Semester project 8

Chebyshev polynomial approximation for transductive learning on graphs

Lausanne
Spring 2011

Perraudin Nathanaël

supervised by
Vandergheynst Pierre
and
Shuman David

LTS2 - EPFL

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

1 Introduction

The goal of transductive learning is to find a way to recover the labels of lots of data with only a few known samples. In this work, we will work on graphs for two reasons. First, it's possible to construct a graph from a given dataset with features. The main assumption we make is that if two vertices are close (connected with a small weight), they will have similiar labels. Thus, graph theory allows us to solve lots of problems. Second, graph problems can be solved distributively.

Imagine you have a sensors network with a limited transmission power. Each sensor can only communicate with its closest neighbours. Implementing a distributive algorithm allows you to compute the solution directly on the sensors and on special point. In that case you can with regression, filter the noise, evaluate a data even if a sensor is broken, detect if a sensor is broken and estimate the data to a point with no sensors. This could also be done centrally by a computer, but doing so all the data will require to be transfered to the computer which consume a lot of resources (here energy).

In [1], Zhou et al. present a recursive algorithm to solve the transductive learning problem. This recursive algorithm can also be implemented distributedly. In this work we compare this algorithm with a Chebyshev polynomial approximation, which is presented in [2] and [3]. The main advantage is Chebyshev polynomial are his recursive properties. We will also study the stability of the different algorithms. The criteria to evaluate if an algorithm will be the communication cost, which is the number of messages exchanged between all the vertices.

This work is divided into two main parts. In the first one, we consider general, the graph regression problem. This first part is again divided into 2 subproblems. The first one is a general regression problem with prior $\|\nabla x\|^2$ (where x is the solution). After showing the solution, we talk about Chebyshev polynomial approximation and graph Fourier transform. We give some examples based on the Minnesota road graph. In the second subproblem, we consider ridge regression. We will study the convergence of our algorithm as a function of the basis. We also compare our algorithm with some known recursive ones and we calculate the impact of the sparsity (needed in order to make our algorithm efficient) on the answer. We have worked on different datasets, but mainly on the DELVE Boston housing.

For the second main part, we consider classification. In this case, the answer (the labels) is not continuous anymore, but discrete. In this way, the computer has to take a decision. We show that we can consider this new problem as a one shot filter. Then we talk about the basis used for the graph Fourier transform. We also study how we can construct a graph and some practical cases based on several datasets (the most studied will be USPS). In that part, we compare our Chebyshev approximation to an iterative algorithm.

The link between all the problems is the way we use Chebyshev polynomial approximation. We choose a basis and we make a graph Fourier transform. Then we approximate the transfer function in the Fourier domain. We end with an inverse transform. In fact we show that we can avoid the direct and inverse graph Fourier transform and compute the solution directly or (and) distributively.

1.1 Some theorical definition about graph

A weighted graph $G = E, V, w$ is a set of N vertices V connected by a set of edge E with a the weighted function w. We can define A of size N x N the adjacency matrix like:

$$a_{m,n} = \begin{cases} w(e) & \text{if } e \in E \text{ connects vertices m and n} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Then we define the diagonal matrix D (size N x N) with $d(m) = \sum_n a_{m,n}$. And we define the Laplacian matrix $\mathcal{L}$ (size N x N) :

$$\mathcal{L} = D - A \tag{2}$$

Remark: In fact, in order to define a Laplacian, we need first to define a edge derivative:

$$\frac{\partial f(v)}{\partial e_u} = \sqrt{A(u,v)}(f(u) - f(v)) \tag{3}$$

With this derivative (2) can be shown.

1.1.1 Graph Fourier Transform

Let supposed we have a Laplacian $\mathcal{L}$ with linearly independant eigenvectors ϕ_l $(l = 1, 2, ..., N)$. We define the graph Fourier transform of f:

$$\hat{f}(l) =< \phi_l, f > \tag{4}$$

We can also define the inverse Fourier transform:

$$f(u) = \sum_{l=1}^{N} \hat{f}(l)\phi_l(u) \tag{5}$$

1.2 Some theorical definition about Chebyshev polynomial approximation

Chebychev Polynomials $T_k(y)$ can be generated by a stable recurence relation: $T_k(y) = 2yT_{k-1}(y) - T_{k-2}(y)$ with $T_0 = 1$ and $T_1 = y$. We can evalute a function f with a infinite sum of a Chebyshev serie:

$$f(y) = \frac{1}{2}c_0 + \sum_{k=1}^{\infty} c_k T_k(y) \tag{6}$$

with

$$c_k = \frac{2}{\pi}\int_{-1}^{1} \frac{T_k(z)f(z)}{\sqrt{1-z^2}}dz = \frac{2}{\pi}\int_0^{\pi} \cos(k\theta)h(\cos(\theta))d\theta \tag{7}$$

In order to approximate the function f we can only take the M first terms of the sum given by (6). We get the truncated Chebyshev polynomial approximation:

$$\tilde{f}(y) = \frac{1}{2}c_0 + \sum_{k=1}^{M} c_k T_k(y) \tag{8}$$

2 Regression on graphs

In this part, we are going to consider regression. This mean the solution is going to be continous. We will see two different problem. The first one is going to be general and can be easily extended to other kind of problem (denoising, unfiltering, ...). In the second we will present Ridge regression which is more efficient. But it can only be used on reconstruction.

2.1 One first general problem

We consider a graph with a given adjacency matrix: A of size N x N. Thus, we can easily get the laplacian matrix $\mathcal{L}$ with (2). Some information about the label of vertices is missing. We know only a few label. In order to formulate the problem, we create a mask M which is a diagonal matrix with $m_{n,n} = 1$ if the label is known and $m_{n,n} = 0$ otherwise for $n = 1, 2, ..., N$. Suppose the value of the labels is contain in a vector c of size N, then we construct b like $b = Mc$. We can choose the norm of the gradient as prior (We will see later that this gradient presents very good properties). Thus, we will consider that the signal of the graph is "quite smooth". The goal of this problem is to recover c.

We need to find a solution to this problem:

$$\min_{x \in \mathcal{R}^n} \left(\|b - Mx\|^2 + \mu \|\nabla x\|^2 \right) \tag{9}$$

where x the solution.

In order to find a solution, we are going to divide this problem into two parts.

2.1.1 Splitting the problem

This is a typical problem of minimising the sum of two functions. We can apply the proximal splitting method: forward-backward splitting [4].

If we define:

$$\begin{cases} f_1(x) = \mu\|\nabla x\|^2 & (10) \\ f_2(x) = \|b - Mx\|^2 & (11) \end{cases}$$

The solution of the problem:

$$\min_{x\in\mathcal{R}^n} \big(f_1(x) + f_2(x)\big) \tag{12}$$

is given by:

$$x = prox_{\gamma f_1}(x - \gamma\nabla f_2(x)) \tag{13}$$

where the definition of the prox of f_1 is:

$$prox_{\gamma f_1}(y) = \min_{x\in\mathcal{R}^n} \big(\gamma f_1(x) + \frac{1}{2}\|x - y\|^2\big) \tag{14}$$

This suggests an iterative method.

$$x_{n+1} = prox_{\gamma_n f_1}(x_n - \gamma_n\nabla f_2(x_n)) \tag{15}$$

We can solve such a problem by alternately searching the solution of two new, but easier questions:

$$\begin{cases} y_n = x_n + \gamma_n\nabla f_2(x_n) & (16) \\ x_{n+1} = prox_{\gamma f_1} y_n & (17) \end{cases}$$

2.1.2 Gradient of $\|b - Ax\|^2$

If we calculate the gradient of f_2, we find:

$$\nabla f_2(x) = \nabla\|b - Mx\|^2 = 2M^*(Mx - b) \tag{18}$$

where M^* is the adjoint matrix of M.

