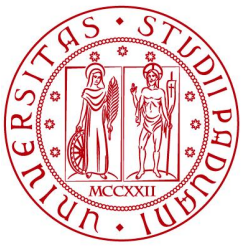

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

Corso di Laurea Triennale in Matematica

Regolarizzazione e Semi-supervised Learning su grafi di grandi dimensioni

Relatore:
Prof. Wolgang Erb

Laureanda: Giulia Cippitelli
Matricola: 1174699

Anno Accademico 2022/2023

24 Febbraio 2023

Abstract

La seguente trattazione si occupa di prevedere l'etichettatura di un grafo parzialmente etichettato e di conseguenza di ricostruire una funzione sul grafo da un insieme di campionamenti. Sono stati analizzati due algoritmi in particolare: la regolarizzazione di Tikhonov e l'interpolazione basate su un metodo kernel. Sono poi state analizzate due stime dell'errore di generalizzazione sui dati confrontandole grazie ad alcune proprietà del kernel. Gli algoritmi sono stati applicati a diversi dataset con lo scopo di esaminare l'errore compiuto. Si è giunti infine alla seguente conclusione: per pochi nodi etichettati l'interpolazione risulta molto valida. Al crescere dei nodi etichettati, per scelte accurate del paramentro di regolarizzazione, l'algoritmo di Tikhonov risulta però più efficace.

Indice

Introduzione

Il semi-supervised learning è un ramo del machine learning in cui vengono utilizzati solo pochi dati etichettati per riuscire a classificare un grande quantitativo di dati. Questo approccio risolve il problema di ottenere set di dati etichettati a mano che potrebbero essere costosi dal punto di vista computazionale. I dati etichettati con il semi-supervised learning, invece, vengono usati con la consapevolezza che sono imprecisi, ma, insieme alla struttura geometrica del grafo, possono comunque essere utilizzati per creare un forte modello predittivo. Il semi-supervised learning può quindi essere considerato un ragionevole metodo tra il supervised e l'unsupervised learning.
Un problema chiave nella teoria dell'apprendimento computazionale è trovare la funzione che, dato un grafo connesso parzialmente etichettato, sia adatta alla struttura del grafo. Per fare questo bisogna prima prevedere le etichette dei restanti vertici. Per ricondurci ad una funzione definita sui vertici del grafo si analizza un problema di minimizzazione tramite due algoritmi proposti da [1], cioè quello di Tikhonov, che utilizza un parametro γ di regolarizzazione, e l'interpolazione, che è un caso particolare di Tikhonov con $\gamma = 0$. Tutti e due gli algoritmi si basano su una funzione kernel che incorpora la struttura geometrica del grafo. Questi metodi appena citati sono tra i più utilizzati nel semi-supervised learning.
Grazie ai risultati studiati da O. Bousquet e A. Elisseeff in [4] riguardanti la stima degli errori di generalizzazione degli algoritmi di apprendimento si ottiene un'ulteriore stima in aggiunta a quella proposta da M. Belkin, I. Matveeva e P. Niyogi in [1]. Per fare questo è stato utile studiare alcune proprietà del kernel di un grafo, definite in [5], e confrontare come le assunzioni fatte sulla liscezza del kernel facciano ottenere una convergenza diversa. Vengono inoltre date alcune stime del modulo della funzione osservando delle relazioni con alcune proprietà del grafo sottostante.
Sono poi stati modificati i codici usati in [2] con lo scopo di confrontare tre esempi diversi di nodi etichettati con due paramentri di regolarizzazione applicati alla regolarizzazione di Tikhonov e all'interpolazione. Il vantaggio della regolarizzazione di Tikhonov è che utilizza un parametro di regolarizzazione γ in più per determinare la soluzione, bisogna tuttavia prestare attenzione alla scelta del dataset che potrebbe non risultare vantaggioso. Inoltre il valore ottimale del paramentro di regolarizzazione è solitamente incognito e spesso nei problemi pratici è determinato ad hoc. Questo tipo di regolarizzazione non risulta, infatti, essere sempre la scelta migliore.

La trattazione è organizzata secondo il seguente schema. Nel primo capitolo vengono definiti due algoritmi per la regressione sui grafi di una funzione e viene poi definito brevemente il kernel. Il secondo capitolo si concentra sull'analisi teorica degli errori di generalizzazione degli algoritmi, osservando come assunzioni più forti sul kernel possano semplificare tale errore. Il terzo capitolo è interamente dedicato ai risultati numerici prendendo in considerazione un particolare dataset e scegliendo due parametri di regolarizzazione. Partendo dal dataset delle due lune si randomizzano tre scelte di nodi

etichettati e si osservano le differenze ottenute. È comunque possibile ottimizzare i risultati su due parametri grazie a [3]. Chiudono la trattazione le conclusioni, seguite da un'Appendice contenente i codici Matlab utilizzati nei test numerici.

Capitolo 1

Regressione su grafi

Vogliamo prevedere le etichette dei vertici di un grafo parzialmente etichettato. L'obiettivo è quello di implementare algoritmi che siano adatti alla struttura del grafo. Consideriamo un grafo pesato $G = (V, E)$, dove $V = \{x_1, ..., x_n\}$ è l'insieme dei vertici e E è l'insieme dei archi. Ad ogni arco del grafo è associato un peso W_{ij}. Si considera un sottoinsieme di questi vertici etichettato con valori y_i. Vogliamo prevedere i valori dei restanti vertici.
Per approssimare una funzione ad un grafo G abbiamo bisogno della nozione di *buona* funzione. Formalizziamo questa nozione tramite lo smoothness functional

$$S(f) = \sum_{i \sim j} W_{ij}(f_i - f_j)^2$$

dove la somma è fatta tra i vertici adiacenti di G. Se f è una *buona* funzione, il funzionale S assume piccoli valori. È importante osservare che

$$\sum_{i \sim j} W_{ij}(f_i - f_j)^2 = \mathbf{f}^T L \mathbf{f}$$

dove L è la matrice laplaciana, cioè $L = D - W$ con $D = diag\,(\sum_i W_{1i}, ..., \sum_i W_{ni})$.

1.1 Algoritmi per regressione su grafi

Introduciamo ora due algoritmi per la regressione di funzioni su grafi parzialmente etichettati.
Sia $G = (V, E)$ il grafo con n vertici e con matrice dei pesi W, i cui elementi W_{ij} sono definiti come sopra. Assumiamo che G sia connesso e che i vertici del grafo siano numerati. Vogliamo ricondurci ad una funzione $f : V \longrightarrow \mathbb{R}$. f è definita sui vertici di G, ma abbiamo informazioni solo per i primi k vertici, quindi $f(x_i) = y_i$, $1 \leq i \leq k$. Inoltre le etichette del grafo possono avere degli errori. Precondizioniamo, quindi, i dati sottraendo la media nel modo seguente

$$\tilde{\mathbf{y}} = (y_1 - \bar{y}, ..., y_k - \bar{y})$$

dove $\bar{y} = \frac{1}{k} \sum y_i$.

