbf{W}_{ik})$, for $i \in R$ and $j \in S$. We will then split the cost (18.1) in terms that involve only the subset S or the rest R , or both of them. To do so, we must first split the diagonal matrix \mathbf{D} (whose elements are row sums of \mathbf{W}) into $\mathbf{D} = \mathbf{D}^S + \mathbf{D}^R$, with \mathbf{D}^S and \mathbf{D}^R the $(n \times n)$ diagonal matrices whose elements are sums over S and R respectively, i.e.

$$\begin{aligned} \mathbf{D}_{ii}^S &= \sum_{j \in S} \mathbf{W}_{ij} \\ \mathbf{D}_{ii}^R &= \sum_{j \in R} \mathbf{W}_{ij}. \end{aligned}$$

The un-normalized Laplacian $L = \mathbf{D} - \mathbf{W}$ can then be written

$$L = \begin{pmatrix} \mathbf{D}_{SS}^S + \mathbf{D}_{SS}^R - \mathbf{W}_{SS} & -\mathbf{W}_{SR} \\ -\mathbf{W}_{RS} & \mathbf{D}_{RR}^S + \mathbf{D}_{RR}^R - \mathbf{W}_{RR} \end{pmatrix}. \quad (18.6)$$

Using (18.6), the cost (18.1) can now be expanded as follows:

$$\begin{aligned} C(\hat{Y}) &= \mu \hat{Y}^\top L \hat{Y} + \mu \epsilon \|\hat{Y}\|^2 + \|\hat{Y}_I - Y_I\|^2 \\ &= \underbrace{\mu \hat{Y}_S^\top (\mathbf{D}_{SS}^S - \mathbf{W}_{SS}) \hat{Y}_S + \mu \epsilon \|\hat{Y}_S\|^2}_{C_{SS}} + \underbrace{\mu \hat{Y}_R^\top (\mathbf{D}_{RR}^R - \mathbf{W}_{RR}) \hat{Y}_R + \mu \epsilon \|\hat{Y}_R\|^2}_{C_{RR}} \\ &\quad + \underbrace{\mu (\hat{Y}_S^\top \mathbf{D}_{SS}^R \hat{Y}_S + \hat{Y}_R^\top \mathbf{D}_{RR}^S \hat{Y}_R - \hat{Y}_R^\top \mathbf{W}_{RS} \hat{Y}_S - \hat{Y}_S^\top \mathbf{W}_{SR} \hat{Y}_R)}_{C_{RS}} \\ &\quad + \underbrace{\|\hat{Y}_I - Y_I\|^2}_{C_L} \end{aligned} \quad (18.7)$$

18.2.2 Resolution

Using the approximation $\hat{Y}_R = \overline{\mathbf{W}}_{RS} \hat{Y}_S$ (18.5), the gradient of the different parts of the above cost with respect to \hat{Y}_S is then

$$\begin{aligned}
\frac{\partial C_{SS}}{\partial \hat{Y}_S} &= [2\mu (\mathbf{D}_{SS}^S - \mathbf{W}_{SS} + \epsilon \mathbf{I})] \hat{Y}_S \\
\frac{\partial C_{RR}}{\partial \hat{Y}_S} &= [2\mu \overline{\mathbf{W}}_{RS}^\top (\mathbf{D}_{RR}^R - \mathbf{W}_{RR} + \epsilon \mathbf{I}) \overline{\mathbf{W}}_{RS}] \hat{Y}_S \\
\frac{\partial C_{RS}}{\partial \hat{Y}_S} &= [2\mu (\mathbf{D}_{SS}^R + \overline{\mathbf{W}}_{RS}^\top \mathbf{D}_{RR}^S \overline{\mathbf{W}}_{RS} - \overline{\mathbf{W}}_{RS}^\top \mathbf{W}_{RS} - \mathbf{W}_{SR} \overline{\mathbf{W}}_{RS})] \hat{Y}_S \\
&= [2\mu (\mathbf{D}_{SS}^R - \mathbf{W}_{SR} \overline{\mathbf{W}}_{RS})] \hat{Y}_S \\
\frac{\partial C_L}{\partial \hat{Y}_S} &= 2\mathbf{S}_{SS}(\hat{Y}_S - Y)
\end{aligned} \tag{18.8}$$