Remark: Here $M^* = M$ because M is a simple mask.

Generalisation of our problem: Here the result is quite broad. We can easily switch M with another kind of matrix providing its adjoint is defined. In fact (16) is the solution of the more general problem:

$$\min_{x\in\mathcal{R}^n} \big(\|b - Ax\|^2 + \gamma\|x - y\|^2\big) \tag{19}$$

2.1.3 Proximity operator for $\mu\|\nabla x\|^2$

The definition of the prox for the function f_1 is the answer to this problem:

$$prox_{\gamma f_1}(y) = \min_{x\in\mathcal{R}^n} \big(\gamma f_1(x) + \frac{1}{2}\|x - y\|^2\big) = \min_{x\in\mathcal{R}^n} \big(\gamma\mu\|\nabla x\|^2 + \frac{1}{2}\|x - y\|^2\big) \tag{20}$$

Taking the zero of the Gateau derivative of (20) and assuming that $\|\nabla x\|^2 = x^T\mathcal{L}x$, leads to this result:

$$x(\beta\mathcal{L} + I) = y \qquad \Leftrightarrow \qquad x = (\beta\mathcal{L} + I)^{-1}y \tag{21}$$

with I being the identity matrix and $\beta = 2\mu\gamma$.

Remark: To find x, we should not inverse the matrix $(\beta\mathcal{L} + I)$ because it would be far too slow for big graphs. As a result, we need to find another way to calculate x.

2.1.4 A shortcut through the Fourier graph transform and Chebyshev's polynomial approximation

If we take the Fourier graph transform of (21), we get:

$$\hat{x}(l) = \frac{1}{\beta\lambda_l + 1}\hat{y}(l) = \hat{h}(\lambda_l)\hat{y}(l) \tag{22}$$

where λ_l is the l^{th} eigenvalue of the Laplacian matrix $\mathcal{L}$.

This last operation is not usual in Maschine Learning. In fact, it is used in Digital Signal Processing. Taking the Fourier transfom of (21), alows us to see the problem differently. Equation (22) is equivalent to filter $\hat{y}$ with $\hat{h}$.

This is the typical equation of a low-pass filter. This means that the prior we chose, assumes that our graph is made mostly by low frequency. As a result, to calculate x, we can take the Fourier graph transform of y, multiply it with our filter and take the inverse graph Fourier transform of this result. This could be a good solution for a complex, but small graph (with a non sparse laplacian matrix). However, usually the laplacian matrix of a graph is big (lots of vertices) but sparse (not many edges). As a consequence, it takes plenty of time and a good memory to calculate the eigenvalues and the eigenvectors of $\mathcal{L}$.

In order to solve this problem, we will approximate the transfer function $\hat{h}(\lambda_l) = \frac{1}{\beta\lambda_l+1}$ with a sum of Chebyshev polynomials:

$$\hat{x}(l) = \Big(\frac{1}{2}c_0 + \sum_{k=1}^{\infty} c_k T_k(\lambda_l)\Big)\hat{y}(l) \tag{23}$$

We will now take the inverse graph Fourier transform of this equation. This is trivial as we know $\widehat{\mathcal{L}f} = \lambda_l \hat{f}$

$$x = \Big(\frac{1}{2}c_0 + \sum_{k=1}^{\infty} c_k T_k(L)\Big)y \tag{24}$$

The equation (24) is far more practical than (21) because we can calculate x, only by summing up power of $\mathcal{L}$. Of course we need to approximate this last equation in order to calculate it with a computer. If we approximate the integral of (7) with n rectangles and we only use the N first Chebyshev polynomials we get an approximation of x:

$$\tilde{x} = \Big(\frac{1}{2}c_0 + \sum_{k=1}^{N} c_{k,n} T_k(\mathcal{L})\Big)y \tag{25}$$

We can use this approximation because the forward-backward algorithm converges even with small numeric errors.

Remark about the prior: We can easily use this method for other priors. But not all priors give us a defined function in the graph Fourier domain. This is the reason why $\|\nabla x\|^2$ is a good prior for solving this problem.

2.1.5 Final algorithm

Knowing all this, we get to a simple algorithm to recover the missing labels. We will use the parameters of the Beck-Teboulle [4] proximal gradient algorithm ($\gamma = \beta^{-1}$).

A) Initialisation: $x_0 = u_0 = b$ and $t_0 = 1$

B) Loop for $n = 0, 1, 2, 3, ...$

1) Calculate $y_n = u_n + \frac{2}{\beta}M^*(Mx - b)(u_n)$

2) Calculate $x_{n+1} = prox_{\frac{2\mu}{\beta}}\|\nabla y_n\|^2$ using the equation 25

3) Update $t_{n+1} = \frac{1+\sqrt{4t_n^2+1}}{2}$

$\lambda_n = 1 + \frac{t_n - 1}{t_{n+1}}$

$u_{n+1} = x_n + \lambda_n(x_{n+1} - x_n)$

C) The solution is given by x_n.

Remark: In order to get better results, we can add a constraint on the known labels to prevent them modifying. This will help the algorithm to be more stable.

2.1.6 Implementation

In order to verify the previous result, we will apply it to a virtual problem. We will take a graph representing the road of Minnesota and asign a label to all vertices.

Calculating the prox To start, we are going to verify that (25) is a good approximation of (24). The evolution of the error is given on the figure 1. The definition of the relative error is $e_{rel} = |c - x|/|c|$. We see that it decreases exponentially with m. The computation with the Chebyshev approximation is as expected rapid compared to the direct solution (more than 100 times faster in this case). Knowing this, we will always use the approximation for the next applications.

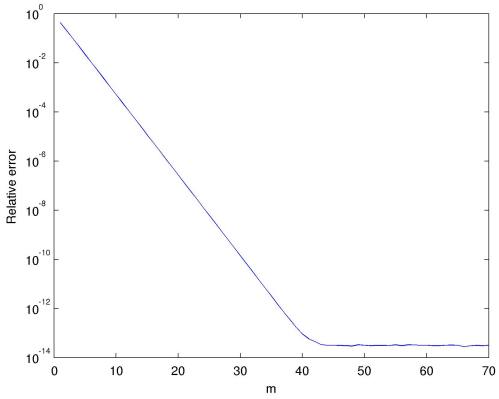

Figure 1: Relative error between the exact solution given by (22) and Chebyshev's polynomial approximation of order m given by (25) for $N = m + 1$. We see that when m reaches 40, we get the computer's precision.

A small denoising problem We have seen that taking the prox of a graph is equivalent to applying a low-pass filter. It can be used to remove a white noise. As shown on figure 2, it works well. It acts exactly the same as a normal signal. The filter removes the noise but also smoothes the edge a little bit.

Recovering missing labels At this point, it is possible to implement the algorithm given at point 2.1.5. Indeed, we have an efficient and precise way to calculate the prox(filtering with Chebyshev polynomial approximation). In the algorithm, we have two parameters (μ and β).

