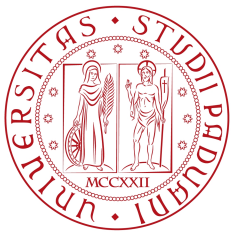

Università degli Studi di Padova

Facoltà di Ingegneria

Corso di laurea in Ingegneria dell'Informazione

Studio di Support Vector Machine per la classificazione e la regressione statistica

Relatore: Ch.mo Prof. Luca SCHENATO

Laureando: Nicola PIOVESAN

Anno Accademico 2012/2013

22 luglio 2013

A Mariarosa, Franco e Serena: la mia famiglia.

Indice

Introduzione

Le macchine a vettori di supporto, meglio note con il termine anglosassone di Support Vector Machine o in breve SVM, sono un insieme di metodi di apprendimento supervisionato che permettono la classificazione e la regressione di pattern. Si deve lo sviluppo di questi metodi a Vladimir Vapnik ed al suo team presso i laboratori Bell AT&T, che vi lavorarono assiduamente durante gli anni '90.

L'apprendimento supervisionato, noto anche con il nome di "apprendimento per esempi", si ha quando l'utente fornisce esempi (un insieme di attributi etichettati con la classe di appartenenza) che descrivono delle classi. Detto questo, ogni algoritmo di apprendimento supervisionato ha bisogno di un set di dati di training (o di addestramento), S, che consista di N dati appartenenti a C classi diverse

$$S = \{\mathbf{x}_n, y_n\} | n = 1, ..., N; \quad \mathbf{x}_n \in \mathbb{R}^D \quad y_n \in C$$

dove $\mathbf{x}_n$ è un vettore (pattern) D-dimensionale le cui componenti sono dette attributi, y_n indica la classe di appartenenza del dato, e C è l'insieme delle classi possibili. La funzione di mappatura $f : y_n = f(\mathbf{x})$, non è nota e un algoritmo di apprendimento si pone il fine di trovarla o perlomeno di approssimarla.

Il set di training rappresenta l'informazione con la frequente assunzione che gli attributi rappresentino le sole proprietà del dato (dell'esempio) e non la relazione tra i vari dati.

Un algoritmo di apprendimento supervisionato si occupa quindi di cercare la funzione f che assegna ad ogni dato la sua classe di appartenenza.

Ogni processo di apprendimento statistico è diviso in due fasi:

1. Fase di apprendimento, dove l'algoritmo analizza i dati di training e riconosce le similitudini nei dati per costruire un modello che approssima f ;

2. Fase di testing, dove il modello generato durante il training viene testato su un diverso set di dati per verificarne le prestazioni.

L'attività di classificazione ha il fine di organizzare le entità di un dato dominio in modo che queste possano essere esposte servendosi di criteri che godano di una certa razionalità.

Dopo aver visto l'uso delle SVM per la classificazione, andremo ad osservare il caso della regressione lineare.

La differenza sostanziale tra il caso della classificazione e quello della regressione sta nel fatto che nel primo ai dati di input viene assegnata un'etichetta che ne identifica la classe, mentre nel secondo ai vari dati di input viene assegnato un valore numerico, ma ciò verrà spiegato più chiaramente nel Capitolo 5.

Possiamo pensare a queste macchine come ad una tecnica alternativa alle classiche tecniche di addestramento delle reti neurali. Potremmo usare, nei nostri problemi, reti neurali ad un solo strato ma in tal caso potremmo lavorare solo nella casistica di dati linearmente separabili. Nel caso optassimo per reti neurali multistrato, potremmo rappresentare funzioni non lineari, utili alla risoluzione di casi a dati non separabili ma va preso in considerazione il fatto che queste reti presentano una serie di problematiche che le rendono difficili da addestrare.

Tali difficoltà sono dovute all'alto numero di dimensioni dello spazio dei pesi e al fatto che le tecniche più diffuse permettono di ottenere i pesi della rete a partire dalla risoluzione di un problema di ottimizzazione non convesso e senza vincoli che, conseguentemente, ha un numero indeterminato di minimi locali.

La tecnica usata per addestrare una SVM risolve entrambe le questioni poste: si ha un algoritmo efficiente ed in grado di rappresentare funzioni non lineari complesse. I parametri richiesti sono inoltre ottenuti dalla soluzione di un problema di programmazione quadratica convesso con vincoli di uguaglianza che prevede un unico minimo globale.

L'apprendimento statistico gioca un ruolo importante in molte aree della scienza, della finanza e dell'industria. Alcuni importanti problemi possono essere risolti facendo uso di sistemi basati sull'apprendimento statistico, come le SVM, tra cui:

- Prevedere se un paziente, ricoverato in seguito ad un attacco cardiaco, avrà un secondo infarto. Tale predizione si basa su osservazioni demografiche, sulla dieta del paziente e su altre osservazioni cliniche;

- Prevedere il prezzo di un azione tra 6 mesi, basandosi sulle performance dell'azienda e su altri dati economici;
- Stimare la quantità di glucosio nel sangue di una persona diabetica a partire dallo spettro di assorbimento dell'infrarosso del sangue del paziente;
- Identificare testi scritti a mano a partire da un'immagine digitalizzata (metodo OCR).

