Support Vector Regression using Deflected Subgradient Methods

Elia Piccoli
Nicola Gugole

January 29, 2021

A project presented for the
Computational Mathematics for Learning and Data Analysis
course

University of Pisa
Artificial Intelligence
A.Y. 2020/2021

Contents

Abstract

Project aim is developing the implementation of a model which follows an SVR-type approach including various different kernels. The implementation uses as optimization algorithm a dual approach with appropriate choices of the constraints to be dualized, where the Lagrangian Dual is solved by an algorithm of the class of deflected subgradient methods.

1 Introduction

Per affrontare questo problema di regressione ci vogliamo affidare ad un modello di apprendimento supervisionato che è il Support Vector Regression. SVR ha come obiettivo trovare una funzione tale per cui ogni record assegnatoci per il training non devii da essa più di ε (per questo ogni valore all'interno del cosiddetto ε-tube non viene considerato come errore nella fase di ottimizzazione, rendendo la loss del modello ε-insensitive). Per fare ciò abbiamo bisogno di un certo parametro C (per capire il livello di regolarizzazione che desideriamo) ed un valore ε (per esprimere l'errore che accettiamo), oltre ad eventuali parametri necessari ad attuare i kernel (e.g. gamma per quanto riguarda il kernel RBF). Parte fondante del modello, oltre a ciò sopra descritto riguardo l' ε-tube, è dare allo stesso tempo importanza al mantenere la funzione *as flat as possible*, per evitare overfitting ed avere dunque un modello che sia un corretto tradeoff tra accuratezza e generalità.

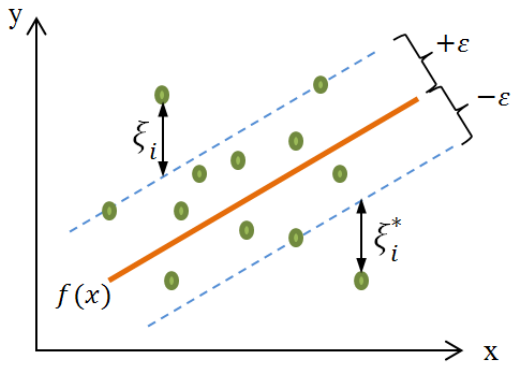

Figure 1: a generic svr

La funzione risultante dall'ottimizzazione del modello è descritta genericamente come:

$$f(x) = wx + b \tag{1}$$

Obiettivo dell'ottimizzazione è dunque fare in modo che la curva sia, di nuovo, *as flat as possible*, ma questo è equivalente ad un problema di ottimizzazione dove vogliamo avere $\|w\|$ minima. Per comodità di formulazione del problema possiamo minimizzare $\|w\|^2$ senza cambiare il significato. Questo

ci permette di portarci in un problema di ottimizzazione quadratico, grazie al quale potremo approfittare del concetto di **strong duality** più tardi. Introduciamo a questo punto delle variabili dette *slack* per formulare la *dual objective function*, la quale rappresenta il nostro *primal problem*:

$$\min_{w,b,\xi_i,\xi_i^*} \frac{1}{2}\|w\|^2 + C\sum_i(\xi_i + \xi_i^*) \tag{2}$$

Ciò che viene sommato a $\|w\|$ è un elemento che ci permette di regolare l'errore, e di conseguenza la penalità, dovuti alla possibile presenza di elementi che non rimangono all'interno dell' ε-tube. Vediamo dunque come C funga da regolarizzatore in una metodica simile a L1. I vari ξ vengono detti *slack variables* e ci permettono di definire i vincoli del problema per qualsiasi i-esimo dato:

$$\begin{aligned} &y_i - w^T\phi(x_i) - b \leq \epsilon + \xi_i, \\ &b + w^T\phi(x_i) - y_i \leq \epsilon + \xi_i, \\ &\xi_i, \xi_i^* \geq 0 \end{aligned} \tag{3}$$

x_i input, y_i output

Essendo in un problema di ottimizzazione quadratico, la soluzione al *dual problem* risulta equivalente a quella del *primal problem*. In particolare in questa casistica risulta più facile la risoluzione del *dual problem* vista la possibile applicazione del concetto di kernel. Costruiamo dunque il *dual problem* definendo la relativa Lagrangiana (equivalente del *primal problem* al quale aggiungiamo i vincoli sommandoli o sottraendoli):

$$\begin{aligned} L(\alpha, \alpha^*, \mu, \mu^*) = &\frac{1}{2}\|w\|^2 \\ &+ C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \\ &+ \sum_{i=1}^{m}(\alpha_i(y_i - w^T\phi(x_i) - b - \epsilon - \xi_i)) \\ &+ \sum_{i=1}^{m}(\alpha_i^*(w^T\phi(x_i) + b - y_i - \epsilon - \xi_i^*)) \\ &- \sum_{i=1}^{m}(\mu_i\xi_i + \mu_i^*\xi_i^*) \end{aligned} \tag{4}$$