By just watching the algorithm, we can understand that β is inversely proportional to the speed of convergence. This is shown in figure 4. We see that we gain a little by preventing the modification of the known labels. In fact, the biggest advantage of this method is that it's far easier to find good parameters because it's more stable.

The parameter μ should be proportional to the weight we accord to the prior and to the smoothness of the reconstructed signal. In fact, because of the splitting, the smoothness is proportional to $\frac{\mu}{\beta}$.

2.1.7 Conclusion of the first regression problem

The main conclusion of this part is that Chebyshev allows us to implement the algorithm given at point 2.1.5. Without this approximation, the algorithm would be far too slow. In fact, we observe that evaluating Chebyshev polynomials (on matrices) is really efficient for sparse matrices.

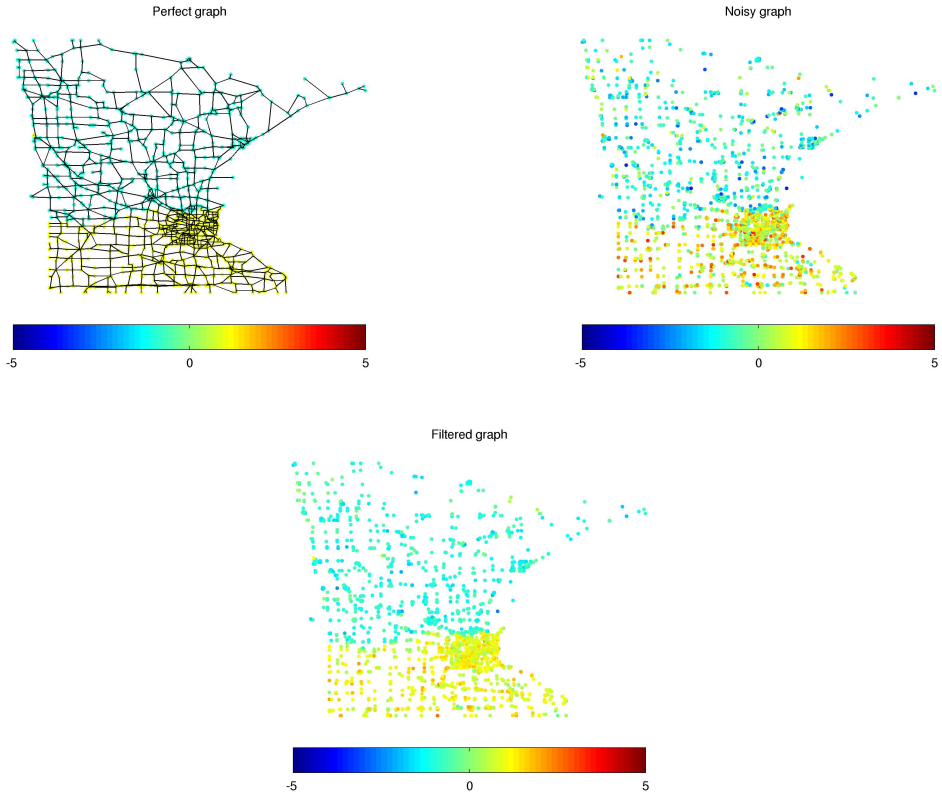

Figure 2: First, we see the perfect label with the edges. Secondly, we see the labels contaminated by a white Gaussian noise ($\sigma = 1$). Thirdly, we see the prox of the noisy graph ($\beta = \sigma$). The mean square error decreases by a factor of 5 (from 1.009 to 0.1951).

2.2 Ridge Regression

We have used Chebyshev polynomials for a general regression problem and for a classification problem. We can also use them for the Ridge Regression. This last problem is a very efficient way to compute a general solution with a training set of data. It's is well presented in [5] and in [6].

2.2.1 Presentation of ridge regression

Suppose we have a training set (of size l) with feature vectors $x_1, x_2, ..., x_l$ and labels $Y = (y_1, y_2, ..., y_l)$. We would like to be able to look for the solution in m different points $x_1^\diamond, x_2^\diamond, ..., x_m^\diamond$. Lets define the unknown labels as: $Y^\diamond = (y_1^\diamond, y_2^\diamond, ..., y_m^\diamond)$. The problem consists in finding a function:

$$y^\diamond = f_A(x_1, y_1, x_2, y_2, ..., x_l, y_l, x^\diamond) \tag{26}$$

which minimizes the functional:

$$R(A) = E\Big(\sum_{i=1}^{m}(y_i^\diamond - f_A(x_1, y_1, x_2, y_2, ..., x_l, y_l, x_i^\diamond))^2\Big) \tag{27}$$

To find the result we are going to transform the last problem into a linear formulation. For this we use a set of n basis functions $\phi_1(x), \phi_2(x), ..., \phi_n(x)$ and we suppose that $f(x)$ can be written like $f(x) = \sum_{i=1}^{n} \alpha_i \phi_i(x)$. The new problem can be written like:

$$\min_{\alpha \in \mathcal{R}^n} \left(\|Y - K\alpha\|^2 + \gamma\|\alpha\|^2\right) \tag{28}$$