Capitolo 1

La classificazione

La classificazione consiste nel problema di identificare a quale categoria appartenga un nuovo dato osservato, sulla base di un set di dati detti "di training" (o di addestramento) di cui è nota la categoria di appartenenza. Vengono analizzate le proprietà quantificabili delle singole osservazioni che possono essere di diverso tipo:

- di categoria (si pensi al gruppo sanguigno che può essere "A", "B", "AB" o "O");
- di ordine di grandezza: ad esempio "grande", "medio" o "piccolo";
- di valore numerico intero;
- di valore numerico reale.

Un algoritmo che implementa un metodo di classificazione basato sulle osservazioni già fatte in fase di training viene detto classificatore. Tale termine viene usato anche in ambito matematico per indicare una funzione, implementata da un algoritmo di classificazione, che "mappa" i dati di ingresso in una categoria. Nel caso che andremo ad esaminare, ossia quello delle Support Vector Machine, la classificazione è considerata un istanza dell'apprendimento supervisionato.

La classificazione è quindi un esempio del problema più generale del riconoscimento di un pattern, che consiste nell'assegnazione di un qualche tipo di valore di uscita a un dato valore di ingresso. Un altro esempio è la regressione, che assegna un valore di uscita di tipo reale ad ogni ingresso.

Per avere un idea più chiara di quale sia lo scopo della classificazione, consideriamo il caso della classificazione binaria e rappresentiamo i dati su un piano.

Abbiamo L dati detti anche *training point*, dove ogni dato di ingresso $\mathbf{x}_n$ ha D attributi (quindi è un vettore di dimensione D) e appartiene a una delle due classi $y_n = -1$ o $y_n = +1$. I nostri dati di training sono quindi esprimibili nella forma:

$$\{\mathbf{x}_n, y_n\} \quad n = 1, ..., N \quad y_n \in \{-1, 1\}, \ \mathbf{x}_n \in \mathbb{R}^D$$

I dati rappresentati nel piano possono apparirci, a questo punto, distribuiti in due modi diversi. Possiamo trovarci di fronte a dati *linearmente separabili* oppure a dati *non linearmente separabili*. Nel primo caso sarà possibile tracciare nel grafico una linea (se $D = 2$) o un iperpiano (se $D > 2$) che divide esattamente le due classi.

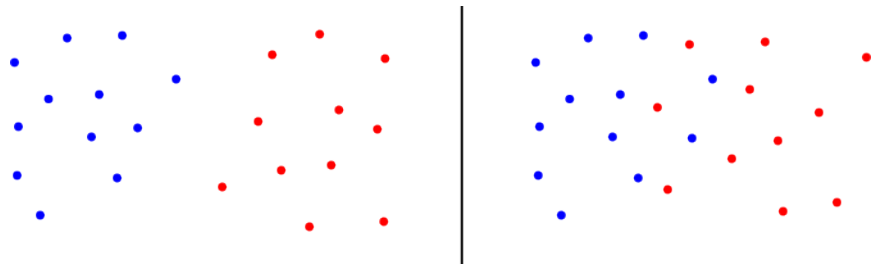

Figura 1.0.1: Possibili casi di separabilità dei dati. dati linearmente separabili (a sinistra), dati non linearmente separabili (a destra)

Nel primo caso riportato in Figura 1.0.1, possiamo intuitivamente tracciare una retta che divide le due classi, indicate con i colori blu e rosso. Nel secondo caso è impossibile tracciare una retta che divida a metà le due classi perché qualche punto si verrebbe a trovare dalla parte errata della retta separatrice.

1.1 Classificazione binaria lineare

La classificazione binaria prevede che i dati appartengano a due classi, che possiamo indicare come "positiva" e "negativa". Supponiamo che questi dati siano separabili, ossia che sia possibile individuare un iperpiano che separa i dati positivi da quelli negativi.

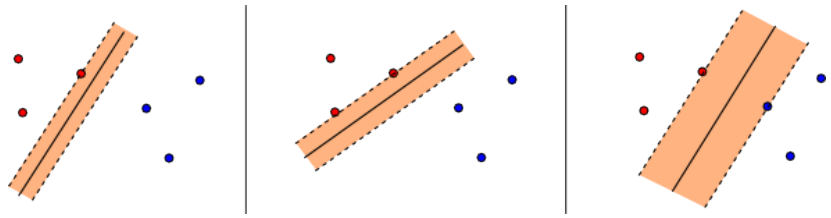

Figura 1.1.1: Iperpiani separatori. Nei primi due casi gli iperpiani non sono ottimali; nell'ultimo caso l'iperpiano è ottimo (ha margine maggiore)

Dalla Figura 1.1.1 possiamo vedere che gli iperpiani che dividono le due classi sono potenzialmente infiniti e che quindi il nostro interesse deve muoversi alla ricerca di quello ottimo che è, intuitivamente, quello con margine maggiore. Possiamo difatti notare che l'iperpiano con margine maggiore consente di ridurre il numero di classificazioni errate in fase di test.