A causa del concetto di ***weak duality*** qualsiasi valore del *primal problem* risulta maggiore (o uguale) del *dual problem*. Da questa considerazione deriviamo il fatto che il punto di massima vicinanza tra i due problemi è dove il *primal problem* ha minimo e il *dual problem* ha massimo. Nella casistica di *strong duality* questa vicinanza si tramuta in uguaglianza. Cerchiamo dunque di rielaborare il *dual problem* per lavorare con meno variabili possibili, in particolare eliminiamo dai calcoli la variabile w, b e le 2 *slack* ξ e ξ^*. In che modo? Stiamo cercando un valore ottimo che tra l'altro sarà unico data la casistica quadratica, dunque la derivata parziale relativa ad ognuna di queste variabili va posta a 0. Questo ci porta a poter ridefinire w:

$$w = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*)\phi(x_i) \tag{5}$$

Infine tramite sostituzione nella Lagrangiana definita precedentemente arriviamo ad una funzione dipendente solamente da α e α^*:

$$\begin{aligned} \max_{\alpha_i, \alpha_i^*} & -\frac{1}{2}\sum_i \sum_j (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(x_i, x_j) \\ & - \epsilon \sum_i (\alpha_i + \alpha_i^*) \\ & + \sum_i y_i(\alpha_i - \alpha_i^*) \end{aligned} \tag{6}$$

Notiamo che le uniche variabili rimaste sono i moltiplicatori lagrangiani α e α^*, i quali sono sottoposti ai seguenti vincoli:

$$\begin{aligned} &\forall i\ \alpha_i, \alpha_i^* \geq 0 && (KKT condition) \\ &\forall i\ \alpha_i, \alpha_i^* \in [0, C] && (from\ deriving\ (6)) \\ &\forall i\ \sum(\alpha_i - \alpha_i^*) = 0 && (from\ deriving\ (6)) \\ &\forall i\ \alpha_i \alpha_i^* = 0 && (from\ model\ construction) \end{aligned} \tag{7}$$

Restando ancora su (6), approfondiamo l'elemento $K(x_i, x_j)$, ovvero ciò rappresenta il concetto di **kernel**, sostituendo ciò che sarebbe un prodotto scalare nello spazio necessitato. Rielaboriamo dunque la precedente frase: certi problemi di regressione non possono essere adeguatamente descritti da modelli lineari. Risulta dunque comodo cambiare spazio di visualizzazione, per portarci in un nuovo spazio (con ogni probabilità con più dimensioni rispetto allo spazio originale) dove il problema diventa linearmente

affrontabile. Il cambio di base risulta essere incredibilmente dispendioso per il training del modello ed è proprio in questo caso che il kernel diventa il fulcro dell'efficienza di SVC/SVR. Ci permette infatti di eseguire il *dot-product* nello spazio attuale ma avere come risultato il *dot-product* nello spazio richiesto. Ci permette di risparmiare molte computazioni e di poterle riutilizzare salvandoci i valori risultanti in una *kernel matrix* (riutilizzabile ad ogni ciclo di ottimizzazione). Ovviamente non c'è la certezza di aver preso in considerazione il kernel corretto per la casistica, bisognerà dunque svilupparne vari, magari certi funzioneranno meglio di altri (*rbf, sigmoid, etc*).
In particolare questa implementazione prevede il possibile utilizzo di diversi kernel:

- linear: $\langle x, x' \rangle$
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$
- rbf: $\exp(-\frac{\|x-x'\|^2}{2\sigma^2})$
- sigmoid: $\tanh(\gamma \langle x, x' \rangle + r)$

Definito il problema possiamo a questo punto cercare il massimo del *dual problem*. La task da noi scelta utilizzerà **deflected subgradient methods** per raggiungere l'obiettivo. Una volta raggiunto il massimo avremo a nostra disposizione i corretti α e α^* necessari per il calcolo di w.