Algoritmo 1: regolarizzazione di Tikhonov (parametro $\gamma \in \mathbb{R}$)
L'obiettivo è minimizzare il quadrato della funzione di perdita sommato alla funzione di penalità liscia.

$$\tilde{\mathbf{f}} = \underset{\substack{\mathbf{f}=(f_1,...,f_n)\\ \sum f_i=0}}{\operatorname{argmin}} \frac{1}{k}\sum_i (f_i - \tilde{y_i})^2 + \gamma \mathbf{f}^T S \mathbf{f}$$

S è una matrice di smoothness, per esempio $S = L$ o $S = L^p, p \in \mathbb{N}$. La condizione $\sum f_i = 0$ è necessaria per rendere l'algoritmo stabile. Senza perdita di generalità possiamo supporre che i primi l vertici del grafo siano etichettati. l potrebbe essere diverso da k, poichè consentiamo ai vertici di avere etichette diverse (o la stessa etichetta più volte). Cerchiamo ora la soluzione del problema quadratico sopra riportato, che non è difficile da ottenere tramite osservazioni di algebra lineare standard. Denotiamo con $\mathbf{1} = (1, 1, ..., 1)$ il vettore con tutte le componenti uguali ad uno, allora la soluzione può essere data nella forma

$$\tilde{\mathbf{f}} = (k\gamma S + I_k)^{-1}(\tilde{\mathbf{y}} + \mu \mathbf{1}) \tag{1.1}$$

Qui $\tilde{\mathbf{y}}$ è il vettore di n componenti $\tilde{\mathbf{y}} = (\sum_i y_{1i}, \sum_i y_{2i}, \sum_i y_{mi}, 0, ..., 0)$, dove si sommano le etichette corrispondenti allo stesso vertice sul grafo.
I_k è una matrice diagonale di molteplicità

$$I_k = diag(n_1, n_2, ..., n_l, 0, ..., 0)$$

dove n_i è il numero di occorrenze del vertice i tra i punti etichettati; μ viene scelto in modo tale che $\mathbf{f}$ sia ortogonale a $\mathbf{1}$. Denotiamo con $s(\mathbf{f})$ il funzionale

$$s : \mathbf{f} \longrightarrow \sum_i f_i$$

Dato che s è lineare, otteniamo $0 = s(\tilde{\mathbf{f}}) = s((k\gamma S + I_k)^{-1}\tilde{\mathbf{y}}) + s((k\gamma S + I_k)^{-1}\mathbf{1})\mu$.
Dunque possiamo scrivere

$$\mu = -\frac{s((k\gamma S + I_k)^{-1}\tilde{\mathbf{y}})}{s((k\gamma S + I_k)^{-1}\mathbf{1})}$$

Notiamo che togliere la condizione $\mathbf{f} \perp \mathbf{1}$ è equivalente a porre $\mu = 0$.

Algoritmo 2: Interpolazione (senza parametri)
Assumiamo che i valori $y_1, ..., y_k$ non abbiano errori. Dunque il problema di ottimizzazione diventa trovare una funzione di massimo smoothness che soddisfi $f(x_i) = \tilde{y_i}$, $1 \leq i \leq k$:

$$\tilde{\mathbf{f}} = \underset{\substack{\mathbf{f}=(\tilde{y_1},...,\tilde{y_k},f_{k+1},...,f_n)\\ \sum f_i=0}}{\operatorname{argmin}} \mathbf{f}^T S \mathbf{f}$$

Come prima S è una smoothness matrix, per esempio L o L^2. Scriviamo S come segue

$$S = \begin{pmatrix} S_1 & S_2 \\ S_2^T & S_3 \end{pmatrix}$$

dove S_1 è una matrice $k \times k$, S_2 è $k \times n-k$ e S_3 è $(n-k) \times (n-k)$. Sia $\tilde{\mathbf{f}}$ il vettore che ha come componenti i valori di f dall'indice $k+1$ all'indice n, dove la funzione è sconosciuta, quindi $\tilde{\mathbf{f}} = (f_{k+1}, ..., f_n)$. Si può quindi scrivere la soluzione:

$$\tilde{\mathbf{f}} = S_3^{-1} S_2^T ((\tilde{y_1}, ..., \tilde{y_k})^T + \mu \mathbf{1}) \tag{1.2}$$

$$\mu = -\frac{s(S_3^{-1}S_2^T\tilde{\mathbf{y}})}{s(S_3^{-1}S_2^T\mathbf{1})}$$

Si può mostrare che, data una funzione f, e denotando rispettivamente la regolarizzazione di Tikhonov e l'interpolazione con Reg_γ e Reg_{int}, si ottiene

$$\lim_{\gamma\to 0} Reg_\gamma(f) = Reg_{int}(f)$$

Tale relazione ci suggerisce di utilizzare la condizione $\mathbf{f} \perp \mathbf{1}$ anche per l'interpolazione, sebbene in questo caso non siano disponibili limiti basati sulla stabilità .

1.2 Kernel

Diamo ora una definizione di kernel che ci servirà più avanti nell'analisi teorica. Consideriamo una funzione di somiglianza del tipo:

$$\begin{aligned} k\colon X \wedge X &\to \mathbb{R} \\ (x, x') &\mapsto k(x, x') \end{aligned}$$

cioè una funzione che, dati due modelli x e x', restituisce un numero reale che caratterizza la loro somiglianza. Se non diversamente specificato assumeremo che k sia simmetrico, cioè $k(x, x') = k(x', x)$ per ogni $x, x' \in X$. La funzione k è chiamata *kernel*. Al fine di essere in grado di utilizzare un prodotto scalare come funzione di somiglianza abbiamo prima bisogno di rappresentare i modelli come vettori in qualche spazio del prodotto scalare. Per fare questo definiamo la mappa:

$$\begin{aligned} \phi\colon X &\to H \\ x &\mapsto \mathbf{x} := \phi(x) \end{aligned}$$

H viene chiamato *feature space*. Si noti che abbiamo usato $\mathbf{x}$ in grassetto per denotare la rappresentazione vettoriale in H. Incorporare i dati in H tramite ϕ ha tre vantaggi:

- ci permette di definire una funzione di somiglianza tramite il prodotto scalare in H

$$k(x, x') := \langle x, x' \rangle = \langle \phi(x), \phi(x') \rangle$$

- ci permette di trattare con i modelli geometricamente, e quindi ci permette di studiare algoritmi di apprendimento che utilizzano algebra lineare e geometria analitica;

- la libertà di scegliere la mappa ϕ ci consentirà di progettare una grande varietà di funzioni di somiglianza e algoritmi di apprendimento.

Ora consideriamo X come un sottoinsieme dello spazio vettoriale di $\mathbb{R}^N$, ($N \in \mathbb{N}$) dotato del canonico prodotto scalare. Supponiamo che siano dati $x \in X$, dove la maggior parte delle informazioni è contenuta nei prodotti di ordine d (chiamati monomi) delle entrate $[x]_j$ di x,

$$[x]_{j_1} \cdot [x]_{j_2} \cdots [x]_{j_d}$$

dove $j_1, ..., j_d \in \{1, ..., N\}$. Spesso questi monomi sono riferiti a un product features. Queste caratteristiche costituiscono la base di molti algoritmi pratici, infatti esiste un intero campo di ricerca sul riconoscimento dei modelli che studia i classificatori polinomiali, che si basa, dapprima, sull'estrazione di product features e poi sull'applicazione di algoritmi di apprendimento su queste caratteristiche. In altre parole i modelli sono processati mappando nello spazio caratteristica tutti i prodotti delle d entrate.

Definizione. Data una funzione da $k : X^2 \to \mathbb{K}$ (dove $\mathbb{K} = \mathbb{C}$ o $\mathbb{K} = \mathbb{R}$) e $x_1, ..., x_m \in X$, la matrice $m \times m$ con elementi

$$K_{ij} := k(x_i, x_j)$$

si chiama *Matrice di Gram* (o matrice di kernel) di k rispetto a $x_1, ..., x_m$.