L'iperpiano di separazione è descritto come $\mathbf{w} \cdot \mathbf{x} = 0$ e definiamo ora, con $\mathbf{x}_n$, il dato più vicino a tale piano.

Facciamo a questo punto due considerazioni atte a semplificare i calcoli successivi:

- normalizziamo $\mathbf{w}$ in modo che $|\mathbf{w} \cdot \mathbf{x}_n| = 1$. In particolare tale prodotto senza valore assoluto varrà 1 se $\mathbf{x}_n$ è un dato positivo e -1 se è negativo;

- Portiamo il termine ω_0 fuori da $\mathbf{w}$ in modo che diventi $\mathbf{w} = (\omega_1, \ldots, \omega_d)$ e definiamo $\omega_0 = b$. L'equazione del piano diventa allora: $\mathbf{w} \cdot \mathbf{x} + b = 0$ (il vettore $\mathbf{x}_0$ non contiene, ovviamente, il termine x_o).

Ci interessa a questo punto trovare la distanza tra $\mathbf{x}_n$ e il piano $\mathbf{w} \cdot \mathbf{x} + b = 0$ dove $|\mathbf{w} \cdot \mathbf{x}_n + b| = 1$.

Per trovare questa distanza è di fondamentale importanza considerare che il vettore $\mathbf{w}$ è perpendicolare al piano nello spazio dei dati di ingresso χ. Dimostrare questo fatto è molto semplice.

Prendiamo due punti qualsiasi $\mathbf{x}'$ e $\mathbf{x}''$ sull'iperpiano separatore. Da quanto detto finora sappiamo che questi punti verificano le equazioni: $\mathbf{w} \cdot \mathbf{x}' + b = 0$ e $\mathbf{w} \cdot \mathbf{x}'' + b = 0$. Questo implica che $\mathbf{w} \cdot (\mathbf{x}' - \mathbf{x}'') + b = 0$. Sappiamo che $(\mathbf{x}' - \mathbf{x}'')$ è un vettore che congiunge i due punti e che, essendo nullo il suo prodotto scalare con $\mathbf{w}$, tale vettore deve essere per forza perpendicolare a $\mathbf{w}$. Dal momento che ciò vale qualsiasi siano i punti scelti sul piano, possiamo dire,

come volevamo dimostrare, che $\mathbf{w}$ è ortogonale a qualsiasi vettore sul piano e quindi al piano stesso.

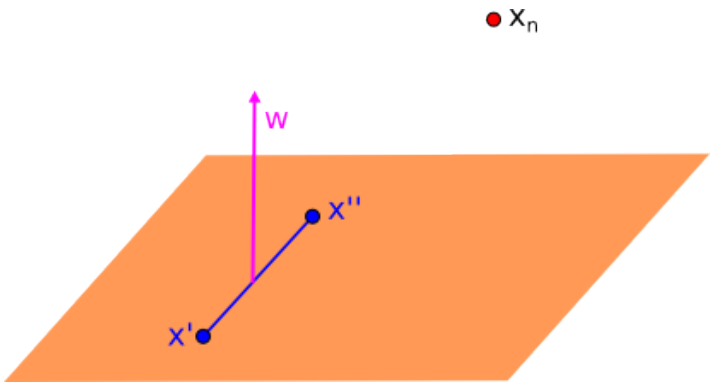

Figura 1.1.2: Il vettore $\mathbf{w}$ è perpendicolare al vettore che congiunge i due punti a caso scelti sul piano.

Per trovare la distanza di un punto $\mathbf{x}_n$ dal piano, scegliamo un qualsiasi punto $\mathbf{x}$ appartenente al piano e cerchiamo la proiezione di $\mathbf{x}_n - \mathbf{x}$ su $\mathbf{w}$. Per trovare tale proiezione moltiplichiamo il versore di $\mathbf{w}$ (che è $\hat{\mathbf{w}} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$) per il vettore $\mathbf{x}_n - \mathbf{x}$.

$$\text{distanza} = |\hat{\mathbf{w}} \cdot (\mathbf{x}_n - \mathbf{x})| = \frac{\mathbf{w}}{\|\mathbf{w}\|}(\mathbf{x}_n - \mathbf{x}) = \frac{1}{\|\mathbf{w}\|}|\mathbf{w} \cdot \mathbf{x}_n - \mathbf{w} \cdot \mathbf{x}| = \frac{1}{\|\mathbf{w}\|}$$

Nell'ultimo passaggio abbiamo considerato che $\mathbf{w} \cdot \mathbf{x}_n = 1$ e $\mathbf{w} \cdot \mathbf{x} = 0$, mentre si è usato il valore assoluto perché si assume che la distanza sia sempre positiva.