Per completare la funzione risolutrice del problema ci basta trovare il parametro b. Ricavarlo risulta semplice se prendiamo in considerazione un qualsiasi elemento del nostro insieme di input tale per cui la relativa predizione y_i sia al limite dell' ε-tube o al di fuori di esso. Proprio quell'insieme di valori sarà infatti l'unico ad avere come vincoli attivi almeno uno tra i vari α e α^*, ovvero un α o α^* diversi da 0. Essendo dunque al margine di un vincolo potremo, derivando da (1) e (5):

$$x_j \; with \; \alpha_j \in (0, C) : b = y_j - \sum_i (\alpha_i - \alpha_i^*) K(x_i, x_j) \tag{8}$$

Una volta trovata la funzione risultante possiamo testare la bontà del modello effettuando *prediction* su un set di test input per poi calcolare la metrica MSE tra output previsto dal modello e output effettivo del set di test input.
Questo fatto è particolarmente importante per il confronto tra modelli e dunque per la ricerca dei parametri migliori possibili (*grid search*).

2 Basic Algorithms Introduction

Come richiesto dalla task utilizzeremo un *subgradient method*, il quale viene utilizzato per trattare funzioni che possono anche avere punti di *nondifferenziabilità*. E' dunque uno dei metodi più semplici e meno esigenti nei confronti di proprietà necessarie per f. In particolare si distingue dagli ordinari metodi di gradiente per due aspetti:

- Lo *stepsize* non viene scelto tramite *line search* ma tramite altri ragionamenti (il più immediato tra tutti è **DSS**, dove lo stepsize viene fissato a priori con degli appropriati vincoli di scelta)
- Non è un *descent method*, ovvero il valore della funzione può incrementare nelle iterazioni. Questo avviene per il motivo che non tutti i subgradient portano a direzioni di discesa. Anche se questo è vero abbiamo provato durante il corso che iterazioni successive ci portano ad avvicinarci a x^* monotonicamente scelta una corretta *stepsize*. Con questa certezza sappiamo che convergeremo lentamente ma sicuramente ad x^* e di conseguenza anche ad f^* (anche se probabilmente non monotonicamente).

Entrando più nel dettaglio per quanto riguarda lo *stepsize*, la scelta di fissarlo a priori (DSS) ci porta ad una convergenza estremamente lenta, dunque preferiremmo utilizzare altre euristiche. In particolare, se avessimo a disposizione f* potremmo seguire la *Polyak stepsize* (**PSS**), la quale ci permette di determinare lo stepsize ottimo basandosi sulla seguente formula:

$$\alpha_i = \beta_i \frac{f(x_i) - f(x^*)}{\|g_i\|^2} \tag{9}$$

Notiamo la necessità di un ulteriore parametro β, il quale deve rispettare il vincolo $\beta \in [0, 2]$.
Purtroppo nel nostro caso non siamo a conoscenza di f^* e dobbiamo di conseguenza introdurre la tecnica che verrà utilizzata in Algorithm2, ovvero *Target Level Stepsize*. Essa si basa sullo stimare f^* per poter poi attuare la stessa formulazione di Polyak, rimanendo sempre pronto a rivalutare la stima nel momento in cui una nuova e migliore stima viene ottenuta.
Più approfonditamente la tecnica Target Value utilizza dei parametri ρ e δ, tramite i quali, dopo aver fissato una stima iniziale $fref$, ci permette di aggiornare la stima secondo i seguenti concetti:

- δ (parametro che segue il vincolo $\delta > 0$) ci permette di definire una stima di $f^* \longrightarrow fref - \delta$ tale per cui la stima $fref$ verrà aggiornata solo se troveremo un valore f che migliori decisamente la stima (ovvero $f < fref - \delta$).

- ρ (parametro che segue il vincolo $\rho \in (0, 1)$) ci permette di ottenere un vanishing δ, ovvero dopo aver superato una certa threshold andremo a rimpicciolire $\delta \longrightarrow (\delta = \rho \cdot \delta)$. Questo viene effettuato perché più ci avvicineremo alla soluzione più il range che ci distanzia da f^* rimpicciolisce, lasciando meno spazio di miglioramento ad f.