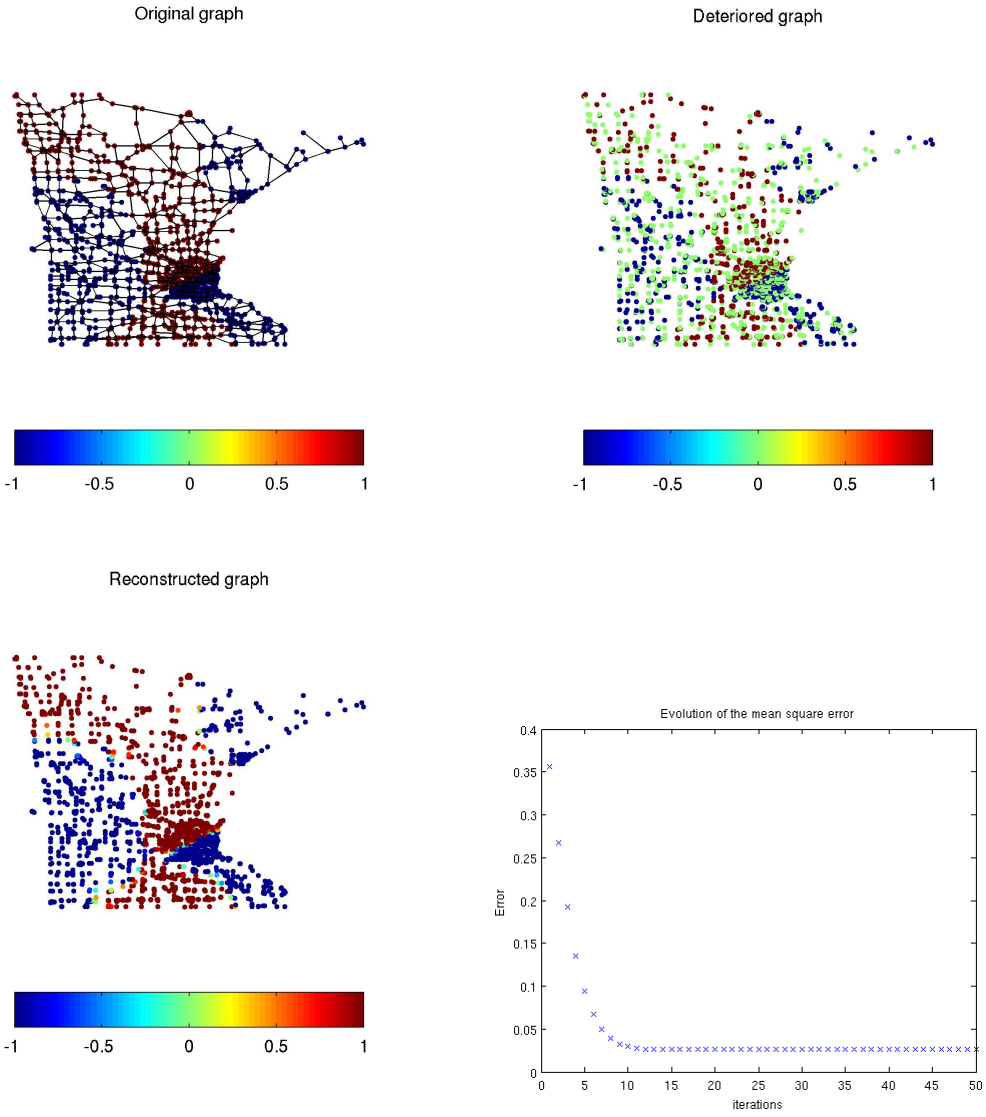

Figure 3: To begin, we see the original graph. Then, we see the known labels (the unknown ones are set to zero (green)). At the end, we see the reconstruction of the graph with $\beta = 4$ and $\mu = 0.5$. The fourth graph show the evolution of the mean square error through iterations.

Remark: We observe that the prior is different than in the first problem. Here the prior is $\|\alpha\|^2$.

$$Y^\diamond = K^\diamond \alpha \tag{29}$$

with

$$K_{i,j} = \phi_j(x_i), \qquad i = 1, ..., l, \qquad j = 1, .., n \tag{30}$$

$$K^\diamond_{i,j} = \phi_j(x^\diamond_i), \qquad i = 1, ..., m, \qquad j = 1, .., n \tag{31}$$

Taking the zero of the Gateau derivative of (28), we obtain:

$$\alpha = (K^t K + \gamma I)^{-1} K^t Y \tag{32}$$

This α should not be mixed up with the second part α parameters.

$$Y^\diamond = K^\diamond \alpha = K^\diamond (K^t K + \gamma I)^{-1} K^t Y \tag{33}$$

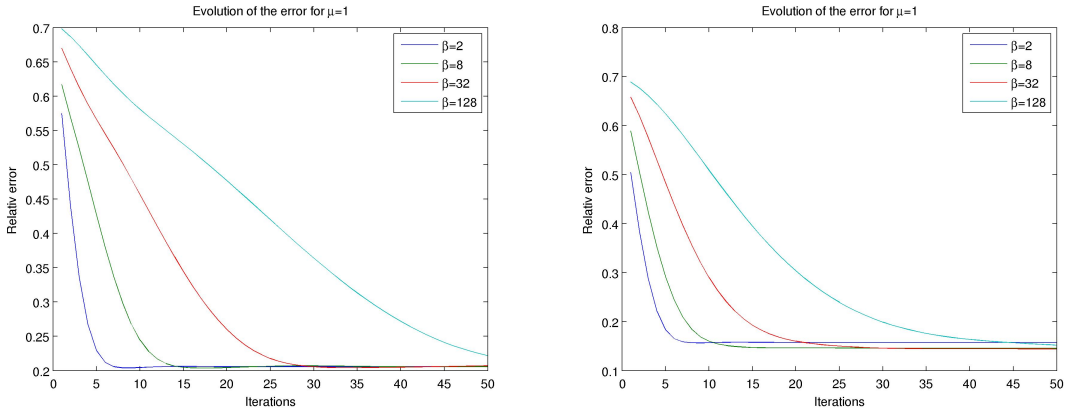

Figure 4: First, the error evolution of the original algorithm given at point 2.1.5. Secondly, we observe the same graph, but for the optimized algorithm (we do not modify the know labels).

2.2.2 Using Ridge Regression

We are going to use Ridge Regression in a particular case which is the most common. Lets define:

$$\phi_i = exp\big(-\frac{\|x - x_i\|^2}{2\sigma^2}\big) \qquad i = 1, ..., l \tag{34}$$

With this basis function the K matrix becomes square ($n = l$). It is the same as the matrix A defined by (48). It's as if we create a graph with K Adjacency matrix of the labelled vertices and $K^\diamond$ Adjacency matrix between labelled and non labelled vertices. This means, we can solve graph problems with Ridge regression using the equality:

$$A = \begin{pmatrix} K & K^\diamond \\ K^{\diamond t} & K^{\diamond\diamond} \end{pmatrix} \tag{35}$$

with $K^{\diamond\diamond}(i,j) = \phi_j^\diamond(x_i^\diamond), \qquad i = 1, ..., m, \qquad j = 1, .., m$

Because of the presence of $K^\diamond$, it does not seem possible to implement (33) as a one shot filter. Although it's possible to find α with one shot filtering the label Y. In this case, we have to use the filter:

$$h_2(x) = \frac{x}{x^2 + \gamma} \tag{36}$$

In that case and like before, the Chebyshev polynomial will be really useful. We just need to chose K as basis and with the same method as before, we can evaluate (33) without inverting in matrix $K^tK + \gamma I$.

In that case, there is no obvious recursive algorithm. But, if we chose K^tK as basis, we can invert the matrix $K^tK + \gamma I$ with the same recursive algorithm as we will use for classification (see section 3) . In that case we have to filter K^tY (not Y) with:

$$h(x) = \frac{1}{x + \gamma} \tag{37}$$

This basis can be really useful because it's not possible to use K if this matrix is not symetric positive definite. This last filtering job can also be done with Chebyshev polynomials. Thus, we are going to compare the two filters, the recursive algorithm we used for classification and the well known recursive algorithms of the minimal residue and of Jacobi.

2.2.3 Datasets

We have made the test on 3 different regression datasets.

Name	Description	Feature	Instances
Boston housing	Estimate the price of the house	13	506
Kin-32th	Predict the movement of robot arm	32	8192
Wine quality	Evaluating the quality of a wine	11	4898

As the results are similar, we are not going to present all 3 datasets. We will concentrate on Boston housing.

2.2.4 Constructing the K matrix

In order to construct the K matrix, we use the function given by equation 34. In this last one, the parameters σ play a really important role. If it is small, the value of K will be nearly 0 except for the diagonal. On the other hand, if it is big, the value of K, will be near to 1.

This is in fact really important because it will directly affect the biggest eigenvalue of K. It has been demonstrated that for a symetric positive definite matrix A of size (r x r), its biggest eigenvalue satisfies [7]:

$$\lambda_{max} \geq \frac{\sum_{i,j} a_{i,j}}{r} \tag{38}$$

This means, that with a big sigma, the biggest eigenvalue of K will also be big. This has two main consequences.

First, recursive algorithm of Jacobi will diverge if the highest eigenvalue of the matrix to invert is bigger than one. Using the relation (38), it is trivial that the largest eigenvalue of $K^tK + \gamma I$ is always superior to 1 if $\gamma > 0$. Thus this algorithm will diverge as shown on figure 5. With the same thinking, it's easy to understand that the recursive algorithm we used for classification will probably not work. Indeed, the basis K^tK will probably have a bigger eigenvalue than γ. Then $\frac{\lambda_{max}}{\gamma} > 1$ and the algorithm will diverge (see (47) with $\frac{1}{\gamma} = \alpha$).