Si sono introdotti qui i concetti di distanza, di iperpiano separatore e di margine perché si tratta di aspetti che verranno largamente usati in seguito, durante la trattazione delle varie tipologie di Support Vector Machine e perché queste caratteristiche consentono di capire, almeno dal punto di vista intuitivo, quale sia il problema che ci troviamo ad affrontare.

L'idea riguardo la massimizzazione del margine è stata introdotta da [Boser et al.(1992a)Boser, Guyon, and Vapnik], e nella sua versione semplificata si cercava un iperpiano separatore tra i due set di punti in modo da massimizzare la distanza tra l'iperpiano di separazione e i punti più vicini delle classi da dividere.

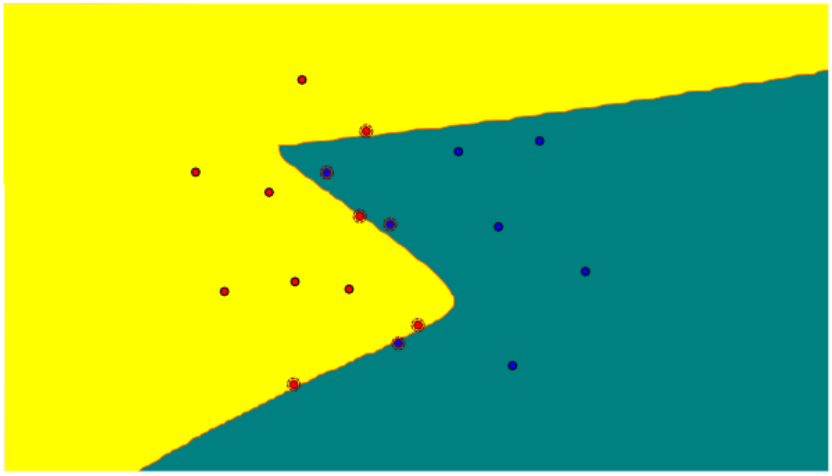

Figura 1.2.1: Esempio di classificazione non lineare facente uso di una funzione polinomiale di 5° grado.

1.2 Classificazione binaria non lineare

Come abbiamo visto, non sempre è possibile dividere correttamente le classi facendo uso di una classificazione di tipo lineare. Possiamo decidere di usare sempre questo tipo di classificazione e accontentarci di ridurre al minimo gli errori che comunque si verrebbero a formare trovandoci nel caso di dati non linearmente separabili, ma è chiaro che questo inciderebbe sull'accuratezza della classificazione in fase di test.

Per ovviare a questo problema possiamo utilizzare classificazioni non lineari. L'idea di [Boser et al.(1992a)Boser, Guyon, and Vapnik] è stata successivamente elaborata da [Cortes and Vapnik(1995)], che hanno introdotto il concetto di Kernel per mappare i dati non separabili in uno spazio di dimensione maggiore e cercare qui un iperpiano di separazione ottimale applicando la classificazione lineare. In questo modo è possibile lavorare in spazi di dimensione molto grande, come quello introdotto dai kernel, senza overfitting. Nel caso della classificazione, le SVM lavorano mappando i dati in spazi di grande dimensione detti *feature space*, dove viene usato un algoritmo lineare per trovare il massimo margine di separazione. L'iperpiano nel feature space può corrispondere a un contorno non lineare nell'insieme dei dati di ingresso. Possiamo vedere un esempio in Figura 1.2.1 dove è stata utilizzata una funzione polinomiale di 5° grado per determinare l'iperpiano separatore del set di dati di

training non linearmente separabili. Tale opzione verrà trattata in modo più esaustivo nel Capitolo 3, riguardante le SVM non lineari.

1.3 Dimensione Vapnik Chervonenkis e misura dell'errore

Supponiamo di avere a disposizione un insieme di N osservazioni e che ogni osservazione consista di un vettore $\mathbf{x}_n \in R^D$ e di un'etichetta $y_n \in \{-1, 1\}$ che ne definisce la classe di appartenenza. Esiste, per assunzione, anche una distribuzione di probabilità non nota $P(x, y)$ da cui sono estratti i dati disponibili.

