UNIVERSITÀ DEGLI STUDI DI PERUGIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA E ROBOTICA

"Ottimizzazione Fitting Linear SVM distribuita tramite ADMM e CVX con approccio split by data"

Studente

Rengo Mattia

Prof.

Paolo Banelli

Tesina per il corso di

Signal Processing and Optimization for Big Data

I. Sommario

"Questo lavoro verte prima sullo studio teorico e poi sugli aspetti implementativi dell'algoritmo Support Vector Machine in ambito Big Data. Il focus è stato posto sulla risoluzione distribuita di un problema di fitting in cui i parametri della SVM lineare utilizzata sono stati ottimizzati in modo distribuito tramite ADMM coadiuvato da CVX seguendo un approccio di tipo split by data. Seguirà poi un'analisi delle prestazioni ottenute dal codice implementato a partire dallo studio teorico comparandole con quelle dello stato dell'arte delle SVM in MATLAB. A conclusione si eseguirà una digressione sulle ottimizzazioni applicate al codice prodotto oltre a un excursus sulle ulteriori tecniche applicabili per velocizzarne ulteriormente l'esecuzione"

Indice

II. Introduzione

L'obiettivo di questa tesina è la risoluzione distribuita di un problema di fitting in cui i parametri della SVM lineare utilizzata sono stati ottimizzati in modo distribuito tramite ADMM coadiuvato da CVX seguendo un approccio di tipo split by data. Seguirà poi un'analisi delle prestazioni ottenute dal codice implementato a partire dallo studio teorico comparandole con quelle dello stato dell'arte delle SVM in MATLAB. A conclusione si eseguirà una digressione sulle ottimizzazioni applicate al codice prodotto oltre a un excursus sulle ulteriori tecniche applicabili per velocizzarne ulteriormente l'esecuzione

La tesi è articolata in tre capitoli:

Nel primo vengono fornite le informazioni teoriche utili a contestualizzare e conoscere meglio gli aspetti centrali e gli obiettivi di questo lavoro.

Nel secondo capitolo si passerà all'implementazione in MATLAB del codice della SVM distribuita, mettendo in pratica quanto discusso nella teoria e citando le ottimizzazioni del codice utilizzate per migliorare le performance.

La trattazione terminerà con il terzo capitolo nel quale verranno mostrate e discusse le prestazioni raggiunte parlando anche di quali strumenti di MATLAB potremmo sfruttare per migliorarle ulteriormente.

Prima di scendere nei dettagli di questa trattazione e prima di esporne i criteri e le scelte di progettazione, è utile una breve introduzione all'ambito di applicazione e agli strumenti impiegati.

1. Informazioni preliminari

1.1 Le Support Vector Machines

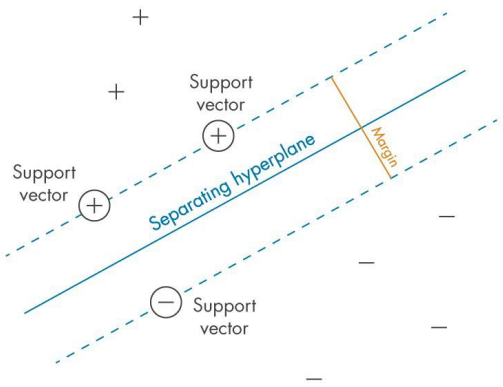
1.1.1 Cos'è una Support Vector Machine

Le Support Vector Machine o SVM sono modelli di classificazione il cui obiettivo è quello di trovare la retta di separazione (in generale è un iperpiano separatore) delle classi che massimizza il margine tra le classi stesse, dove con margine si intende la distanza minima della retta dai punti delle classi, tra tutti i punti quelli essenziali per la SVM sono i Support Vectors.

I Support Vectors sono quei valori di una classe più vicini alla retta di separazione, cioè proprio quelli che essendo più vicini a quelli dell'altra classe sono classificabili correttamente con maggiore difficoltà.

1.1.2 Come funziona una SVM

La SVM cerca gli esempi più difficili, quelli che tendono ad essere più vicini all'altra classe, cioè i vettori di supporto e considera solo quelli per eseguire la classificazione dei nuovi dati. Inoltre, maggiore è il margine, migliore sarà la generalizzazione. Il motivo è che maggiore sarà il margine, maggiore sarà anche la distanza tra le classi e quindi la possibilità di fare confusione (misclassification) sarà minore.



La SVM in questo differisce dagli altri algoritmi di classificazione come, ad esempio, la Linear Regression che invece impara a classificare prendendo come riferimento gli esempi più rappresentativi di una classe.