Secondly, the higher eigenvalue also has an influence on the order of the Chebyshev polynomial approximation needed to converge to a certain tolerance. In order to be able to approximate the filter with a low order Chebyshev polynomial, we need a small maximum eigenvalue. I will not prove it, but I will just show that the bound of the truncation error is growing if λ_{max} is increasing.

The truncation error of the Chebyshev polynomial approximation of the function $f(x)$ when $x = [-1, x_2, x_3, ..., x_{n-1}, 1]$ is [1]:

$$|R_n(x)| = |P_n(x) - f(x)| \leq \frac{|f^{(n+1)}(c)|}{(n+1)!} Q(x) \leq \frac{|f^{(n+1)}(c)|}{(n+1)!} \leq \frac{1}{(n+1)!} \max_{x \in [-1,1]} (|f^{(n+1)}(x)|) \tag{39}$$

where $Q(x) = (x+1) \cdot (x - x_2) \cdot (x - x_3) \cdot ... \cdot (x - x_{n-1}) \cdot (x - 1)$

If we transform the interval of interpolation from $[-1, 1]$ to $[0, \lambda_{max}]$, we find $x = \frac{\lambda_{max}}{2}\lambda + \frac{\lambda_{max}}{2}$ we can reformulate the last equation like so:

$$|R_n(\lambda)| = |P_n(\lambda) - f(\lambda)| \leq \frac{|f^{(n+1)}(c)|}{(n+1)!} Q(\lambda) \leq \frac{2{\lambda_{max}}^{n+1}}{4^{n+1}(n+1)!} \max_{\lambda \in [0, \lambda_{max}]} (|f^{(n+1)}(\lambda)|) \tag{40}$$

We observe that function Q is not bounded to 1 anymore this means the error can grow a lot. This is not a demonstration, but an explanation of the result obtained.

2.2.5 Comparison of methods

We are going to compare the recursive method and the Chebyshev approximation. Thus, we have to define a criterion. We chose the communication cost [2]. Let's imagine you do not have a central computer, but a multiple of sensors with small computation units. In this case, you can compute the solution locally. At each vertex, we can find the solution by asking the neighbourhood's values. We define the communication cost as the number of messages exchanged in the graph.

1. This formula and the next one is taken from the website:
http://math.fullerton.edu/mathews/n2003/chebyshevpolymod.html

If you have to multiply a vector (containing all the vertex) by $\mathcal{L}$ you will need $2|\mathcal{L}|$ exchange of messages (where $2|\mathcal{L}|$ is equal to the number of elements of $\mathcal{L}$). After detail computation, we find that the communication cost of the M iteration of the recurrence is the same as M order Chebyshev approximation: $2 \cdot M \cdot |\mathcal{L}|$.

Because of this result, we are going to plot the error versus the number of iteration or the order of Chebyshev approximation (same axis).

2.2.6 Result

Looking to figure 5 and 6, we can make some observations that convince us of all the previous theory. First, we observe that the recursive algorithms are diverging (except minimal residue) because λ_{max} is too big. Second the filter $h(x)$ is less good than $h_2(x)$ because the maximum eigenvalue of K^tK is bigger than the one of K.

One important result is that the filter $h_2(x)$ seems to give better results than the minimal residue algorithm.

If we change parameters we find that Chebyshev polynomial approximation can be less good than the minimal residue algorithm if λ_{max} is really big. However, in that case, the matrix K has a lot of big values (equation 38) and could not be sparsified. Thus there is no big use in implementing it distributively.

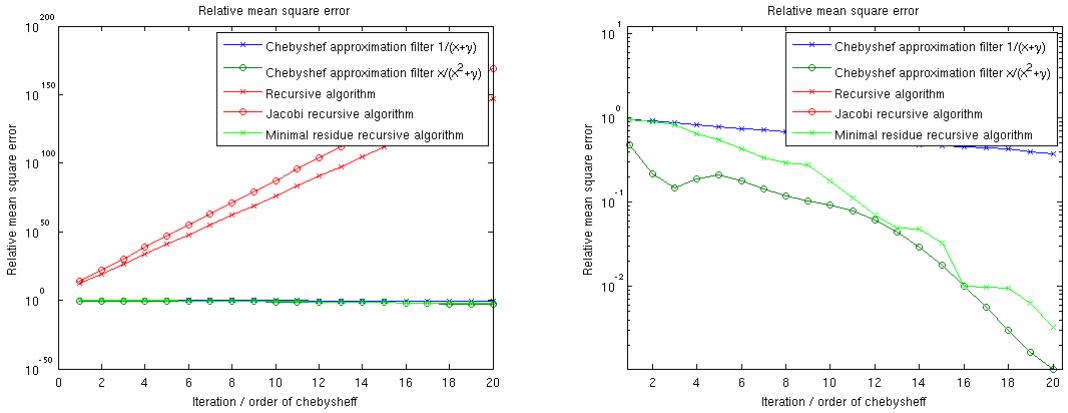

Figure 5: Comparison of the different methods to solve the problem and get α. The parameters are: $\gamma = 2.405$, $\sigma = 2$. The highest eigenvalue of K is 110 and of K^tK is $12'167$. The plot on the right is only a close-up of the main plot (left). Recursive algorithms (in red) are diverging.

2.2.7 Consequence of sparsification

In order to solve a problem distributively, we do not only care about the order of the Chebyshev polynomial approximation. To get a low communication cost, we would like to sparsify the K matrix as much as possible with the lowest induced error. In this work, we have first observed the result of sparsifying the matrix K on α. Then, we have calculated the induced error on $Y^\diamond$. In fact sparsifying the matrix K put a limit to the precision of the calculated α. This leads to a point that even if, we increase the order of the Chebyshev polynomial, we do not reduce the error (figure 7 left).

On figure 7 (right), we observe the impact of the alpha error on the desired result $Y^\diamond$. We tried a different threshold to sparsify the matrix K and for each of them, we have calculated the induced error on $Y^\diamond$ and the percent of zeros in the matrix K. We observe that the error on alpha is first helping the solution to get to accuracy (negative induced error). But, when K is more sparse, the induced error on Y^* grows in same shape than the error on α. In fact it grows faster and a relative error on α induced approximatively two times more error on $Y^\diamond$.

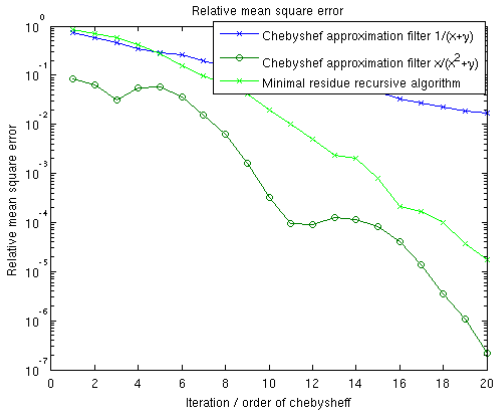

Figure 6: Comparison of the different methods to solve the problem and get α. The parameters are: $\gamma = 2.405$, $\sigma = 1$. The highest eigenvalue of K is 27 and of K^tK is 741. The K matrix has been sparsiefied with a threshold of 0.01 and had 46% of non zero values.