Lo scopo di una macchina per l'apprendimento automatico è quello di apprendere una funzione che sia in grado di mappare ogni vettore di attributi $\mathbf{x}_n$ nella corrispondente classe di appartenenza y_n. La macchina in questione è definita da un insieme di possibili funzioni $f(\mathbf{x}, \alpha) : \mathbb{R}^D \longmapsto \{-1, 1\}$. Si suppone che tale macchina sia deterministica ossia che per un dato ingresso $\{\mathbf{x}, \alpha\}$ la funzione restituisca sempre lo stesso risultato $f(\mathbf{x}, \alpha)$. Una particolare scelta di α genera una macchina addestrata e addestrare la macchina significa proprio determinare il vettore dei parametri α.

Il valore atteso dell'errore di test per una macchina addestrata è definito come:

$$R(\alpha) = \int \frac{1}{2}|y - f(\mathbf{x}, \alpha)|dP(\mathbf{x}, y)$$

La quantità $R(\alpha)$ è detto *rischio effettivo*.

Un altra importante quantità relativa alla misura dell'errore è il *rischio empirico*, definito come il tasso di errore medio misurato nell'insieme dei dati di training:

$$R_{emp}(\alpha) = \frac{1}{2N}\sum_{n=1}^{N}|y_n - f(\mathbf{x}_n, \alpha)|$$

In questo caso non compare nessuna distribuzione di probabilità e ciò rende possibile calcolare tale quantità con i dati disponibili. $R_{emp}(\alpha)$ è un valore numerico definito per ogni particolare scelta di α e di un insieme di dati di training $\{\mathbf{x}_n, y_n\}_{n=1,..,N}$. La quantità $\frac{1}{2}|y_n - f(\mathbf{x}_n, \alpha)|$ è detta *loss*.

Fissato un numero $\eta : 0 \leq \eta \leq 1$, con probabilità $1-\eta$ vale la diseguaglianza

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\left(\frac{h(\log(2N/h)+1) - \log(\eta/4)}{N}\right)} \tag{1.3.1}$$

dove h è un numero intero non negativo detto *dimensione Vapnik Chervonenkis (VC)* che misura la capacità di classificazione espressa dalla macchina rappresentata dalle funzioni appartenenti all'insieme $\{f(\mathbf{x}, \alpha)\}$. Il secondo termine di (1.3.1) è invece detto *VC confidence* e dipende dalla macchina di apprendimento scelta (cioè dalla classe di funzioni scelta) a differenza del rischio empirico e del rischio effettivo che dipendono invece dalla funzione determinata durante l'addestramento della macchina.

Capitolo 2

Support Vector Machine lineari

2.1 Il caso di dati separabili

Supponiamo di avere un insieme di N dati separabili con cui vogliamo addestrare la SVM e di voler effettuare una classificazione di tipo binario, cioè con sole due classi possibili. Questi dati di training sono etticchettati come:

$$\{\mathbf{x}_n, y_n\}, n = 1, \ldots, N, \quad y_n \in \{-1, 1\}, \quad \mathbf{x}_n \in \mathbb{R}^D$$

dove y_n è un'etichetta che può assumere uno dei due valori previsti per l'identificazione della classe di appartenenza mentre $\mathbf{x}_n$ è un vettore contenente gli attributi che caratterizzano il dato. Se il dato, ad esempio, è un'immagine formata da 1000 pixel, il vettore $\mathbf{x}_n$ conterrà 1000 elementi, ciascuno indicante la rappresentazione numerica del colore del singolo pixel. Se invece ci troviamo in un caso clinico, il vettore $\mathbf{x}_n$ conterrà elementi inerenti ad osservazioni fatte sul paziente.

Nel caso binario, che stiamo ora considerando, abbiamo dati con cui effettueremo il training, che possono essere positivi o negativi (rispettivamente il valore di y_n sarà 1 o -1). Lo scopo della SVM è quello di trovare un iperpiano di separazione che divida, nel miglior modo possibile queste due classi.

I punti $\mathbf{x}$ che si trovano sull'iperpiano soddisfano, come già visto, l'equazione: $\mathbf{w} \cdot \mathbf{x} + b = 0$

Se definiamo con d_+ la distanza minima tra l'iperpiano di separazione e il dato positivo più vicino, e con d_- la distanza minima tra l'iperpiano ed il dato

negativo più vicino, possiamo definire il margine dell'intervallo di separazione come $d_+ + d_-$.

Il problema può essere formulato in questo modo: supponiamo che tutti i dati di training soddisfino le seguenti condizioni

$$\mathbf{x}_n \cdot \mathbf{w} + b \geq +1 \quad per\ y_n = +1 \tag{2.1.1}$$

$$\mathbf{x}_n \cdot \mathbf{w} + b \leq -1 \quad per\ y_n = -1 \tag{2.1.2}$$

Trovandoci nel caso binario e assegnando le etichette come illustrato precedentemente, è possibile combinare queste due condizioni in un'unica diseguaglianza:

$$y_n \left(\mathbf{x}_n \cdot \mathbf{w} + b\right) - 1 \geq 0 \quad \forall n \tag{2.1.3}$$

In questo modo vengono considerati solo i punti corretti dal momento che sia nel caso di dati positivi che nel caso di dati negativi il prodotto risulta positivo.