where to obtain (18.8) we have used the equality $\mathbf{D}_{RR}^S \overline{\mathbf{W}}_{RS} = \mathbf{W}_{RS}$, which follows from the definition of $\overline{\mathbf{W}}_{RS}$.

Recall the original linear system in \hat{Y} was $(\mathbf{S} + \mu L + \mu \epsilon \mathbf{I}) \hat{Y} = \mathbf{S}Y$ (18.2). Here it is replaced by a new system in \hat{Y}_S , written $\mathbf{A} \hat{Y}_S = \mathbf{S}_{SS} Y_S$ with

$$\begin{aligned}
\mathbf{A} = & \mu (\mathbf{D}_{SS}^S - \mathbf{W}_{SS} + \epsilon \mathbf{I} + \mathbf{D}_{SS}^R - \mathbf{W}_{SR} \overline{\mathbf{W}}_{RS}) \\
& + \mu \overline{\mathbf{W}}_{RS}^\top (\mathbf{D}_{RR}^R - \mathbf{W}_{RR} + \epsilon \mathbf{I}) \overline{\mathbf{W}}_{RS} \\
& + \mathbf{S}_{SS}.
\end{aligned}$$

Since the system's size has been reduced from n to $|S| = m$, it can be solved much faster, even if \mathbf{A} is not guaranteed¹ to be sparse anymore (we assume $m \ll n$).

Unfortunately, in order to obtain the matrix \mathbf{A} , we need to compute \mathbf{D}_{RR}^R , which costs $O(n^2)$ in time, as well as products of matrices that cost $O(mn^2)$ if \mathbf{W} is not sparse. A simple way to get rid of the quadratic complexity in n is to ignore C_{RR} in the total cost. If we remember that C_{RR} can be written

simplified cost function

$$C_{RR} = \mu \left(\frac{1}{2} \sum_{i,j \in R} \mathbf{W}_{ij} (\hat{y}_i - \hat{y}_j)^2 + \epsilon \|\hat{Y}_R\|^2 \right)$$

this corresponds to ignoring the Smoothness Assumption between points in R , as well as the regularization term on R . Even if it may look like a bad idea, it turns out it usually preserves (and even improves) the performance of the semi-supervised classifier, for various reasons:

- assuming the subset S is chosen to correctly “fill” the space, smoothness between points in S and points in R (encouraged by the part C_{RS} of the cost) also enforces smoothness between points in R only,

1. In practice, if \mathbf{W} is sparse, \mathbf{A} is also likely to be sparse, even if additional assumptions on \mathbf{W} are needed if one wants to prove it.

- when reducing to a subset, the loss in capacity (we can choose m values instead of n when working with the full set) suggests we should weaken regularization, and the smoothness constraints are a form of regularization, thus dropping some of them is a way to achieve this goal,
- for some points $i \in R$, the approximation (18.4)

$$\hat{y}_i = \frac{\sum_{j \in S} \mathbf{W}_{ij} \hat{y}_j}{\sum_{j \in S} \mathbf{W}_{ij} + \epsilon}$$

may be poor (e.g. for a point far from all points in S , i.e. $\sum_{j \in S} \mathbf{W}_{ij}$ very small), thus smoothness constraints between points in R could be noisy and detrimental to the optimization process (this is not a big issue when considering smoothness between a point x_i in R and a point x_j in S as the smoothness penalty is weighted by \mathbf{W}_{ij} , which will be small if x_i is far from all points in S).