1.2 ADMM

1.1.3 Cos'è ADMM

L' Alternating Direction Method of Multipliers (ADMM) è un algoritmo che risolve i problemi di ottimizzazione convessi suddividendoli in parti più piccole, ognuna delle quali è quindi più facile da gestire. Il suo punto di forza è che rende facilmente distribuibili su più agenti problemi di ottimizzazione convessa disaccoppiando il problema di ricerca del minimo della funzione originale dal problema di ricerca del minimo della funzione di regolarizzazione, alternando l'ottimizzazione tra la variabile primale x e una nuova variabile fittizia z chiamata slack variable.

La formulazione generale è la seguente:

$$\min_{(\underline{x},\underline{z})} f(\underline{x}) + g(\underline{z})$$

$$\text{s.t. } A\underline{x} + B\underline{z} = \underline{c} \rightarrow \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} \underline{x} \\ \underline{z} \end{bmatrix} = \begin{bmatrix} \underline{c}_1 \\ \underline{c}_2 \end{bmatrix}$$

A questo punto definiamo il lagrangiano del problema dato

da f (x) + g(z) più il vincolo e il termine di regolarizzazione come:

$$\mathcal{L}_\rho(\underline{x}, \underline{z}, \nu) = f(\underline{x}) + g(\underline{z}) + \nu^T(A\underline{x} + B\underline{z} - \underline{c}) + \frac{\rho}{2}\|A\underline{x} + B\underline{z} - \underline{c}\|_2^2$$

Da cui si ricavano i tre passi di aggiornamento iterativo di ADMM qui riportati in versione scalata:

$$\underline{x}^{(k+1)} = \underset{\underline{x}}{\text{argmin}}\{f(\underline{x}) + \frac{\rho}{2}\|A\underline{x} + B\underline{z}^{(k)} - \underline{c} + \underline{u}^{(k)}\|_2^2\}$$

$$\underline{z}^{(k+1)} = \underset{\underline{z}}{\text{argmin}}\{g(\underline{x}) + \frac{\rho}{2}\|A\underline{x}^{(k+1)} + B\underline{z} - \underline{c} + \underline{u}^{(k)}\|_2^2\}$$

$$\underline{u}^{(k+1)} = \underline{u}^{(k)} + (A\underline{x}^{(k+1)} + B\underline{z}^{(k+1)} - \underline{c})$$

1.3 Analisi teorica SVM con ADMM distribuito

1.3.1 Dal problema centralizzato a quello distribuito

L'algoritmo SVM permette di definire un separatore lineare risolvendo il seguente problema di ottimizzazione:

$$min_{w,b} \sum_{i=1}^{M} max\left(0\,,1-y_i\left(w^T\underline{x}_i+b\right)\right)+\lambda||\,w||^2$$

Vediamo ora come si giunge alla scrittura teorica che permette di addestrare le SVM quando i dati di training vengono distribuiti su più agenti ed essi devono raggiungere un consenso per restituire la soluzione ottima, cioè i parametri che rappresentano l'iperpiano separatore ottimo.

Per raggiungere questo obiettivo, il problema del training di un SVM lineare centralizzata può essere rivisto come un insieme di sotto problemi di ottimizzazione convessa, distribuiti uno per nodo con un vincolo sui parametri che permetta alle singole soluzioni di ogni agente di convergere ad un'unica soluzione globale.

La funzione obbiettivo è convessa in quanto è costituita dalla somma di funzioni convesse: possiamo infatti vederla scomposta in due parti, $f(x)\ e\ g(x)$, in cui:

La funzione $f(x)=\ \sum_{i=1}^{M} max\left(0\,,1-y_i\left(w^T\underline{x}_i+b\right)\right)$ è la Hinge Loss

Mentre la $g(x)=\lambda||\,w||^2$ rappresenta il termine di regolarizzazione.

Interpretando il problema di ottimizzazione in forma più generale come una loss function più una regolarizzazione $l(Ax\ -\ b)+r(x)$ otteniamo la seguente scrittura compatta: $min_x\ \underline{1}^T(\max(0\,,1+Ax))+\lambda||x||^2$.