Consideriamo i punti che verificano la sola eguaglianza (2.1.1). Questi punti sono posizionati sull'iperpiano $H_1 : \mathbf{x}_n \cdot \mathbf{w} + b = 1$ mentre, in modo simile, i punti che verificano l'eguaglianza (2.1.2) sono posti sull'iperpiano $H_2 : \mathbf{x}_n \cdot \mathbf{w} + b = -1$.

Gli altri punti si troveranno nello spazio esterno a quello delimitato dai piani H_1 e H_2 andando a verificare la diseguaglianza stretta (2.1.3). Nessun punto si potrà trovare, invece, nello spazio interno a quello delimitato dai piani marginali.

Considerando le definizioni date nel capitolo precedente, la distanza tra l'iperpiano separatore e l'iperpiano H_1 è la stessa che vi è tra l'iperpiano separatore e l'iperpiano H_2 e vale $d_+ = d_- = 1/\left\|\mathbf{w}\right\|$. Detto questo, la larghezza del margine è banalmente la somma di queste due distanze, ossia $2/\left\|\mathbf{w}\right\|$.

Tenendo presente che gli iperpiani H_1 e H_2 sono paralleli, avendo entrambi come vettore ortogonale lo stesso vettore $\mathbf{w}$, iniziamo a considerare l'obiettivo di individuare i due iperpiani che danno il margine maggiore, minimizzando $\left\|\mathbf{w}\right\|^2$ sotto la condizione (2.1.3).

Come abbiamo visto, i punti di training che verificano l'uguaglianza (2.1.3) sono quelli che si trovano esattamente sopra uno degli iperpiani che determinano il margine. Questi punti sono di fondamentale importanza, in quanto la loro rimozione modifica la soluzione trovata. Tali punti sono detti *support*

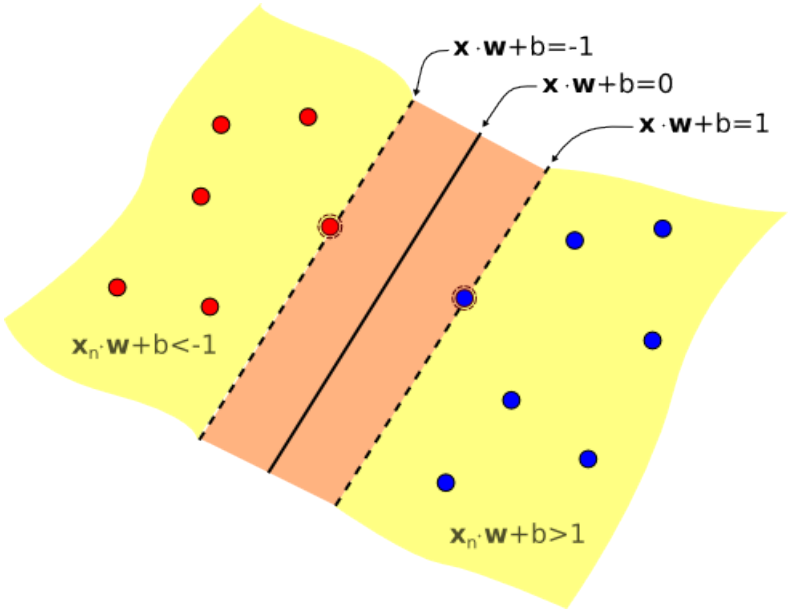

Figura 2.1.1: SVM lineare con dati linearmente separabili. Sono indicate le formule degli iperpiani.

vector e nelle figure presenti in queste pagine sono rappresentati da una doppia cerchiatura.

2.1.1 Formulazione Lagrangiana del problema

La massimizzazione del margine deve avvenire nel rispetto del vincolo imposto dalla diseguaglianza (2.1.3), per cui si tratta di:

$$\text{massimizzare } \frac{1}{\|\mathbf{w}\|} \quad \text{con } \; y_n \left(\mathbf{w}\cdot\mathbf{x}_n + b\right) \geq 1 \; \forall n$$

Possiamo vedere questo problema anche come un problema di minimizzazione. Si rende massimo il margine minimizzando il denominatore. Il problema diventa allora equivalente a

$$\text{minimizzare } \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{con } \; y_n \left(\mathbf{w}\cdot\mathbf{x}_n + b\right) \geq 1 \; ; \qquad n = 1, 2, ..., N \; \mathbf{w} \in \mathbb{R}^D, b \in \mathbb{R}$$

dove il termine $\frac{1}{2}$ è stato aggiunto per semplificare i calcoli, e permetterà poi di risolvere il problema come un problema di programmazione quadratica.