This last figure (7 right), varies a lot from one experience to another. Unfortunately, it's not really possible to make an average because we do not chose the values on the abscissa. On other experiments, the shape is quite the same, but the error can be different (bigger or smaller). Sometimes, the sparsification always gives positive induced error.

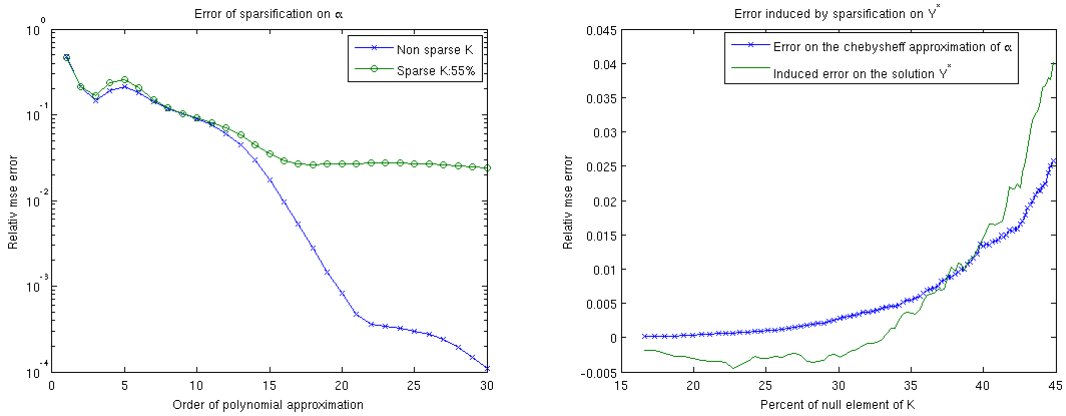

Figure 7: Comparison of the different methods to solve the problem and get α. The parameters are: $\gamma = 2.405$, $\sigma = 2$. On the left we see the effect of the sparsification over iterations on α. On the right we see the induced error by sparsification.

3 Classification

Until now, we have solved regression problems although some applications need a discrete solution. For instance, imagine that we would like to determine what the written number on an image is. We have already labelled some samples and the goal is to determine all the others. This problem can be solved with graph reconstruction techniques. First we create the graph. For this, we assign the distance between two images to the edge connecting the corresponding two vertices. (In order to get a sparse edge matrix we can add some other conditions like a minimum threshold or maximum number of elements per line). If we formulate the problem like this, it becomes equivalent to classifying elements into different classes. To solve this type of problem, we can just

classify the solution of a regression problem. Unfortunately, this technique gives bad results when the number of classes is greater than two.

In order to avoid reclassification trouble, we can directly consider the discrete problem. We have to reformulate it a little, but in the end, it would still be possible to use our Chebyshev approximation trick (shown in part 2.1.4).

The algorithm and his solution are shown in [1] and [8]

3.1 A new formulation of the problem

We have a set of n points $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ which can take c $(1, 2, ..., c)$ different labels. We only know some of them.

Let's define a matrix B of size n x c. $B_{i,j} = 1$ if $x_i = j$ and $B_{i,j} = 0$ otherwise. Moreover, $B_{i,j} = 0$ if the label of x_i is unknown.

If we solve:

$$\min_{F \in \mathcal{R}^{nxc}} \left(\|B - F\|^2 + \mu \|\nabla F\|^2 \right) \tag{41}$$

The solution of our classification problem is given by $y_i = j$ with j satisfying $\max_{j \leq c} F_{i,j}$

3.2 Another edge derivative

In the previous problem, we considered the edge derivative as being $\frac{\partial f(v)}{\partial e_u} = \sqrt{\mathcal{L}(u,v)}(f(u) - f(v))$. In fact, it's not the only way to define this derivative. Other ways would just lead to other Fourier Graph Transform definitions. Let's look at an example to make it clear.

We can define the edge derivative like this:

$$\frac{\partial f(v)}{\partial e_u} = \sqrt{\mathcal{L}(u,v)}\left(\frac{f(u)}{\sqrt{d_u}} - \frac{f(v)}{\sqrt{dv}}\right) \tag{42}$$

Let's calculate the norm of the gradient of f. We have:

$$\|\nabla f\|^2 = \frac{1}{2} \sum_u \sum_v \left(\frac{\partial f(v)}{\partial e_u}\right)^2 = f^T D^{-\frac{1}{2}} \mathcal{L} D^{-\frac{1}{2}} f = f^T \mathcal{L}_n f \tag{43}$$

The last equation shows that for our new definition of the edge derivative, we need to consider the normalized laplacian. As a result, the graph Fourier transform will be computed with the eigenvector of the normalized laplacian. We see that taking another edge derivative definition simply leads to another basis.

Remark: Later we are going to consider two more definitions of laplacian: $D^{-1}\mathcal{L}$ and $\mathcal{L}^T D^{-1}$. There are a lot of others (K-scaling, L-scaling, inverse cosine, ...).

3.2.1 The different basis we tested

$\mathcal{L}$ is the usual definition of the laplacian, D is the diagonal of $\mathcal{L}$ and I the identity matrix.

Laplacian	Recurrence matrix	Name given to the method
$\mathcal{L}$	$I - \mathcal{L}$	Unnormalized Laplacian
$\mathcal{L}_n = D^{-0.5} \cdot \mathcal{L} \cdot D^{-0.5}$	$S = I - \mathcal{L}_n$	Normalized Laplacian
$\mathcal{L}_{n2} = D^{-1} \cdot \mathcal{L}$	$P = I - \mathcal{L}_{n2}$	Label propagation with P
$\mathcal{L}_{n3} = \mathcal{L} \cdot D^{-1}$	$P^t = I - \mathcal{L}_{n3}$	Label propagation with P^t

The recurrence matrix is the one used to solve the problem iteratively with (46) (In that equation the recurrence matrix is given by S).

Remark: When we solve a problem, we don't really care about the edge derivative. In fact, we just need the definition of the laplacian (or the kernel) that will impose the graph Fourier transform. We cannot take a random matrix in order to get a valid basis. We need the eigenvectors all to be linearly independent from each other. Thus, we will take only positive definite matrices as bases.

3.3 Solution of the discrete problem

The solution of (41) is:

$$F = (\mu\mathcal{L}_n + I)^{-1} \cdot B \tag{44}$$

We can reformulate it with a matrix: $S = I - \mathcal{L}_n$ and two new variables: $\alpha = \frac{\mu}{1+\mu}$ and $\beta = \frac{1}{1+\mu}$:

$$F = \beta \cdot (I - \alpha S)^{-1} \cdot B \tag{45}$$

This formulation can be found in literature because it suggests an iterative algorithm. We will adopt the new formulation in order to compare our results easily. The iterative algorithm is:

$$F(k+1) = \alpha SF(k) + (1-\alpha)B \tag{46}$$

The obtained solution after t iteration is:

$$F(t) = (\alpha S)^{t-1}B + (1-\alpha)\sum_{i=0}^{t-1}(\alpha S)^i B \tag{47}$$

Providing $\alpha \cdot \lambda_{max} < 1$ (where λ_{max} is the maximal eigenvalue of S),the solution of this recursion is correct to a factor $\frac{\beta}{1-\alpha}$. We do not really care about this factor, as we need to take the maximum of each line.

Thus, we have two ways of solving the problem. Either we iterate the recurrence, or we approximate the filter function with Chebyshev polynomials.