Capitolo 2

Analisi teorica

Esaminiamo ora alcune garanzie teoriche per la generalizzazione dell'errore di regolarizzazione su grafi. Usiamo la nozione di stabilità dell'algoritmo per ottenere limiti per reti di regolarizzazione.
L'obiettivo di un algoritmo di apprendimento è quello di conoscere una funzione su qualche spazio V tramite modelli. Dato un insieme di modelli T, l'algoritmo di apprendimento produce una funzione $f_T : V \longrightarrow \mathbb{R}$. Pertanto una regola di apprendimento è una mappa da un insieme di dati a una funzione su V. Siamo interessati al caso in cui V sia un grafo.
Il rischio empirico $R_k(f)$ (con il quadrato della funzione di perdita) ci quantifica quanto bene sia stato scelto il set di partenza:

$$R_k(f) = \frac{1}{k}\sum_{1}^{k}(f(x_i) - y_i)^2$$

L'errore di generalizzazione $R(f)$, invece, è il valore atteso di quanto approssimiamo bene su tutti i punti etichettati e non, cioè:

$$R(f) = \mathbb{E}_\mu(f(\mathbf{x}) - y(\mathbf{x}))^2$$

dove il valore atteso è rilevato in una distribuzione sottostante μ su $V \times \mathbb{R}$ secondo cui vengono disegnati gli esempi etichettati.
Denotiamo con λ_1 il più piccolo autovalore non banale della smoothness matrix S. Una interpretazione di λ_1 è che ci dà una stima di come V sia suddiviso. Ci aspettiamo che λ_1 sia relativamente grande, cioè $\lambda_1 > O\left(\frac{1}{n^r}\right), 0 \leq r \ll 1$. Per esempio, per un ipercubo n-dimensionale λ_1=2.
Il seguente teorema afferma che finché k è grande e i valori della soluzione del problema di regolarizzazione sono limitati, si ottiene una buona generalizzazione dei risultati. Notiamo che la costante K può essere limitata usando le proprietà del grafo.

2.1 Errori di generalizzazione sui dati

Teorema 1. Sia γ il parametro di regolarizzazione, T un insieme di $k \geq 4$ vertici $x_1, ..., x_k$, dove ogni vertice non si presenta più di t volte, insieme con i valori $y_1, ..., y_k$, tali che $|y_i| \leq M$. Sia f_T la soluzione di regolarizzazione usando lo smoothness functional S con il secondo più piccolo autovalore λ_1. Assumiamo che $\forall \mathbf{x}\, |f_T(x)| \leq K$ abbia

probabilità 1-δ (condizionata dal fatto che la molteplicità non sia più grande di t):

$$|R_k(f_T) - R(f_T)| \leq \beta + \sqrt{\frac{2\log\left(\frac{2}{\delta}\right)}{k}}(k\beta + (K+M)^2)$$

dove

$$\beta = \frac{3M\sqrt{tk}}{(k\gamma\lambda_1 - t)^2} + \frac{4M}{k\gamma\lambda_1 - t}$$

Dimostrazione. Per riuscire a dare una dimostrazione di questo teorema dobbiamo riscrivere la formula del teorema di Bousquet, Elisseeff e poi applicare il teorema di stabilità di regolarizzazione su grafi. Entrambi i teoremi verranno descritti in seguito. □

Questo teorema ci fa capire che se k è grande e i valori della soluzione al problema di regolarizzazione sono limitati, otteniamo una buon risultato.
Si nota che, come spesso accade nelle stime di generalizzazione dell'errore, questo decresce con un passo di $\frac{1}{\sqrt{k}}$. È importante notare che la stima è quasi indipendente dal numero totale di vertici n nel grafo. Diciamo *quasi* perchè la probabilità di punti con molteplicità aumenta con l'avvicinarsi di k a n e il valore di λ_1 può (o non può) dipendere implicitamente dal numero di vertici.

Ora procediamo dando due stime per K, una per il caso di una S generale e l'altra, più accurata, nel caso in cui la smoothness matrix sia la matrice laplaciana $S = L$. La costante K può essere stimata usando le proprietà del grafo.

Proposizione 1. Con λ_1, M e γ come sopra otteniamo la seguente disuguaglianza:

$$\|f\|_\infty \leq \frac{M}{\sqrt{\lambda_1\gamma}}$$

Dimostrazione. Denotiamo con $P(\mathbf{f})$ la quantità che vogliamo minimizzare:

$$P(\mathbf{f}) = \frac{1}{k}\sum_i (f_i - y_i)^2 + \gamma \mathbf{f}^T S \mathbf{f}$$

Osserviamo che quando $\mathbf{f} = 0, P(\mathbf{f}) = \frac{1}{k}\sum_i y_i^2 \leq M^2$. Pertanto se $\tilde{\mathbf{f}}$ minimizza $P(\mathbf{f})$, otteniamo $0 \leq \gamma\tilde{\mathbf{f}}^T L\tilde{\mathbf{f}} \leq M^2$. Ricordiamo che $f \in H$, dove H è lo spazio lineare dei vettori con media 0 e tale che il più piccolo autovalore di S limitato da H sia λ_1. Quindi, ricordando che $\|f\|_2 \geq \|f\|_\infty$, otteniamo

$$\tilde{\mathbf{f}}^T L\tilde{\mathbf{f}} \geq \lambda_1 \|f\|_2^2 \geq \lambda_1 \|f\|_\infty^2$$

Pertanto

$$\|f\|_\infty \leq \sqrt{\frac{\tilde{\mathbf{f}}^T L\tilde{\mathbf{f}}}{\lambda_1}} \leq \frac{M}{\sqrt{\lambda_1\gamma}}$$

□

Proposizione 2. Sia $W = \min_{i\sim j} w_{ij}$ il più piccolo peso diverso da zero del grafo G. Assumiamo che G sia connesso. Sia D il diametro non pesato del grafo, cioè la lunghezza massima del cammino più breve tra due nodi del grafo. Allora la massima componente di K della soluzione del problema di γ-regolarizzazione, con y limitato da M, soddisfa la seguente disequazione:

$$K \leq M\sqrt{\frac{D}{\gamma W}}$$

Un caso particolare della proposizione appena descritta è:

Corollario. Se tutti i pesi di G sono 0 o 1, allora

$$K \leq M\sqrt{\frac{D}{\gamma}}$$

Dimostrazione. Usando la notazione precendente, si vede che, sostituendo il vettore 0, se $\tilde{\mathbf{f}}$ minimizza $P(\mathbf{f})$, allora $P(\tilde{\mathbf{f}}) \leq M^2$.
Sia K la più grande componente di $\mathbf{f}$ con il vertice corrispondente v_1. Prendiamo un vertice v_2 tale che sia $y \leq 0$. Tale vertice esiste dal momento che i dati hanno media 0. Ora sia $e_1, e_2, ..., e_m$ una sequenza di archi del grafo che collegano v_1 e v_2. Poniamo $w_1, w_2, ..., w_m$ i pesi corrispondenti e siano $g_0, g_1, ..., g_m$ i valori di $\tilde{\mathbf{f}}$ corrispondenti ai vertici consecutivi di quel cammino. Ora sia $h_i = g_i - g_{i-1}$ la differenza di valori di $\tilde{\mathbf{f}}$ lungo il cammino. Otteniamo $\sum_i h_i = g_m - g_0 \geq K$.
Consideriamo il valore minimo Z di $\sum_i w_i h_i^2$, ricordando che $\sum_i h_i \geq K$. Usando il metodo dei moltiplicatori di Lagrange notiamo che la soluzione è $h_i = \frac{\alpha}{w_i}$. Troviamo α usando la condizione $\sum_i h_i = \alpha \sum_i \frac{1}{w_i} = K$. Pertanto

$$\sum_i w_i h_i^2 = \sum_i \frac{\alpha^2}{w_i} = \frac{K^2}{\sum_i \frac{1}{w_i}}$$

Visto che $\frac{m}{\sum_{i=1}^m \frac{1}{w_i}}$ è la media armonica dei w_i ed è più grande del $\min(w_1, ..., w_m)$, otteniamo

$$\sum_i w_i h_i^2 \geq \frac{K^2}{m} \min(w_1, ..., w_m)$$

Osserviamo anche che

$$\tilde{\mathbf{f}}^T L \tilde{\mathbf{f}} = \sum_{i<j,\ i\sim j} w_{ij}(\tilde{f}_i - \tilde{f}_j)^2 \geq \sum_i w_i h_i^2$$

poichè il termine di destra della disuguaglianza è una somma parziale del termine di sinistra. Pertanto

$$P(\tilde{\mathbf{f}}) \geq \frac{K^2}{m} \min(w_1, ..., w_m)$$

Ricordando che $P(\tilde{\mathbf{f}}) \leq M^2$, otteniamo:

$$K \leq \frac{M\sqrt{m}}{\sqrt{\gamma \min(w_1, ..., w_m)}}$$

Poiché il cammino tra questi punti può essere scelto arbitrariamente, possiamo sceglierlo in modo che la lunghezza del percorso m non superi il diametro non pesato D del grafo. In particolare, se tutti i pesi di G sono o 0 o 1 abbiamo:

$$K \leq \frac{M\sqrt{D}}{\sqrt{\gamma}}$$

assumendo chiaramente che G sia connesso. $\square$

Per dimostrare il teorema 1 utilizziamo il risultato di Bousquet e Elisseeff, ma prima diamo la definizione di stabilità di un algoritmo di apprendimento.