Giunti a questo punto, consideriamo la formulazione Lagrangiana di tale problema per almeno un paio di motivi:

- le condizioni (2.1.3), presenti nel problema di minimizzazione, verranno sostituite da condizioni sui moltiplicatori Lagrangiani stessi, che sono molto più facili da gestire;
- in questa riformulazione, i dati di training appariranno nella forma di prodotti scalari tra vettori.

Questo ultimo aspetto è molto importante in quanto consentirà, successivamente, di estendere la procedura al caso non lineare. Passando alla formulazione di Lagrange otteniamo:

$$\mathcal{L}_P(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} \alpha_n \left(y_n \left(\mathbf{w} \cdot \mathbf{x}_n + b\right) - 1\right)$$

che va minimizzata rispetto a $\mathbf{w}$ e b, ricordando che deve valere sempre la condizione $\alpha_n > 0$ e massimizzata rispetto a $\boldsymbol{\alpha}$.

Come si vede dalla formula che vogliamo minimizzare, abbiamo introdotto i moltiplicatori di Lagrange $\alpha_n, \quad n = 1, ..., N$ (uno per ogni condizione) che devono essere tutti positivi. La regola per la formulazione del problema di Lagrange prevede che per condizioni del tipo $c_i \geq 0$, le equazioni della condizione vengano moltiplicate per dei moltiplicatori di Lagrange positivi e sottratte dalla funzione obiettivo, per formare così la Lagrangiana.

Si tratta di un problema di programmazione quadratica convessa, essendo convessa la funzione che vogliamo minimizzare ed essendo convesso anche l'insieme dei punti che verificano le condizioni (2.1.3). (Ogni condizione lineare definisce un insieme convesso; un insieme di N condizioni lineari definisce l'intersezione di N insieme convessi che è a sua volta convesso).

Ciò significa che possiamo risolvere il problema duale: massimizzare $\mathcal{L}_P$ sotto la condizione che le derivate parziali di $\mathcal{L}_P$ rispetto a $\mathbf{w}$ e b siano nulle e inoltre i vari α_n siano tutti positivi, ottenendo lo stesso risultato. Questa formulazione duale del problema è chiamato "Problema duale di Wolfe" [Fletcher(1987)].

Il concetto di dualità è molto usato nella programmazione matematica. Questa teoria permette di esprimere il problema con una formulazione alternativa che rende possibile una sua risoluzione più efficiente dal punto di vista computazionale. L'idea di base è quella di costruire in corrispondenza di un problema di minimo

$$\min_{x \in S} f(x)$$

detto problema primale, un problema di massimo

$$\max_{u \in U} \psi(u)$$

detto problema duale, in modo che valga almeno la condizione:

$$\inf_{x \in S} f(x) \geq \sup_{u \in U} \psi(u)$$

Nel caso si riesca a determinare un problema duale sarà possibile ottenere una serie di caratteristiche del problema primale tramite lo studio del duale, come:

- stime del valore ottimo
- condizioni di ottimalità
- metodi di soluzione ottenuti tramite lo studio del problema duale

Porre le derivate parziali uguali a zero, permette di ottenere:

$$\nabla_{\mathbf{w}} \mathcal{L}_P = \mathbf{w} - \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n = 0 \quad \rightarrow \ \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n \tag{2.1.4}$$

$$\frac{\partial \mathcal{L}_P}{\partial b} = -\sum_{n=1}^{N} \alpha_n y_n = 0 \quad \rightarrow \ \sum_{n=1}^{N} \alpha_n y_n = 0 \tag{2.1.5}$$

che essendo condizioni di eguaglianza che devono essere verificate nella formulazione duale, possono essere sostituite in $\mathcal{L}_P$ per ottenere:

$$\mathcal{L}_D(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n \cdot \mathbf{x}_m \tag{2.1.6}$$

$\mathcal{L}_D$ va massimizzata rispetto alla sola variabile $\boldsymbol{\alpha} = \alpha_1, ..., \alpha_N$ sotto le condizioni $\alpha_n \geq 0$ e $\sum_{n=1}^{N} \alpha_n y_n = 0$ per $n = 1, ..., N$.

Si nota immediatamente che la forma duale richiede solo il calcolo del prodotto di ogni vettore di ingresso. Ciò tornerà utile nello studio delle SVM non lineari.

Il training della SVM consiste nel massimizzare $\mathcal{L}_D$ rispetto ai moltiplicatori α_n, sotto le condizioni di positività degli α_n e la condizione (2.1.5). La risoluzione del problema di programmazione quadratica fornisce come risultato il vettore dei moltiplicatori di Lagrange $\boldsymbol{\alpha} = \alpha_1, ..., \alpha_N$ che sostituito nella formula $\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n$ permette di ricavare la soluzione.