Volendo risolvere questo problema di ottimizzazione convesso attraverso ADMM distribuito, lo riscriviamo come

$$min\sum_{i=1}^{N}1^T(\max\underline{x}\,(0\,,1+A_ix_i))+\lambda||z||^2$$

$$\text{s.t}\qquad x_i-z=0\qquad i=1,\dots,N$$

Calcolando il suo Lagrangiano aumentato otteniamo:

$$\mathcal{L}\left(\,[x_i]_{i=1}^N\,,z\,,\,[y_i]_{i=1}^N\,\right)=\sum_{i=1}^{N}\left[\ \underline{1}^T(\max(0\,,1+A_ix_i))+\lambda\left||z|\right|^2+y_i^T(x_i-z)+\frac{\rho}{2}\left||x_i-z|\right|^2\ \right]$$

Che in fine ci porta alla scrittura dei passi di aggiornamento di ADMM che utilizzeremo operativamente nel codice per ricavare i parametri dell'iperpiano ottimo della SVM:

$$x_i^{k+1} = min_{x_i}\ \underline{1}^T(\max(0\ , 1 + A_i x_i)) + \frac{\rho}{2}\left|\left|x_i - z^k + u_i^k\right|\right|^2,\ \ i = 1, \dots, N$$

$$z^{k+1} = \frac{N\rho}{(2\lambda + N\rho)}(\overline{x}^{k+1} + \overline{u}^k)$$

$$u_i^{k+1} = u_i^k + x_i^{k+1} - z^{k+1},\ \ i = 1, \dots, N$$

2.Implementazione SVM con ADMM distribuito

2.1 Scrittura del codice

Il codice del progetto, ottimizzato ove possibile, è composto da tre script MATLAB che contengono le funzioni utili per:

- Caricamento e generazione dati;
    - Imposta un comportamento di default modificabile a piacere e permette di scegliere se caricare un dataset o generare dati randomici.
    - Il codice è stato ideato per essere flessibile. Ciò è stato fatto introducendo meno ipotesi possibili sulla struttura del dataset usato e calcolando dinamicamente tutte le dimensioni ed i parametri necessari in modo da permettere al codice di adattarsi a più dataset.
    - È previsto un comportamento di fallback in caso non si scelga alcun dataset che richiama la funzione di generazione dati con i parametri di default.
- Calcolo distribuito con ADMM dei parametri $\underline{w}$ e b dell'iperpiano separatore (training SVM);
    - Esegue i passi di ADMM distribuito simulando in maniera sequenziale un'architettura multi-agente parallela.
    - Ogni agente utilizza CVX all'interno del passo di aggiornamento della propria variabile primale x di ADMM.
    - Restituisce i valori ottimizzati $\underline{w}$ e b per l'iperpiano separatore racchiudendoli all'interno della variabile lastx.

    - La funzione termina la sua esecuzione al raggiungimento della convergenza, cioè controllando condizioni di stop sulla tolleranza, o al raggiungimento del massimo numero di iterazioni permesse.
    - Anche qui dove possibile il codice è stato ottimizzato (ad es. preallocazione variabili, notazione matriciale) per snellire l'esecuzione.
- Esecuzione dell'intero progetto con visualizzazione dei risultati ottenuti e confronto con il fitting realizzato da MATLAB mediante la sua apposita funzione;
    - RunMe.m avvia il progetto. Al suo interno troviamo i parametri e i flag modificabili con i quali possiamo regolare il comportamento della SVM durante l'apprendimento. Possiamo scegliere se caricare i dati con "load" o crearne di nuovi con "random", forzare una configurazione sbilanciata dei dati negli agenti con "worstAssign", modificare il lambda della SVM e il Rho degli step di ADMM o impostare un diverso limite per il massimo numero di iterazioni.
    - Dopo il caricamento dei dati viene effettuato il fitting in modo distribuito. I parametri w e b ottimi vengono restituiti al termine della chiamata dello script svm_admm che addestra la SVM (svm1). Vengono poi generati i grafici riguardanti l'addestramento di svm1.

    - A questo punto viene eseguito l'addestramento di una seconda SVM (svm2) stavolta tramite la funzione built-in di MATLAB fitcsvm e vengono confrontati e visualizzati graficamente i piani separatori individuati dalle due SVM rispetto i dati di training.

    - Si procede con la prediction sui dati di test, si misura l'accuracy delle due SVM e si visualizzano graficamente i dati di test classificati e i due decision boundaries delle due SVM.

Il codice completo è disponibile su GitHub al seguente indirizzo:

<https://github.com/MRColorR/SVM_ADMM-Distributed>

2.2 Ottimizzazione dell'esecuzione

L'ottimizzazione già presente nel codice riguarda principalmente:

- La scrittura delle operazioni e delle variabili in una forma congeniale a MATLAB che permette di ottenere performance migliori.
- Il preallocamento delle variabili ove ritenuto necessario, ad esempio nelle variabili di appoggio dei cicli for o parfor (vedi più avanti).
- La riscrittura del problema CVX sottoforma di funzione chiamata, il che permette di eseguire CVX in cicli parfor su una processes parallel pool, parallelizzando così effettivamente il calcolo.
- Cicli parfor per il sottocilo di aggiornamento della variabile duale u.
    - Non chiamando CVX in questo aggiornamento si può utilizzare parfor anche con la più semplice parallel pool thread, che generalmente funziona correttamente su tutte le macchine a differenza della parallel pool processes che è però più potente e completa.