Remark: Because we take the maximum to recover the label, this problem becomes highly non linear. In fact, we just do the filtering for each class and then, for each vertex, we watch which class contaminated it the most. With this method, the class with the biggest amount of labelled vertex will have an advantage because it also has the most energy. We can abolish this privilege by dividing each column of F by the number of labelled elements of each class after the filtering. The result will be slightly improved. Let's call this last method: "optimized classification" and the original one "original classification".

3.4 USPS dataset

This dataset is composed by hand-written numbers (16 x 16 pixels black and white images). Our goal is to be able to assign a number to each image. We only work on four numbers (1 to 4). First we are going to create a graph by taking into account the distance between all the images. The inverse distance function we have used is (x is the vector of features, for USPS, it's made of all the pixel values):

$$A_{i,j} = exp\big(-\frac{\|x_i - x_j\|}{2\sigma^2}\big) \tag{48}$$

Thus the obtained matrix (A) will be symetric and positive definite. This implies that the chosen basis, $\mathcal{L}$,$\mathcal{L}_n$,S,P,P^t are also symetric positive definite.

In order to get a sparse graph, we set all edges smaller than a threshold T to zero(We can also consider the k closer neighbour exclusively). This is, of course, not the best way to create the graph, however our goal is not to obtain the best result possible, but to compare two different methods.

3.4.1 Results: the big non linearity

On the USPS dataset, the algorithm gives the best result with a very small α, but non zero (between 0.001 and 0.1)(see figure 8). This means a very small μ and a very small contribution of the prior. We hardly filter the signal. In that case, the convergence is also really fast: We nearly converge with the first order approximation. With a big α, close to 1 (=> big μ), the algorithm converges to a solution which is not the right one at all. Moreover, it takes more iteration to reach the convergence.

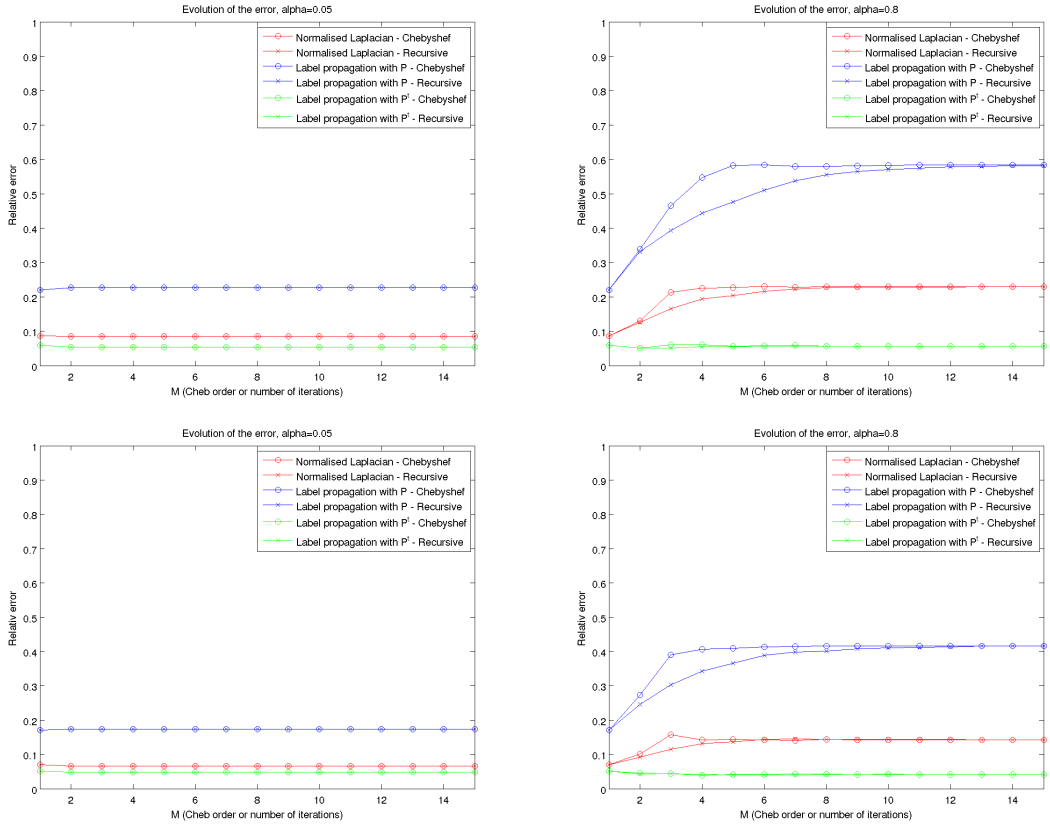

Figure 8: Result obtained for the USPS datasets:
- First plot: original classification with α=0.05
- Second plot: original classification with α=0.8
- Third plot: optimized classification with α=0.05
- Fourth plot: optimized classification with α=0.8

This strange result is due to our dataset (in order to get a connected graph, we need to keep to edges) and to the non linearity of our algorithm. In order to get the correct class, we look (line by line) for the column with the maximum value. Considering this, we understand why we do not need to filter a lot of the data. In fact, the algorithm with a small α will give a solution very similar to the "closest neighbourhood algorithm" (look for the closest labelled neighbour and adopt its label).

On figure 8, we also see that the "optimized classification" algorithm gives slightly better results than the original one.

It seems that our Chebyshev algorithm is better for a big α. But, the algorithm converges to a bad solution. For a small α, Chebyshev or the recursiv algorithm converges directly after one iteration to a good solution. As a result, we cannot really say that one of those two methods is better.

3.5 Minessota dataset

We know that our results depend highly on the chosen dataset. Thus we have tested our method to another dataset. We create a fake problem with the graph of the road of Minessota. The problem consists in classifying all the labels in 5 different classes with only 5 percent of known labels. To create the problem, we just add the sign function of the 4^{th} and the 8^{th} eigenvector of

the Laplacian matrix. The result is shown on figure 9

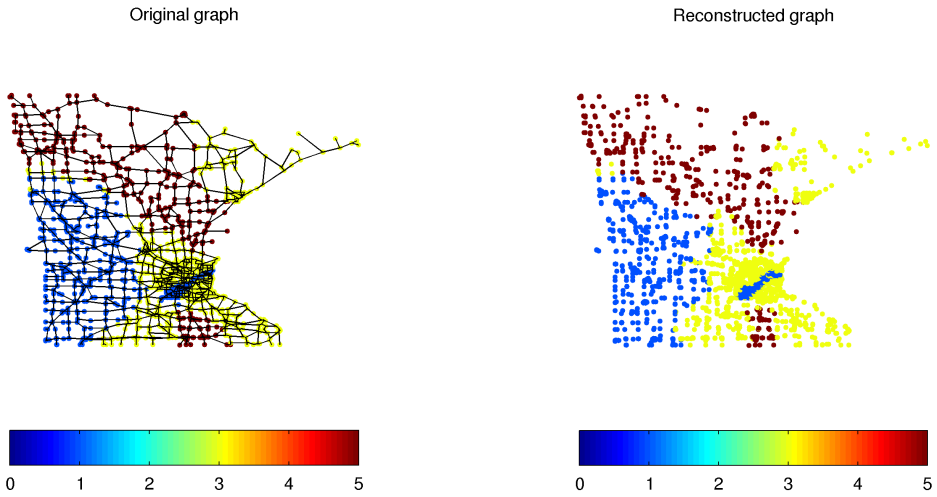

Figure 9: The first graph shows the label of the graph. The second is a reconstructed graph with $\alpha = 0.1$ for 5 percent of known labels

In this problem, the graph is far more sparse than in the USPS. As a result it takes more iteration (or bigger order of the Chebychev approximation) to converge the solution. The role of α seems to be different (see figure 10). The results are slightly better for a bigger α (near one). We also remark that the choice of the basis has hardly any influence on the solution. This was really different for the USPS dataset.