Given the above considerations, ignoring the part C_{RR} leads to the new system

$$(\mathbf{S}_{SS} + \mu (\mathbf{D}_{SS} - \mathbf{W}_{SS} - \mathbf{W}_{SR} \overline{\mathbf{W}}_{RS} + \epsilon \mathbf{I})) \hat{\mathbf{Y}}_S = \mathbf{S}_{SS} \mathbf{Y}_S$$

which in general can be solved in $O(m^3)$ time (less if the system matrix is sparse).

18.3 Subset Selection

18.3.1 Random Selection

In general, training using only a subset of $m \ll n$ samples will not perform as well as using the whole dataset. Carefully choosing the subset S can help in limiting this loss in performance. Even if random selection is certainly the easiest way to choose the points in S , it has two main drawbacks:

- It may not pick points in some regions of the space, resulting in the approximation (18.4) being very poor in these regions.
- It may pick uninteresting points: the region near the decision surface is the one where we are more likely to make mistakes by assigning the wrong label. Therefore, we would like to have as many points as possible in S being in that region, while we do not need points which are far away from that surface.

As a result, it is worthwhile considering more elaborate subset selection schemes, such as the one presented in the next section.

18.3.2 Smart Data Sampling

There could be many ways of choosing which points to take in the subset. The algorithm described below is one solution, based on the previous considerations about the random selection weaknesses. The first step of the algorithm will be

to select points somewhat uniformly in order to get a first estimate of the decision surface, while the second step will consist in the choice of points near that estimated surface.

18.3.2.1 First step

Equation (18.4)

$$\hat{y}_i = \frac{\sum_{j \in S} \mathbf{W}_{ij} \hat{y}_j}{\sum_{j \in S} \mathbf{W}_{ij} + \epsilon}$$

covering the manifold

suggests that the value of \hat{y}_i is well approximated when there is a point in S near x_i (two points x_i and x_j are nearby if \mathbf{W}_{ij} is high). The idea will therefore be to cover the manifold where the data lie as well as possible, that is to say ensure that every point in R is near a point (or a set of points) in S . There is another issue we should be taking care of: as we discard the part C_{RR} of the cost, we must now be careful not to modify the structure of the manifold. If there are some parts of the manifold without any point of S , then the smoothness of \hat{y} will not be enforced at such parts (and the labels will be poorly estimated).

avoiding outliers

This suggests to start with $S = \{1, \dots, l\}$ and $R = \{l + 1, \dots, n\}$, then add samples x_i by iteratively choosing the point farthest from the current subset, i.e. the one that minimizes $\sum_{j \in S} \mathbf{W}_{ij}$. The idea behind this method is that it is useless to have two points nearby each other in S , as this will not give extra information while increasing the cost. However, one can note that this method may tend to select outliers, which are far from all other points (and especially those from S). A way to avoid this is to consider the quantity $\sum_{j \in R \setminus \{i\}} \mathbf{W}_{ij}$ for a given x_i . If x_i is such an outlier, this quantity will be very low (as all \mathbf{W}_{ij} are small). Thus, if it is smaller than a given threshold δ , we do not take x_i in the subset. The cost of this additional check is of $O((m + o)n)$ where o is the number of outliers: assuming there are only a few of them (less than m), it scales as $O(mn)$.

18.3.2.2 Second step

discarding uninformative samples

Once this first subset is selected, it can be refined by training the algorithm presented in Section 11.3.2 on the subset S , in order to get an approximation of the \hat{y}_i for $i \in S$, and by using the induction formula (18.4) to get an approximation of the \hat{y}_j for $j \in R$. Samples in S which are far away from the estimated decision surface can then be discarded, as they will be correctly classified no matter whether they belong to S or not, and they are unlikely to give any information on the shape of the decision surface. These discarded samples are then replaced by other samples that are near the decision surface, in order to be able to estimate it more accurately.