Definizione. Un algoritmo di apprendimento si dice $uniformemente$ ($o\, algoritmicamente$) $\beta - stabile$ se per qualsiasi coppia di training set T_1, T_2, che si differenzia al massimo per un punto, si ha:

$$\forall \mathbf{x} \qquad |f_{T_1}(\mathbf{x}) - f_{T_2}(\mathbf{x})| \leq \beta$$

Teorema (Bousquet, Elisseeff). Per un algoritmo β-stabile $T \to f_T$ si ha:

$$\forall \epsilon > 0 \qquad \text{Prob}(|R_k(f_T) - R(f_T)| > \epsilon + \beta) \leq 2\exp\left(-\frac{k\epsilon^2}{2(k\beta + (K+M))^2}\right)$$

Procediamo ora a mostrare che la regolarizzazione su grafi, usando lo smoothness functional S, è $\beta-$stabile, con β come nel Teorema 1.

Teorema (Stabilità di regolarizzazione su grafi). Per campioni di dati di dimensione $k \geq 4$ di molteplicità al più t, la $\gamma-$regolarizzazione, che usa lo smoothness functional S, è un algoritmo $\left(\frac{3M\sqrt{tk}}{(k\gamma\lambda_1 - t)^2} + \frac{4M}{k\gamma\lambda_1 - t}\right)$-stabile, assumendo che il denominatore $k\gamma\lambda_1 - t$ sia positivo.

Dimostrazione. Sia H l'iperpiano ortogonale al vettore $\mathbf{1} = (1, ..., 1)^T$. Denotiamo con P_H l'operatore corrispondente alla proiezione ortogonale su H. Ricordiamo che la soluzione del problema di regolarizzazione è data da:

$$(k\gamma S + I_k)\mathbf{f} = \tilde{\mathbf{y}} + \mu\mathbf{1}$$

dove μ viene scelta in modo tale che $\mathbf{f}$ appartenga ad H. Ordiniamo il grafo così che i punti etichettati siano i primi, allora la matrice diagonale I_k si scrive

$$I_k = diag(n_1, ..., n_l, 0..., 0)$$

dove l è il numero di vertici distinti etichettati del grafo e $n_i \leq t$ è la molteplicità dell'$i-$esimo data point. Il raggio spettrale di I_k è $\max(n_1, ..., n_l)$ e non è quindi maggiore di t. Notiamo che $l \leq k$.
D'altra parte, il più piccolo autovalore di S ristretto ad H è λ_1. Notando che H è invariante sotto S e che per qualsiasi vettore $\mathbf{v}$ si ha $\|P_H(\mathbf{v})\| \leq \|\mathbf{v}\|$ (poiché P_H è un

operatore di proiezione ortogonale), e usando la disuguaglianza triangolare, si ottiene immediatamente che per qualsiasi $\mathbf{f} \in H$

$$\|P_H(k\gamma S + I_k)\mathbf{f}\| \geq \|P_H k\gamma S\mathbf{f}\| - \|P_H I_k \mathbf{f}\| \geq (\lambda_1 \gamma k - t)\,\|\mathbf{f}\|$$

Ne consegue che il raggio spettrale dell'operatore inverso, quando limitato ad H, $(P_H(k\gamma S + I_k))^{-1}$, non supera $\frac{1}{\lambda_1\gamma k - t}$. (Naturalmente, l'inverso non è nemmeno definito al di fuori di H).
Per dimostrare la stabilità dobbiamo dimostrare che l'output dell'algoritmo non cambia molto quando cambiamo l'input esattamente in un punto del nostro insieme campione. Supponiamo che $\mathbf{y}, \mathbf{y}'$ siano i vettori di dati diversi in al più una componente. Possiamo assumere che $\mathbf{y}'$ abbia un nuovo punto. Scriviamo:

$$\mathbf{y} = (\sum_i y_{i1}, \sum_i y_{i2}, ..., \sum_i y_{il}, 0, ..., 0)$$

$$\mathbf{y}' = (\sum_i y_{i1}, \sum_i y_{i2}, ..., \sum_i{}' y_{il}, y_{l+1}, 0, ..., 0)$$

Le somme vengono fatte su tutti i valori di y che corrispondono ad un nodo del grafo. L'ultima somma $\sum_i{}'$ contiene un termine in meno rispetto alla somma corrispondente per $\mathbf{y}$.
Siano $\bar{y}, \bar{y}'$ le medie rispettivamente di $\mathbf{y}, \mathbf{y}'$. Notiamo che $|\bar{y} - \bar{y}'| \leq \frac{2M}{k}$ e che le componenti di $\tilde{\mathbf{y}}, \tilde{\mathbf{y}}'$ differiscono al massimo di due entrate, che variano al massimo di $2M + \frac{2M}{k}$. Naturalmente le ultime $n - l - 1$ componenti di entrambi i vettori sono uguali a zero. Dunque, assumendo $k \geq 4$, si ha:

$$\|\tilde{\mathbf{y}} - \tilde{\mathbf{y}}'\| \leq \sqrt{2\left(2M + \frac{2M}{k}\right)^2 + k\left(\frac{2M}{k}\right)^2} < 4M$$

Le soluzioni al problema di regolarizzazione $\mathbf{f}, \mathbf{f}'$ sono date dalle equazioni

$$\mathbf{f} = (P_H(\gamma k S + I_k))^{-1}\tilde{\mathbf{y}}$$

$$\mathbf{f}' = (P_H(\gamma k S + I_k^{'}))^{-1}\tilde{\mathbf{y}}'$$

dove I_k e I_k' sono matrici diagonali $n \times n$, $I_k = diag(n_1, n_2, ..., n_l, 0, ..., 0)$, $I_k^{'} = diag(n_1, n_2, ..., n_l - 1, 1, 0, ..., 0)$ e gli operatori sono ristretti all'iperpiano H.
Per accertare la stabilità, occorre stimare la differenza massima tra le componenti di $\mathbf{f}$ e $\mathbf{f}'$, cioè $\|\mathbf{f} - \mathbf{f}'\|_\infty$. Assumiamo il fatto che $\|\ \|_\infty \leq \|\ \|$.
Poniamo $A = P_H(\gamma k S + I_k), B = P_H(\gamma k S + I_k^{'})$, limitati all'iperpiano H. Otteniamo

$$\mathbf{f} - \mathbf{f}' = A^{-1}\tilde{\mathbf{y}} - B^{-1}\tilde{\mathbf{y}}' = A^{-1}(\tilde{\mathbf{y}} - \tilde{\mathbf{y}}') + A^{-1}\tilde{\mathbf{y}}' - B^{-1}\tilde{\mathbf{y}}'$$