Altre ottimizzazioni che per semplicità di portabilità sono state omesse nella versione finale, poiché specifiche o dell'ambiente SW o dell'HW della macchina su cui viene eseguito il codice ma che sono ovviamente facilmente reinseribili essendo il codice prodotto già predisposto sono:

- Cicli parfor per il sottociclo di aggiornamento delle x dei vari agenti
    - Richiede necessariamente una JVM e una processes parallel pool attiva, oltre alla riscrittura in funzione già discussa sopra, per evitare che CVX smetta di funzionare.
    - A seconda della configurazione della macchina su cui si esegue il codice (ad esempio regole firewall) e della versione Java installata la parallel pool di tipo processes può riuscire ad avviarsi o meno poiché la validazione del può fallire a causa di problemi di connessione tra il processo di MATLAB e la JVM.
    - Per questo motivi i cicli parfor sono stati sostituiti nel codice distribuito con i più lenti e semplici cicli for. È consigliabile reinserirli se si ha la possibilità di sfruttare una pool processes.
- Utilizzo per le variabili strutture dati come gpuarray che permettono di immagazzinare le variabili direttamente nelle GPU Nvidia ottenendo alcuni vantaggi nella velocità di elaborazione delle operazioni sulle stesse.
    - Questa ottimizzazione dipende dalla configurazione HW e SW della macchina e perciò non è presente nella versione finale del codice dove si sono dichiarate le variabili in maniera classica per compatibilità.

3.Analisi dei risultati e conclusioni

3.1 Risultati ottenuti

La soluzione implementata da svm_admm che risolve il problema del fitting SVM in modo distribuito raggiunge buone prestazioni sia sul piccolo dataset allegato sia su dataset di dimensioni maggiori generati in modo randomico.

Questa soluzione, seguendo la teoria già discussa nel primo capitolo, tratta la funzione del problema iniziale come una somma di problemi più piccoli risolvibili in modo indipendente da ogni agente. Questi, grazie all'introduzione di un vincolo di consenso, al passare delle iterazioni convergono ad una soluzione unica condivisa che risulta molto vicina alla soluzione raggiunta dalla svm2 addestrata direttamente dalla funzione built-in di MATLAB come si può vedere dai grafici prodotti.