L'altro elemento richiestoci dalla task è l'utilizzo di un *deflected method*. Questa metodologia ci permette di basare il nuovo punto di valutazione di f (ovvero x_i) non solo dando peso al subgradient calcolato ma anche basandosi sulla *precedente direzione intrapresa*, evitando un andamento esageratamente a zig zag. Per pesare le due parti necessitiamo di un parametro γ (vincolato tra 0 e 1), arrivando alla formulazione:

$$d_i = \gamma_i g_i + (1 - \gamma_i) d_{i-1} \tag{10}$$

In particolare all'interno di Algorithm2 andremo ad utilizzare l'approccio *deflection-first/stepsize-restricted*, ovvero ad ogni iterazione definiremo innanzitutto la direzione d da intraprendere seguendo (10) per poi ottenere lo $stepsize$:

$$stepsize_i = \beta_i \frac{f(x_i) - fref + \delta}{\|d_i\|^2} \tag{11}$$

Vincolando $\beta \longrightarrow \beta \leq \gamma$.

3 Structures for implementation

Passando dalla formalizzazione matematica vista nella sezione precedente giungiamo alla sua implementazione. In Algorithm1, partendo dai dati di input ed il rispettivo output, vengono calcolati i valori dei moltiplicatori lagrangiani i quali successivamente permettono di trovare il valore di w e b. L'obiettivo è quindi quello di rappresentare il problema di massimizzazione formalizzato in (6). Per comodità il primo passaggio è trasformare la ricerca di un massimo nella ricerca di un minimo, facilmente attuabile tramite il cambio di segno della nostra funzione. Successivamente vanno definite le matrici e i vincoli che ci permettono di rappresentare il *Quadratic Problem*:

$$\min \frac{1}{2}x'Qx + qx \longrightarrow \forall_i \, lower\_bound \leq x_i \leq upper\_bound \tag{12}$$

Analizziamo quindi la formalizzazione dei singoli componenti di (12) insieme ad altri componenti necessari all'implementazione:

- **kernel matrix**: presi m record di input sarà una matrice simmetrica $K(x, x') \in \mathbb{R}^{m,m}$
- **alpha matrix (x)**: presi m record di input sarà un vettore $x \in \mathbb{R}^{2m}$ tale per cui $x_i \neq 0$ solo se il relativo input genera un output che contribuisce alla $loss$. Prima metà di x rappresenterà l'insieme di α mentre la seconda metà rappresenterà l'insieme di $\alpha*$.
- **Q**: da (6) si nota che è rappresentato dalla *kernel matrix*, fatto non sufficiente per assicurare la proprietà di essere *positive semi-definite*, abbreviato da qui in poi con **PSD** (basta pensare al kernel *sigmoid*). La formalizzazione che viene utilizzata all'interno di Algorithm1 [1] ci assicura invece la proprietà di **PSD**, lo spazio del problema diventa dunque convesso assicurandoci che punti stazionari sono anche di minimo globale, proprietà non ottenuta da altri modelli (per esempio NN).
- **q**: costruita in modo da unire i rimanenti elementi di (6).
- **linear constraint**: il vincolo lineare presente sui moltiplicatori lagrangiani ci permette di definire $lb = 0$ e $ub = C$.

Data questa formalizzazione tramite Algorithm2 verrà risolto il problema di minimizzazione ottenendo i valori dei moltiplicatori lagrangiani. A questo punto sarà quindi possibile calcolare il valore di w e b in modo tale da rappresentare la nostra funzione:

$$f(x) = wx + b$$

In particolare, per determinare il valore di b andremo ad applicare quanto già evidenziato precedentemente in (8), sarà necessario trovare un elemento che si trovi al di fuori dell'ε-tube e rispetto quest'ultimo calcolare il valore.

4 Pseudo Code

Algorithm 1: Get_SVR

Input: X input matrix of size $m \times d$
y output vector of size m
C regularization parameter
ε parameter defining sensitiveness of model
$kernel\_type$ defining which kernel to use
$args$ necessary and different for each kernel
Output: Parameters w e b for resulting function ($f(x) = wx + b$)

begin
  $K \longleftarrow kernel(X, kernel\_type, args)$
  `// to assure` Q `to be Positive Semidefinite`
  $Q \longleftarrow [K \quad -K; \quad -K \quad K]$
  $q \longleftarrow [-y \quad y]$
  $q \longleftarrow q + \varepsilon$
  $upper\_bound \longleftarrow C$
  $alphas \longleftarrow Fit\_SVR(Q, q, upper\_bound)$
  $alpha^+ \longleftarrow alphas_{0:(size(alphas)/2)}$
  $alpha^- \longleftarrow alphas_{(size(alphas)/2):size(alphas)}$
  $w \longleftarrow (alpha^+ - alpha^-)K$
  `// first` $i \mid alpha_i^+ \neq 0$ `or` $alpha_i^- \neq 0 \longrightarrow alpha\_not\_zero = i$
  $b = y_{alpha\_not\_zero} - \sum_i (alpha_i^+ - alpha_i^-) K_{i,alpha\_not\_zero}$
end