Quindi

$$\|\mathbf{f} - \mathbf{f}'\|_\infty \leq \|\mathbf{f} - \mathbf{f}'\| \leq \left\|A^{-1}(\tilde{\mathbf{y}} - \tilde{\mathbf{y}}')\right\| + \left\|A^{-1}\tilde{\mathbf{y}}' - B^{-1}\tilde{\mathbf{y}}'\right\|$$

Dal momento che il raggio spettrale di A^{-1} e di B^{-1} è al massimo $\frac{1}{k\gamma\lambda_1 - t}$ e $\|\tilde{\mathbf{y}} - \tilde{\mathbf{y}}'\| \leq 4M$,

$$\left\|A^{-1}(\tilde{\mathbf{y}} - \tilde{\mathbf{y}}')\right\| \leq \frac{4M}{k\gamma\lambda_1 - t}$$

D'altra parte si può verificare che $\|\tilde{\mathbf{y}}'\| \leq 2\sqrt{tk}M$. Infatti si può facilmente vedere che la lunghezza è massimizzata quando la molteplicità di ogni punto è esattamente t. Notando che il raggio spettrale di $P_H(I_k - I_k{}')$ non può superare $\sqrt{2} < 1.5$, si ottiene:

$$\left\|A^{-1}\tilde{\mathbf{y}}' - B^{-1}\tilde{\mathbf{y}}'\right\| = \left\|B^{-1}(B-A)A^{-1}\tilde{\mathbf{y}}'\right\| = \left\|B^{-1}P_H(I_k - I_K^{'})A^{-1}\tilde{\mathbf{y}}'\right\| \leq \frac{3M\sqrt{tk}}{(k\gamma\lambda_1 - t)^2}$$

Quindi

$$\|\mathbf{f} - \mathbf{f}'\|_\infty \leq \frac{3M\sqrt{tk}}{(k\gamma\lambda_1 - t)^2} + \frac{4M}{k\gamma\lambda_1 - t}$$

□

I due teoremi sopra citati con $\beta = \left(\frac{3M\sqrt{tk}}{(k\gamma\lambda_1 - t)^2} + \frac{4M}{k\gamma\lambda_1 - t}\right)$ dimostrano il Teorema 1. Tipicamente ci si aspetterebbe $\frac{2M\sqrt{tk}}{(k\gamma\lambda_1 - t)^2} \ll \frac{4M}{k\gamma\lambda_1 - t}$. Tuttavia un problema rimane ancora irrisolto, cioè avere molteplicità elevate è piuttosto improbabile finchè $k \ll n$ e la distribuzione è simile a quella uniforme.

Diamo ora un'ulteriore stima dell'errore di generalizzazione.

Teorema 2. Utilizzando tecniche di regolarizzazione con $\|k\|_H \leq \kappa$ e $(f(x) - y)^2 \leq M$,

$$R(f_T) \leq R_k(f_T) + \frac{4M\kappa}{k\gamma} + 4M\sqrt{\left(\frac{2\kappa^2}{\gamma^2} + \frac{4\kappa}{\gamma} + 2\right)\frac{\ln(\frac{2}{\delta})}{k}}$$

e

$$R(f_T) \leq R_k(f_T) + 2M\sqrt{\left(\frac{64\kappa}{\gamma} + 2\right)\frac{1}{k\delta}}$$

Dimostrazione. Denotiamo con f_T l'elemento che minimizza C, dove $C(f) = \frac{1}{k}\sum_{i=1}^{k}(f(x_i) - y_i)^2) + \gamma\|f\|_H^2$ Definiamo

$$C^j(f) = \frac{1}{k}\sum_{i\neq j}^{k}(f(x_i) - y_i)^2 + \frac{1}{k}(f(x_j') - y_j')^2 + \gamma\|f\|_H^2$$

Sia ${f_T}^j$ l'elemento che minimizza C^j e denotiamo con g la differenza ${f_T}^j - f_T$. Per $t \in [0,1]$ otteniamo

$$C(f_T) - C(f_T + tg) = -\frac{2t}{k}\sum_{i=1}^{k}(f_T(x_i) - y_i)g(x_i) - 2t\gamma < f_T, g > + t^2A(g)$$

dove $A(g)$, che non è scritto esplicitamente, qui è il fattore di t^2. Similmente otteniamo

$$C^j({f_T}^j) - C^j({f_T}^j - tg) = \frac{2t}{k}\sum_{i\neq j}({f_T}^j(x_i) - y_i)g(x_i) + \frac{2t}{k}({f_S}^j(x_j') - y_j')g(x_j') + 2t\gamma < {f_S}^j, g > + t^2A^j(g)$$

Ottimizzando otteniamo

$$C(f_T) - C(f_T + tg) \leq 0 \quad \text{e} \quad C^j({f_T}^j) - C^j({f_T}^j - tg) \leq 0$$

Dunque, sommando queste disuguaglianze, dividendo per $\frac{t}{k}$ e facendo il limite per $t \to 0$ otteniamo

$$2\sum_{i\neq j} g(x_i)^2 - 2(f_T(x_j) - y_j)g(x_j) + 2(f_T{}^j(x'_j) - y'_j)g(x'_j) + 2k\gamma \|g\|_H^2 \leq 0$$

e quindi

$$k\gamma \|g\|_H^2 \leq (f_T(x_j) - y_j)g(x_j) - (f_T(x'_j) - y'_j)g(x'_j) \leq 2\sqrt{M}_k \|g\|_H$$

Ricordando che $|f(x)| \leq \|f\|_H \|k\|_H \leq \kappa \|f\|_H$, otteniamo

$$\left\|f_T{}^j - f_T\right\|_H \leq 2\sqrt{M}\kappa/(k\gamma)$$

e quindi

$$\forall x, y \left|(f_T(x) - y)^2 - (f_T{}^j(x) - y)^2\right| \leq 2\sqrt{M}\left|f_T(x) - f_T{}^j(x)\right| \leq 4M\kappa/(k\gamma)$$

□

Abbiamo quindi dimostrato che la minimizzazione di $C(f)$ è una procedura $\frac{4Mk}{k\gamma}$-stabile.

Notiamo che l'assunzione sulla liscezza del kernel $\|k\|_H \leq \kappa$ fa ottenere una convergenza diversa e più semplice di quella del Teorema 1. Nel teorema sopra citato abbiamo assunto che la funzione simmetrica del kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ abbia la seguente proprietà:

$$|f(x)| \leq \|f\|_H \|k\|_H \leq k \|f\|_H$$

con $f : \mathbb{R}^d \to \mathbb{R}$ e H uno spazio che minimizza

$$C(f) = \frac{1}{m}\sum_{j=1}^{m}(f(x_j) - y_j)^2 + \lambda \|f\|_H^2$$

dove $\|f\|_H^2$ è la norma nello spazio H.

Capitolo 3

Esperimenti numerici

Riscriviamo l'algoritmo di Tikhonov del Capitolo 1 come segue:

$$min_{f \in H_K} \frac{1}{l} \sum_{i=1}^{l} (y_i - f(x_i))^2 + \gamma \|f\|_K^2$$

dove K indica, in modo ovvio a seconda del contesto, la funzione del kernel o la matrice di Gram. l sono i campioni etichettati, mentre u sono quelli non etichettati. Nel nostro caso

$$\|f\|_K^2 = f^T K^{-1} f$$

Prima descriviamo l'algoritmo che andremo ad utilizzare. Vengono dati in input l campioni etichettati, cioè $\{(x_i, y_i)\}_{i=1}^{l}$ e u campioni non etichettati $\{x_i\}_{j=l+1}^{l+u}$. L'output dell'algoritmo è il segno della funzione $f^* : \mathbb{R}^n \to \mathbb{R}$. L'algoritmo svolge quattro step:

- Sceglie una funzione del kernel $K(x, y)$ e calcola la relativa matrice di Gram $K_{ij} = K(x_i, y_i)$;
- Sceglie γ;
- Calcola α^*;
- Dà in output il $sign(f^*)$, dove la funzione $f^*(x) = \sum_{j=i}^{l+u} \alpha_i^* K(x_i, x)$.