The distance from a point x_i to the decision surface is estimated by the confidence we have in its estimated label \hat{y}_i . In the binary classification case considered here (with targets -1 and 1), this confidence is given by $|\hat{y}_i|$, while in a multi-class setting it would be the absolute value of the difference between the predicted scores

of the two highest-scoring classes. One should be careful when removing samples, though: we must make sure we do not leave “empty” regions. This can be done by ensuring that $\sum_{j \in S} \mathbf{W}_{ij}$ stays above some threshold for all $i \in R$ after a point has been removed.

Overall, the cost of this selection phase is on the order of $O(mn + m^3)$. It is summarized in Algorithm 18.1.

Algorithm 18.1 Subset selection

Choose a small threshold δ (e.g. $\delta \leftarrow 10^{-10}$)
 Choose a small regularization parameter ϵ (e.g. $\epsilon \leftarrow 10^{-11}$).
(1) Greedy selection
 $S \leftarrow \{1, \dots, l\}$ {The subset we are going to build contains the labeled points}
 $R \leftarrow \{l + 1, \dots, n\}$ {The rest of the unlabeled points}
while $|S| < m$ **do**
 Find $i \in R$ s.t. $\sum_{j \in R \setminus \{i\}} \mathbf{W}_{ij} \geq \delta$ and $\sum_{j \in S} \mathbf{W}_{ij}$ is minimum
 $S \leftarrow S \cup \{i\}$
 $R \leftarrow R \setminus \{i\}$
end while
(2) Decision surface improvement
 Compute an approximate of \hat{y}_i with $i \in S$ by applying the standard semi-supervised minimization of Section 11.3.2 with the data set S .
 Compute an approximate of \hat{y}_j with $j \in R$ by (18.4)
 $S_H \leftarrow$ the points in S with highest confidence (see Section 18.3.2.2)
 $R_L \leftarrow$ the points in R with lowest confidence
for all $i \in S_H$ **do**
 if $\min_{j \in R} \sum_{k \in S \setminus \{i\}} \mathbf{W}_{jk} \geq \delta$ **then**
 { i can be safely removed from S without leaving empty regions}
 $k^* \leftarrow \operatorname{argmin}_{k \in R_L} \sum_{j \in S} \mathbf{W}_{jk}$ {Find point with low confidence farthest from S }
 Replace i by k^* in S (and k^* by i in R)
 end if
end for

18.3.3 Computational Issues

We are now in position to present the overall computational requirements for the different algorithms proposed in this chapter. As before, the subset size m is taken to be much smaller than the total number of points n , and the weight matrix \mathbf{W} may either be dense or sparse (with k non-zero entries in each row or column). Table 18.1 summarizes time and memory requirements for the following algorithms:

- *NoSub*: the original transductive algorithm (using the whole dataset) that consists in solving the system (18.2), as presented in Chapter 11 (Algorithm 11.2),
- *RandSub*: the approximation algorithm discussed in Section 18.2.2, with the subset S being randomly chosen (Section 18.3.1),

Table 18.1 Comparative computational requirements of *NoSub*, *RandSub* and *SmartSub* (n = number of labeled and unlabeled training data, m = subset size with $m \ll n$, k = number of neighbors for each point in \mathbf{W} when \mathbf{W} is sparse)

	Time	Memory
<i>NoSub</i> (sparse \mathbf{W})	$O(kn^2)$	$O(kn)$
<i>NoSub</i> (dense \mathbf{W})	$O(n^3)$	$O(n^2)$
<i>RandSub</i>	$O(m^2n)$	$O(m^2)$
<i>SmartSub</i>	$O(m^2n)$	$O(m^2)$

- *SmartSub*: the same approximation algorithm as *RandSub*, but with S being chosen as in Section 18.3.2.