Algorithm 2: Fit_SVR

Input: Q matrix of size $2m \times 2m$ for qp
q vector of size $2m$ for qp
ub upper bound for constrained variables
$stopping\_criterion$ ($default = 1e{-}12$)
γ for Deflection ($default = 0.5$)
β for Target Level ($default = 0.5$)
eps for Target Level ($default = 1e{-}6$)
δ_reset for Target Level ($default = 1e{-}4$)
ρ for Target Level ($default = 0.95$)
$maxiter$ for stopping iteration ($default = 1000$)

Output: Optimal x to minimize constrained $\frac{1}{2}x'Qx + qx$

```
begin
    x ⟵ ([0] · size(q))'                                  // init of lagrangian multipliers
    xref ⟵ x
    fref ⟵ inf
    δ ⟵ 0
    dprev ⟵ 0
    iter ⟵ 1
    while true do
        // check if in stopped condition
        if iter > maxiter then
            return xref
        end
        v ⟵ ½x'Qx + qx
        g ⟵ Qx + q
        // check if in optimal condition
        if ‖g‖ < stopping_criterion then
            return x
        end
        // reset δ if v is good or decrease it otherwise
        if v ≤ fref − δ then
            δ ⟵ δ_reset · max v, 1
        else
            δ ⟵ max(δρ, eps · max(|min(v, fref)|, 1))
        end
        // update fref and xref if needed
        if v < fref then
            fref ⟵ v
            xref ⟵ x
        end
        // compute stepsize and new x
        d, dprev ⟵ Get_Direction(γ, g, dprev, ub, x)
        stepsize ⟵ β(v−fref+δ) / ‖d‖²                     // deflection-first → β ≤ γ
        x ⟵ x − stepsize · d
        iter ⟵ iter + 1
    end
end
```

Algorithm 3: Get_Direction

Input: $\gamma \in [0, 1]$ needed for Deflection ($default = 0.5$)
g gradient
$dprev$ previous direction for Deflection
ub upperbound for ensuring constraints
x constrained variable $\longrightarrow x_i \in [0, ub]$

Output: d direction for next step
$dprev$ updated previous direction

```
1  begin
2      // compute direction d using deflection
3      d ⟵ γg + (1 − γ)dprev
4      // ensure constraints ⟶ ∀_i x_i ∈ [0, ub]
5      for i ← 0 to size(x) do
6          if (ub − x_i < 1e−10 and d_i < 0) or (x_i < 1e−10 and d_i > 0) then
7              d_i ⟵ 0
8          end
9      end
10     return d, d
11 end
```

5 Doubts

1. Abbiamo letto diversi paper in letteratura e abbiamo trovato diversi modi in cui viene determinato il valore b. Alcuni dicono che basta usare un punto che sia fuori o al margine dell'ε-tube [2], altri invece lo calcolano su tutti i punti per poi fare una media. C'è un metodo più corretto di un altro?

2. Nel calcolo del *Polyak stepsize* utilizziamo un parametro β che nel caso di *deflection-first* $\longrightarrow \beta \leq \gamma$. Nelle slide e appunti però non abbiamo visto nessuna formula rispetto la quale è definito, esiste o è determinato tramite una *gridsearch*?

3. Abbiamo visto studiando dagli appunti presi a lezione e dalle slide che γ (parametro necessario al *deflected subgradient*) viene definito:

   $$\gamma_i \in \min\{\|\gamma g_i + (1-\gamma)d_{i-1}\|^2 : \gamma \in [0,1]\}$$

   Come ha detto Lei non è obbligatorio avere un valore variabile di γ che si adatta a quello che è il comportamento della nostra ricerca del minimo, anche se avere un valore adattivo ci permetterebbe di ottenere risultati migliori. Dobbiamo quindi analizzare entrambe le possibilità? Nel caso in cui dovessimo ottenere γ tramite minimizzazione dovremo andare a risolvere un problema di minimizzazione ad ogni iterazione del nostro Algorithm2? Non diventa **pesante** dal punto di vista della computazione?

6 References

1. LIBSVM: A Library for Support Vector Machines

2. SVR KKT Dual Derivation