Sostituendo $f^*(x) = \sum_{j=i}^{l+u} \alpha_i^* K(x_i, x)$ nel problema di minimizzazione posto sopra otteniamo:

$$\alpha^* = \text{argmin} \frac{1}{l} (Y - K\alpha)^T (Y - K\alpha) + \gamma \alpha^T K \alpha$$

dove α è una variabile l-dimensionale, cioè $\alpha = [\alpha_1, ..., \alpha_l]^T$
Derivando otteniamo:

$$\frac{1}{l} (Y - K\alpha^*)^T (-K) + \gamma K \alpha^* = 0$$

α^* calcolato nel terzo step dell'algoritmo è quindi:

$$\alpha^* = (K + \gamma l I)^{-1} Y$$

dove K è la $l \times l$ matrice di Gram $K_{ij} = K(x_i, x_j)$, Y è un vettore etichettato l-dimensionale, cioè $Y = [y_1, ..., y_l]^T$.

In questa trattazione K è stato scelto in modo tale che $K(x_i, y_j) = W_{ij}$, in questo modo la matrice di smoothness del Capitolo 1 è:

$$S = K^{-1}$$

Negli esperimenti numerici, trattati successivamente, la condizione $\mathbf{f} \perp \mathbf{1}$ o equivalentemente $\mu = 0$ delle equazioni 1.1 e 1.2 non è più necessaria perchè la matrice di kernel scelta è invertibile.

Per l'algoritmo di Tikhonov del Capitolo 1 abbiamo $\gamma \geq 0$, mentre per l'interpolazione si ha $\gamma = 0$.

Eseguiamo alcuni test numerici e analizziamo i risultati ottenuti. Scegliendo il dataset delle due lune, che è un insieme di punti in due dimensioni che formano due semicerchi intrecciati, possiamo, grazie ai codici oppurtunatamente modificati da [3], variare i parametri di γ e dei punti etichettati e confrontare i risultati. Scegliamo due parametri per γ: $\gamma = 0$ (per l'interpolazione) e $\gamma = 100$.

Nei grafici sottostanti la regione azzurra indica dove la funzione assume valori uguali a 1, mentre la regione bianca contrassegna i punti in cui il valore della funzione è pari a -1. I punti etichettati sono rappresentati dai rombi rossi e dai cerchi blu, mentre i dati non etichettati sono raffigurati con i quadrati neri vuoti.

Per 8 punti etichettati otteniamo che il classificatore migliore si ha per $\gamma = 0$, quindi l'interpolazione è la scelta migliore, come si può osservare dai grafici in Figura 3.1 .

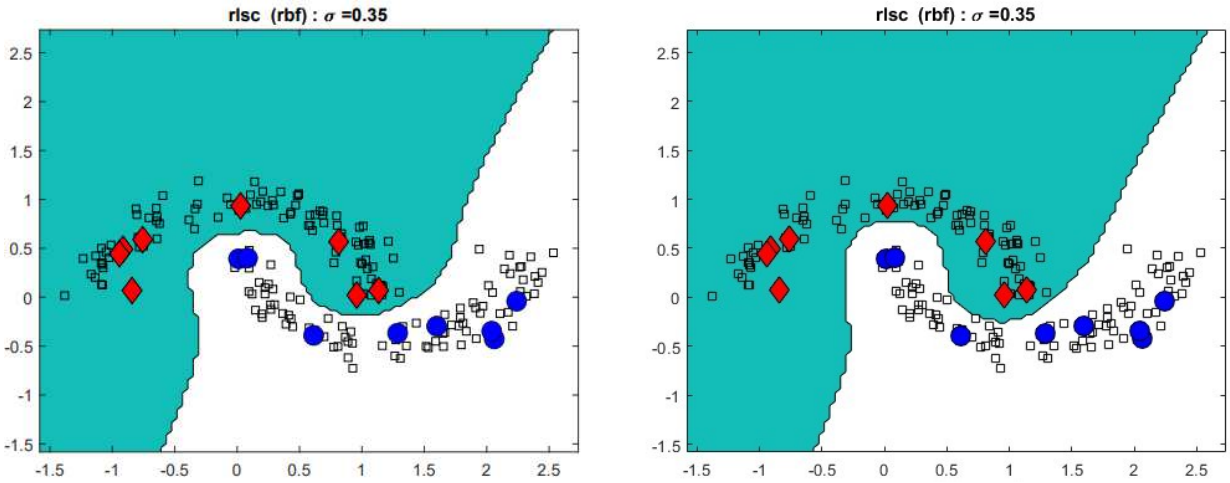

Figura 3.1: A sinistra l'interpolazione con $\gamma = 0$, a destra regolarizzazione di Tikhonov con $\gamma = 100$ con 8 punti etichettati.

Si vede chiaramente che per $\gamma = 100$ si ha un errore peggiore, infatti l'errore compiuto in questo caso è dell' 1%.

Consideriamo ora 40 punti etichettati; in questo caso si verifica che la scelta migliore ricade sulla regolarizzazione di Tikhonov, infatti con l'interpolazione la funzione non può essere ben definita perchè l'errore è del 3%. I due esempi sono mostrati in Figura 3.2

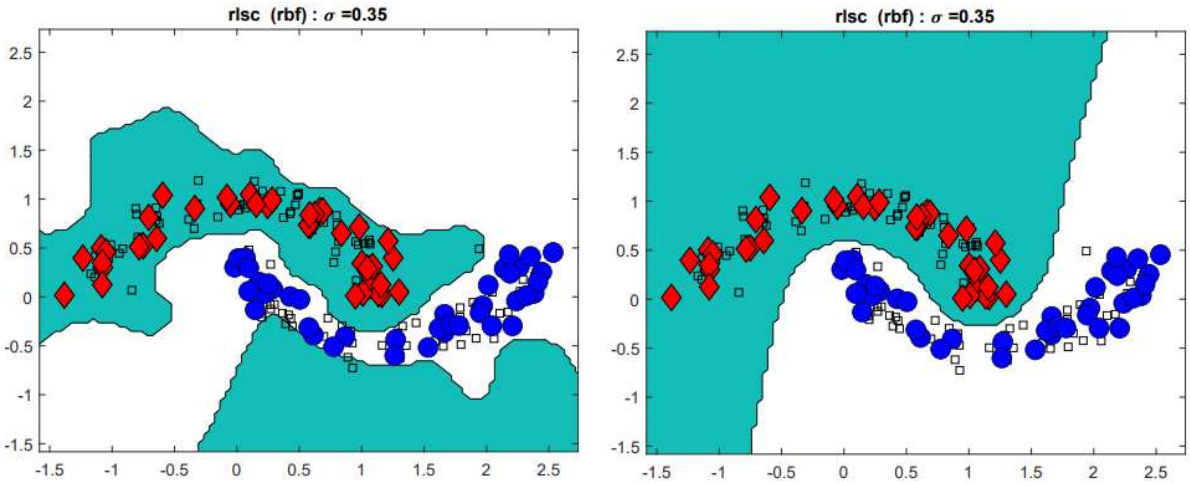

Figura 3.2: A sinistra interpolazione con $\gamma = 0$, a destra regolarizzazione di Tikhonov con $\gamma = 100$ con 40 punti etichettati.