The table shows the approximation method described in this chapter is particularly useful when \mathbf{W} is dense or n is very large. This is confirmed by empirical experimentation in Figure 18.1, which compares the training times (on the benchmark dataset **SecStr** described in Chapter 21 of this book) of *NoSub* with a dense kernel, *NoSub* with a sparse kernel, and *SmartSub* with a dense kernel. With a dense kernel, *NoSub* becomes quickly impractical because of the need to store (and solve) a linear system of size $n = l + u$, with $l = 100$ and $u \in [2000, 50000]$. With a sparse kernel (and the iterative version presented in Algorithm 11.2) it scales much better, but still exhibits a quadratic dependency in n . On the other hand, *SmartSub* can handle much more unlabeled data as its training time scales only linearly in n . We have not presented a sparse version of *SmartSub* since our current code cannot take advantage of a sparse weighting function. However, this could be useful to obtain further improvement, especially in terms of memory usage (working with full $m \times m$ matrices can become problematic when $m \geq 10000$).

18.4 Discussion

This chapter follows-up on Chapter 11 to allow large-scale applications of semi-supervised learning algorithms presented previously. The idea is to express the cost to be minimized as a function of only a subset of the unknown labels, in order to reduce the number of free variables: this can be obtained thanks to the induction formula introduced in Chapter 11. The form of this formula suggests it is only accurate when the points in the subset cover the whole manifold on which the data lie. This explains why choosing the subset randomly can lead to poor results, while it is possible to design a simple heuristic algorithm (such as Algorithm 18.1) giving much better classification performance. Better selection algorithms (e.g. explicitly optimizing the cost we are interested in) are subject of future research.

One must note that the idea of expressing the cost from a subset of the data is not equivalent to training a standard algorithm on the subset only, before extending to the rest of the data with the induction formula. Here, the rest of the data is

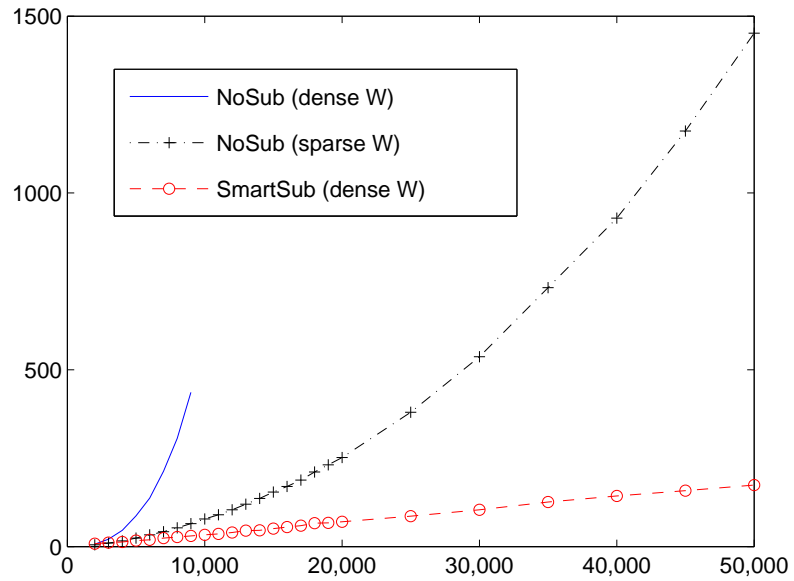


Figure 18.1 Training time (in seconds) w.r.t. the amount of unlabeled samples on benchmark dataset `SecStr` (cf. Chapter 21). W_X is a Gaussian kernel (combined with an approximate 100-nearest-neighbor kernel in the sparse case). There are $l = 100$ labeled samples, and *SmartSub* selects $m = 500$ unlabeled samples in the subset approximation scheme. Note how the dependence of *SmartSub* in the total number of unlabeled samples $u \in [2000, 50000]$ is only linear. *NoSub* with dense \mathbf{W} fails for $u \geq 10000$ because of memory shortage. Experiments were performed on a 3.2 GHz P4 CPU with 2 Gb of RAM.

explicitly used in the part of the cost enforcing the smoothness between points in the subset and points in the rest (part C_{RS} of the cost), which helps to obtain a smoother labeling function, usually giving better generalization.