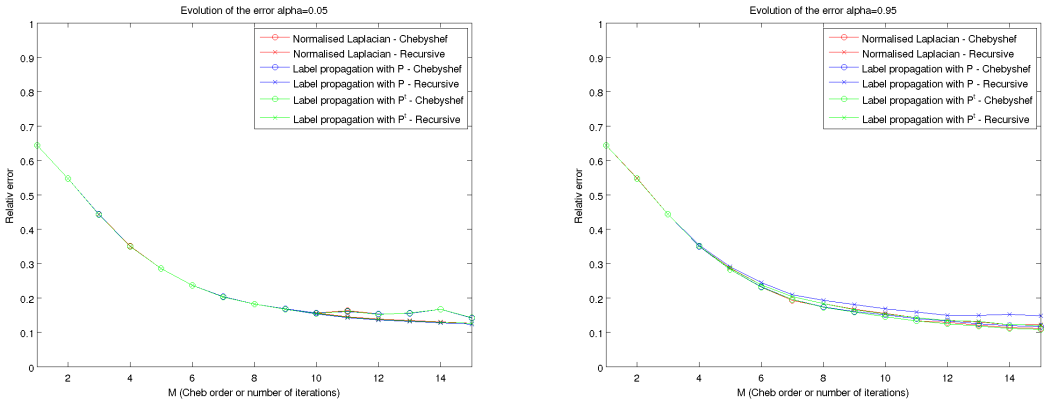

Figure 10: Result obtained for the Minessota dataset:
- First plot: original classification with α=0.05
- Second plot: original classification with α=0.95

3.6 Other datasets

We have also tested 3 other classical datasets.

Name	Description	Feature	Instances	Categories
Iris	Classification of flowers	4	150	3
Letter Recognition	Recognition of English capital letters	17	20000	26
Haberman's Survival	Prediction of the survival of patients who had undergone surgery for breast cancer	3	306	2

All the features of these datasets have been normalized to a zero mean and unity variance. To create the graph we have used the basis function as for USPS (see (48)). As we would like to minimize the communication cost, we need a sparse graph. For this we can play on two different parameters: σ the variance of the inverse distance function given by (48) and T the threshold above which we set an edge to zero.

3.6.1 Results

We are not going to present all the result in details. Instead we will just give the principal conclusions. We made the test for different basis', values of α, σ, and thresholds.

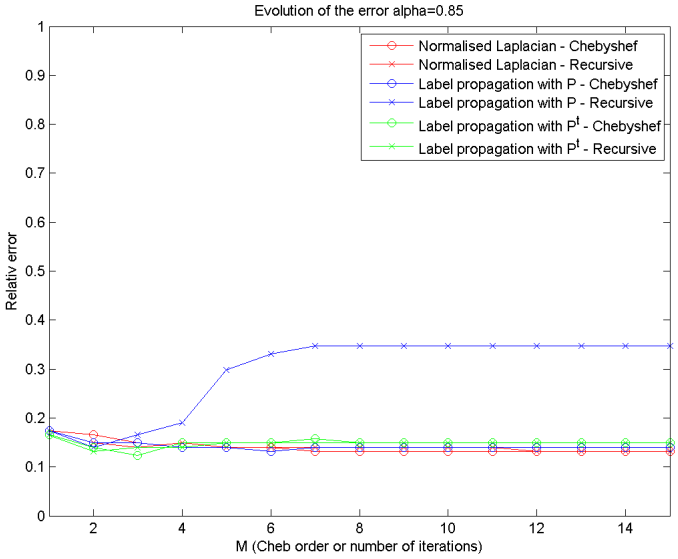

Figure 11: Result obtained for the iris dataset with $\sigma = 0.5$ and $T = 0.02$. Out of 150 samples, 25% are labelled. The resulting basis is made of 26% of non zero elements (approximatively 37 connection per vertex). P is close to singularity.

Sparsity: The sparsification does not really make the error of classification grow (some percent). For the letter recognition, the sparsification is even good as it improves the percent of good classification. This suggests that classification problems can be solved distributively with a low communication cost.

Recursive algorithm VS Chebyshev polynomials The basis P can be close to singularity when we try to spasify it. This is a problem for the recursive algorithms. In fact they converge to

a point, but it's not the right one. The Chebyshev polynomial approximation is far more stable and doesn't suffer from this (See figure 11).

Conclusion: Because of its stability, the Chebyshev polynomial approximation seems to be better than the recursive algorithm.

4 Conclusion

In this work we have studied Chebyshev polynomial approximation in two different cases: regression and classification. The first problem was really general and our method allows us to reconstruct a transformed signal providing the self-adjoined matrix of the transformation is defined. In that case, a graph regression problem, we have seen that Chebyshev allows us to solve the problem quickly.

Then we have worked on ridge regression. We mented on different datasets. We studied in more detail the convergence of Chebyshev approximation. We find that for sparse matrices, Chebyshev converges faster than traditional recursive algorithms. We also calculated the impact of sparsification on the answer.

In the second times, we care about classification. We have studied different datasets and basis'. We have observed that Chebyshev has the same communication cost as the recursive algorithm. But it's more stable and it always converges to the right solution, unlike to the recursive algorithm, which needs a basis with good properties.

The main conclusion of this work is that Chebyshev polynomial approximation is really efficient and stable for sparse matrices. As a consequence, it's well adapted for graph signals with a few edges. We have also observed that it's possible to create sparse graphs based on data without depleting the solution too much. Thus, we have a very stable way to implement a regression or a classification problem distibutively.

References

[1] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Adv. Neural Inf. Process. Syst.*, S. Thrun, L. Saul, and B. Schölkopf, Eds. MIT Press, 2004, pp. 321–328.

[2] D. I. Shuman, P. Vandergheynst, and P. Frossard, "Chebyshev polynomial approximation for distributed signal processing," *CoRR*, vol. abs/1105.1891, 2011.

[3] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Appl. Comput. Harmon. Anal.*, vol. 30, no. 2, pp. 129–150, Mar. 2011.

[4] P. L. Combettes and J.-C. Pesquet, "Proximal Splitting Methods in Signal Processing," *ArXiv e-prints*, Dec. 2009.

[5] O. Chapelle, V. Vapnik, and J. Weston, "Transductive inference for estimating values of functions," *Advances in Neural Information Processing Systems*, vol. 12, pp. 421–427, 1999.

[6] C. Cortes and M. Mohri, "On transductive regression," *Advances in Neural Information Processing Systems 19*, vol. 19, pp. 305–312, 2007.

[7] R. Panigrahy, K. Talwar, and U. Wieder, "Lower bounds on near neighbor search via metric expansion," *CoRR*, vol. abs/1005.0418, 2010.

[8] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Proc. Ann. Conf. Comp. Learn. Theory*, ser. Lect. Notes Comp. Sci., B. Schölkopf and M. Warmuth, Eds. Springer, 2003, pp. 144–158.