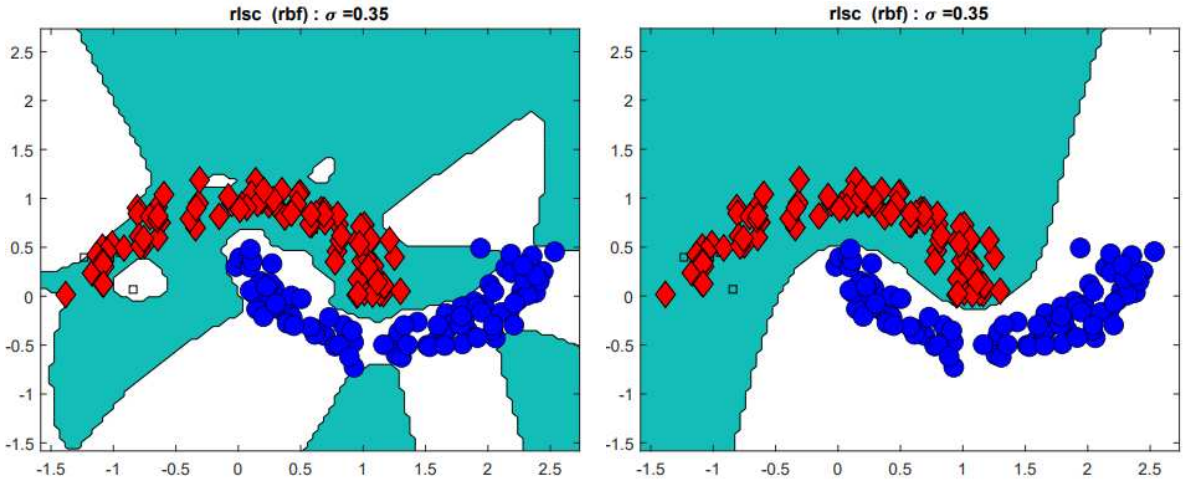

Figura 3.3: A sinistra interpolazione con $\gamma = 0$, a destra regolarizzazione di Tikhonov con $\gamma = 100$ con 96 punti etichettati.

Scegliendo il numero massimo di punti etichettati, cioè 96, i due algoritmi non sono ottimali. In questo caso infatti la funzione non può essere ben definita. Si nota quindi che, aumentando i punti etichettati, l'interpolazione non è efficace. Al contrario, con pochi punti etichettati risulta essere la scelta migliore.
In Figura 3.3 si nota che la scelta di $\gamma = 0$ produce un errore dell' 8.5%. Grazie ai codici riportanti in Appendice A, riusciamo a trovare il paramentro di regolarizzazione

ottimale (in Figura 3.4).

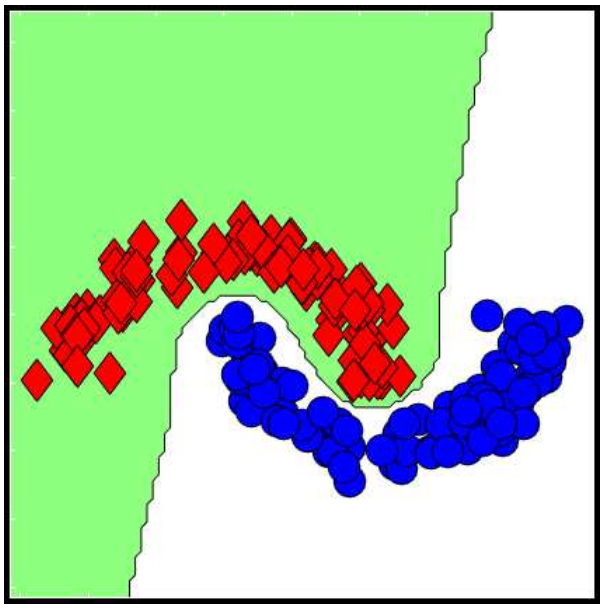

Figura 3.4: Regolarizzazione ottimale con 96 punti etichettati ($\gamma \approx 0.23$).

Conclusioni

Grazie agli esperimenti numerici svolti nel capitolo precedente possiamo trarre le seguenti conclusioni: disporre di un paramentro di regolarizzazione modificabile dall'utente non implica necessariamente una regolarizzazione migliore. Il metodo di Tikhonov, quindi, non è sempre il più efficace, in particolare per un numero di punti etichettati relativamente piccolo potrebbe risultare una scelta svantaggiosa.
L'interpolazione, al contrario, sembra essere svantaggiosa nel caso si consideri un numero di dati etichettati grande rispetto al dataset scelto, ma molto più performante con pochi punti etichettati dell'algoritmo di Tikhonov. Tuttavia è difficile scegliere a priori quale metodo utilizzare tra i due a causa della scelta del dataset. Si vede, però, che entrambi risultano essere poco efficaci se i punti etichettati aumentano.

Appendice A: Codici Matlab

Riportiamo di seguito i codici Matlab utilizzati nel Capitolo 3, tratti da [3].