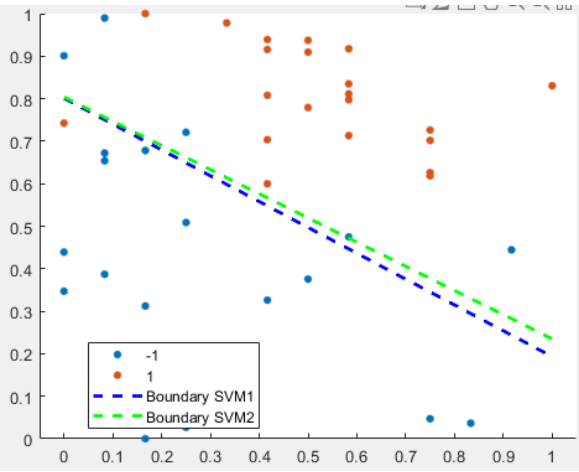


Figura 1 Training svm1 e svm2 su dataset.xml

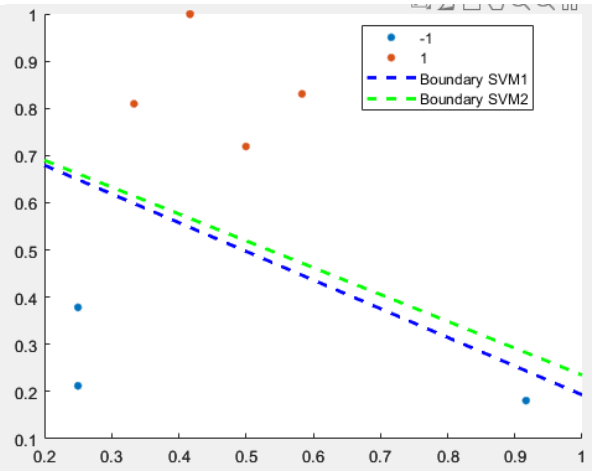


Figura 2 Test svm1 e svm2 su dataset.xml

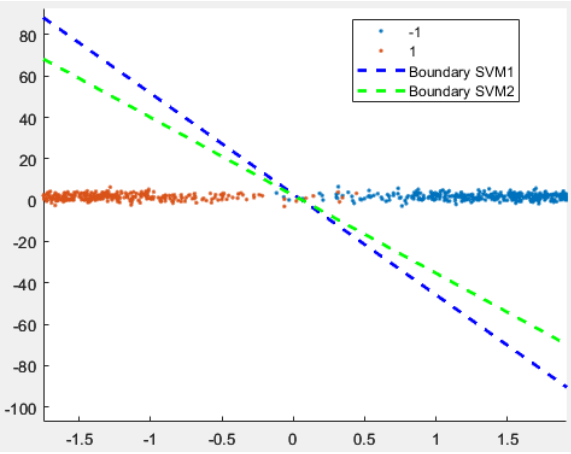


Figura 3 Training svm1 e svm2 su random dataset

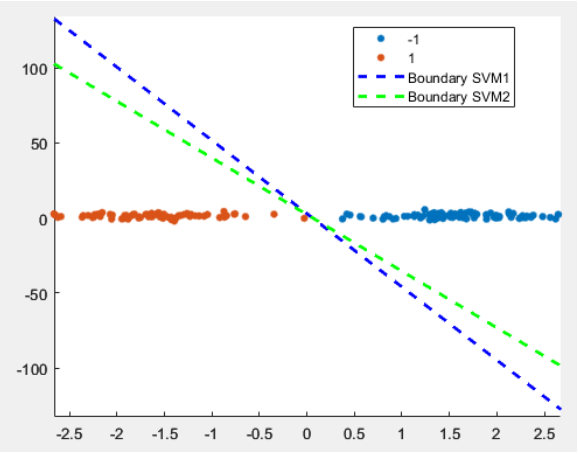


Figura 4 Test svm1 e svm2 su random dataset

In entrambi gli scenari si è giunti a convergenza soddisfacendo le condizioni di stop sulle tolleranze senza dover giungere ad uno stop per raggiungimento del limite sulle iterazioni massime.

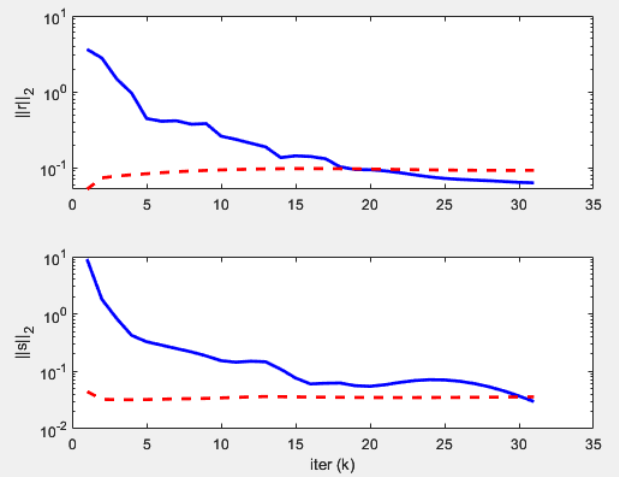


Figura 5 Andamento stop condition su dataset.xls

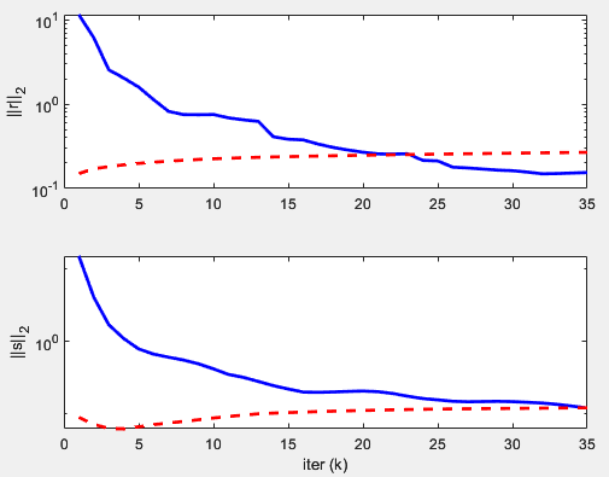


Figura 6 Andamento stop condition dataset random

Anche l'accuratezza raggiunta nei test dalla svm1 addestrata tramite svm_admm essendo in media di 0.98 è risultata assimilabile a quella raggiunta dalla svm2 di riferimento pari in media a 0.99. Questo risultato è in linea con quanto potevamo già aspettarci osservando i decision boundaries selezionati.

Pertanto, gli obiettivi del progetto possono essere considerati sostanzialmente raggiunti pur rimanendo ancora margine per possibili miglioramenti.