Il seguente codice è stato utilizzato per ottenere l'ottimizzazione su 96 punti etichettati rispetto al parametro γ della formula dell'algoritmo di Tikhonov proposta nel Capitolo 3.

```
function varargout = demo(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @demo_OpeningFcn,
                      ...
                   'gui_OutputFcn',  @demo_OutputFcn,
                      ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
       varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

function demo_OpeningFcn(hObject, eventdata, handles,
   varargin)

handles.output = hObject;
handles.data = load('2moons.mat');
handles.mode=2; % fully supervised
handles.kernel='rbf';
handles.kernelparam=0.35;
handles.algo='rlsc';
handles.lab=2;
handles.generate=1;
handles.Y=handles.data.y;
set(handles.radiobutton2,'Value',1);
set(handles.lambda1,'Value', 0.4511); handles.gamma_A
   =0.014362;
set(handles.gamma_A_value,'String',num2str(handles.
   gamma_A));
```

```
set(handles.lambda2,'Value',0.7852); handles.gamma_I
    =0.7852;
set(handles.gamma_I_value,'String',num2str(handles.
    gamma_I));
guidata(hObject, handles);

function varargout = demo_OutputFcn(hObject, eventdata,
    handles)
varargout{1} = handles.output;

function lambda1_CreateFcn(hObject, eventdata, handles)
usewhitebg = 0;
if usewhitebg
     set(hObject,'BackgroundColor',[.9 .9 .9]);
else
     set(hObject,'BackgroundColor',get(0,'
        defaultUicontrolBackgroundColor'));
end


% --- Executes on slider movement.
function lambda1_Callback(hObject, eventdata, handles)
p=get(hObject,'Value');
handles.gamma_A=10^(7*p-5)-10^(-5);
set(handles.gamma_A_value,'String',num2str(handles.
    gamma_A));
guidata(hObject,handles);

% --- Executes during object creation, after setting all
     properties.
function lambda2_CreateFcn(hObject, eventdata, handles)
usewhitebg = 1;
if usewhitebg
     set(hObject,'BackgroundColor',[.9 .9 .9]);
else
     set(hObject,'BackgroundColor',get(0,'
        defaultUicontrolBackgroundColor'));
end


% --- Executes on slider movement.
function lambda2_Callback(hObject, eventdata, handles)
p=get(hObject,'Value');
handles.gamma_I=10^(7*p-5)-10^(-5);
set(handles.gamma_I_value,'String',num2str(handles.
    gamma_I));
guidata(hObject,handles);

% --- Executes during object creation, after setting all
     properties.
function select_algo_CreateFcn(hObject, eventdata,
    handles)
if ispc
     set(hObject,'BackgroundColor','white');
else
     set(hObject,'BackgroundColor',get(0,'
        defaultUicontrolBackgroundColor'));
end
```

```
% --- Executes on selection change in select_algo.
function select_algo_Callback(hObject, eventdata,
    handles)
contents = get(hObject,'String');
handles.algo=contents{get(hObject,'Value')};
guidata(hObject,handles);

% --- Executes during object creation, after setting all
    properties.
function enter_kernelparams_CreateFcn(hObject, eventdata
    , handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'
        defaultUicontrolBackgroundColor'));
end



function enter_kernelparams_Callback(hObject, eventdata,
    handles)
handles.kernelparam=str2double(get(hObject,'String'));
guidata(hObject,handles);

% --- Executes on button press in run.
function run_Callback(hObject, eventdata, handles)
X=handles.data.x;
Y=handles.data.y;
handles.mode

options.NN=6;
options.gamma_A=handles.gamma_A;
options.gamma_I=handles.gamma_I;
options.Kernel=handles.kernel;
options.KernelParam=handles.kernelparam;
options.GraphDistanceFunction='euclidean';
options.GraphWeights='binary';
options.GraphNormalize=1;
options.GraphWeightParam=1;

method=handles.algo;

 % semi-supervised

gen=handles.generate;
if gen==1

    ipos=find(Y>0);
    ineg=find(Y<0);
    lab=handles.lab;
    rpos=randperm(length(ipos)); rneg=randperm(length(
        ineg));
    Y1=zeros(length(Y),1);
    Y1(ipos(rpos(1:lab)))=1;
    Y1(ineg(rneg(1:lab)))=-1;
```

```
    handles.generate=0;

    clear y1;
    load l.mat;

    handles.Y=Y1;

else
    handles.generate=0;
end

classifier=ml_train(X,handles.Y,options,method);

xmin=min(X(:,1)); ymin=min(X(:,2)); rmin=min(xmin,ymin)
   -0.2;
xmax=max(X(:,1)); ymax=max(X(:,2)); rmax=max(xmax,ymax)
   +0.2;
steps=(rmax-rmin)/100;
xrange=rmin:steps:rmax;
yrange=rmin:steps:rmax;

plotclassifiers(classifier,xrange,yrange); title(method,
   'Color','w');
hold on;h=gca; set(h,'Xcolor','w'); set(h,'Ycolor','w')
unlab=find(handles.Y==0);
plot2D(X(unlab,:),handles.Y(unlab),10); hold on;
lab=find(handles.Y);
plot2D(X(lab,:),handles.Y(lab),17);
hold off;
guidata(hObject,handles);


% --- Executes during object creation, after setting all
    properties.
function select_dataset_CreateFcn(hObject, eventdata,
   handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'
       defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in select_dataset.
function select_dataset_Callback(hObject, eventdata,
   handles)
contents = get(hObject,'String');
dataset=contents{get(hObject,'Value')};
handles.data=load([dataset '.mat']);
guidata(hObject,handles);



% --- Executes during object creation, after setting all
    properties.
function numlabeled_CreateFcn(hObject, eventdata,
   handles)
```

```
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'
       defaultUicontrolBackgroundColor'));
end



function numlabeled_Callback(hObject, eventdata, handles
   )
handles.lab=str2double(get(hObject,'String'));
guidata(hObject,handles);


% --- Executes during object creation, after setting all
    properties.
function rbfwidth_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'
       defaultUicontrolBackgroundColor'));
end



function rbfwidth_Callback(hObject, eventdata, handles)
handles.kernelparam=str2double(get(hObject,'String'));
guidata(hObject,handles);

% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata,
   handles)
off = [handles.radiobutton2,handles.radiobutton3];
mutual_exclude(off)
handles.mode=1; % fully supervised
guidata(hObject,handles);
% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata,
   handles)
off = [handles.radiobutton1,handles.radiobutton3];
mutual_exclude(off)
handles.mode=2; % unsupervised
guidata(hObject,handles);
% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata,
   handles)
off = [handles.radiobutton1,handles.radiobutton2];
mutual_exclude(off)
handles.mode=3; % semisupervised
guidata(hObject,handles);

function mutual_exclude(off)
set(off,'Value',0)


% --- Executes on button press in generate.
```

```
function generate_Callback(hObject, eventdata, handles)
handles.generate=1;
guidata(hObject,handles);


% --- Executes during object creation, after setting all
    properties.
function gamma_A_value_CreateFcn(hObject, eventdata,
   handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'
       defaultUicontrolBackgroundColor'));
end



function gamma_A_value_Callback(hObject, eventdata,
   handles)
% --- Executes during object creation, after setting all
    properties.
function gamma_I_value_CreateFcn(hObject, eventdata,
   handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'
       defaultUicontrolBackgroundColor'));
end

function gamma_I_value_Callback(hObject, eventdata,
   handles)
```

La seguente routine Matlab genera i grafici e i relativi errori di regolarizzazione. Inoltre è stata utilizzata per ottenere il miglior classificatore tra i parametri scelti, cioè $\gamma = 0$ e $\gamma = 100$.

```
function [X,error]=experiment_moon(X,Y,XT,YT,method,q,s)
   ;

% 2 Moons Experiment

options=ml_options('gamma_A',0.1, 'NN',6, 'Kernel','rbf'
   ,'KernelParam',0.35);


%q=-1:5;
if nargin==6 % perform a search over lambda1 and lambda2
    lambda1=2.^q;
    lambda2=2.^q;
else % interpret q as lambda1 and s as lambda2

    lambda1=q;
    lambda2=s;

end
```

```
% best SVM
min_err=100;

if strcmp(method,'svm') | strcmp(method,'tsvm')  strcmp(
   method,'rlsc')
% optimize over just 1 parameter
for i=1:length(q)
    options.gamma_A=lambda1(i);
    options.gamma_I=0;
    classifier=ml_train(X,Y,options, method);
    [f,labels,error]=ml_test(classifier,XT,YT);
    [lambda1(i) error min_err]
    if error < min_err
         min_err=error;
         best_classifier=classifier;
    end

end



else % optimize over both

   for i=1:length(q)
      for j=1:length(q)
            options.gamma_A=lambda1(i);
            options.gamma_I=lambda2(j);
            classifier=ml_train(X,Y,options, method);
            [f,labels,error]=ml_test(classifier,XT,YT);
            [lambda1(i) error min_err]
    if error < min_err
         min_err=error;
         best_classifier=classifier;
    end
end
end

end
lab=find(Y);
xmin=min(X(:,1)); ymin=min(X(:,2)); rmin=min(xmin,ymin)
   -0.2;
xmax=max(X(:,1)); ymax=max(X(:,2)); rmax=max(xmax,ymax)
   +0.2;
steps=(rmax-rmin)/100;
xrange=rmin:steps:rmax;
yrange=rmin:steps:rmax;
plotclassifiers(best_classifier, xrange, yrange);
```

Questo codice utilizza lo script precedente e, cambiando i punti etichettati (denotati con l), permette di ottenere i tre test eseguiti nel Capitolo 3.

```
makefigure_moon


load 2moons.mat;
```

```
l=8; % number of labeled examples
%l=40;
%l=96;

for i = 1:100
pos=find(y==1);
neg=find(y==-1);
ipos=randperm(length(pos));
ineg=randperm(length(neg));
y1=zeros(length(y),1);
y1(pos(ipos(1:l)))=1;
y1(neg(ineg(1:l)))=-1;



[a,error(i)]=experiment_moon(x,y1,xt,yt,'rlsc',0,0);
end

plot2D(x,y1,12);
```

Bibliografia

[1] Mikhail Belkin, Irina Matveeva, and Partha Niyogi. Regularization and semi-supervised learning on large graphs. In *Learning Theory: 17th Annual Conference on Learning Theory, COLT 2004, Banff, Canada, July 1-4, 2004. Proceedings 17*, pages 624–638. Springer, 2004.

[2] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. *On manifold regularization, ManifoldLearn*, 2004. `http://manifold.cs.uchicago.edu/manifold_regularization/manifold.html`.

[3] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(17-22), 2006.

[4] Olivier Bousquet and André Elisseeff. Algorithmic stability and generalization performance. *Advances in Neural Information Processing Systems*, 13(5-6), 2000.

[5] Alex J Smola and Bernhard Schölkopf. *Learning with kernels*, volume 4. MIT Press Ltd, 1998.