Andiamo ora a mostrare un fatto particolarmente importante. In precedenza abbiamo considerato la seguente condizione KKT:

$$\alpha_n \left(y_n \left(\mathbf{w} \cdot \mathbf{x}_n + b \right) - 1 \right) = 0$$

Notiamo che i casi affinché si annulli tale prodotto sono due:

- $y_n \left(\mathbf{w} \cdot \mathbf{x}_n + b \right) - 1 = 0$, oppure
- $\alpha_n = 0$

Il primo caso si verifica quando il dato $\mathbf{x}_n$ si trova su H_1 o H_2, ossia su uno dei due iperpiani di margine. Nel caso di punti interni, invece, l'unico modo perché si annulli tale prodotto è che i moltiplicatori di Lagrange rispettivi a questi dati siano nulli.

A questo punto possiamo affermare che se $\alpha_n > 0$ allora $\mathbf{x}_n$ è un support vector.

Da tutto ciò possiamo capire che i support vector sono di primaria importanza per queste macchine in quanto la soluzione del problema di training e la determinazione degli iperpiani dipende esclusivamente da essi. Se eliminassimo tutti gli altri vettori per poi ripetere nuovamente la fase di training otterremmo come risultato lo stesso iperpiano.

2.1.2 Calcolo del vettore w e del termine b

Per il calcolo del vettore $\mathbf{w}$ possiamo usare la seguente formulazione:

$$\mathbf{w} = \sum_{\mathbf{x}_m \text{è} SV} \alpha_m y_m \mathbf{x}_m \tag{2.1.7}$$

dal momento che, nella formulazione precedente, il valore di α era 0 per vettori $\mathbf{x}$ che non fossero support vector. Giunti a questo punto possiamo calcolare il valore del termine b, inserendo il vettore $\mathbf{w}$ appena calcolato nella formula:

$$y_s \left(\mathbf{w} \cdot \mathbf{x}_s + b \right) = 1 \quad \mathbf{x}_s \, SV \tag{2.1.8}$$

Sostituendo in questa formula la (2.1.7) otteniamo

$$y_s \left(\sum_{\mathbf{x}_m \grave{e} \, SV} \alpha_m y_m \mathbf{x}_m \cdot \mathbf{x}_s + b \right) = 1$$

Moltiplicando entrambi i membri per y_s e considerando $y_s^2 = 1$ si ottiene

$$b = y_s - \sum_{\mathbf{x}_m \grave{e} \, SV} \alpha_m y_m \mathbf{x}_m \cdot \mathbf{x}_s$$

Invece di usare un vettore di supporto arbitrario $\mathbf{x}_s$, è consigliato usare una media dei valori di b calcolati a partire da tutti i support vector, ossia

$$b = \frac{1}{N_s} \sum_{\mathbf{x}_s \grave{e} \, SV} \left(y_s - \sum_{\mathbf{x}_m \grave{e} \, SV} \alpha_m y_m \mathbf{x}_m \cdot \mathbf{x}_s \right)$$

Abbiamo così ottenuto i valori di $\mathbf{w}$ e b che definiscono l'iperpiano di separazione ottimale della Support Vector Machine.

2.1.3 Le condizioni di Karush-Kuhn-Tucker

Le condizioni di Karush-Kush-Tucker (indicate spesso con l'acronimo KKT) giocano un ruolo importante sia nella teoria che nella pratica dell'ottimizzazione condizionata. Per il problema primale, visto nel paragrafo precedente, queste condizioni possono essere espresse come:

$$\nabla_{\mathbf{w}} \mathcal{L}_P = \mathbf{w} - \sum_n \alpha_n y_n x_{n\nu} = 0 \tag{2.1.9}$$

$$\frac{\partial}{\partial b} \mathcal{L}_P = - \sum_n \alpha_n y_n = 0 \tag{2.1.10}$$

$$y_n \left(\mathbf{x}_n \cdot \mathbf{w} + b \right) - 1 \geq 0 \quad n = 1, ..., N \tag{2.1.11}$$

$$\alpha_n \geq 0 \quad \forall n \tag{2.1.12}$$

$$\alpha_n \left(y_n \left(\mathbf{w} \cdot \mathbf{x}_n + b \right) - 1 \right) = 0 \quad \forall n \tag{2.1.13}$$

Tali condizioni sono verificate dalla soluzione di qualsiasi problema di ottimizzazione vincolata (convessa o meno), con ogni tipo di condizione. Esse sono necessarie per la soluzione di un problema di programmazione non lineare in cui i vincoli soddisfano una delle condizioni di regolarità dette condizioni di qualificazione dei vincoli. Si tratta sostanzialmente di una generalizzazione del metodo dei moltiplicatori di Lagrange, applicato a casi in cui sono presenti anche vincoli di disuguaglianza.

Questa assunzione vale per tutte le SVM dal momento che le condizioni sono sempre lineari.

Inoltre il problema di ottimizzazione per le SVM è convesso (la funzione obiettivo è convessa) e per problemi convessi le condizioni KKT sono necessarie e sufficienti perché $\mathbf{w}$, b e $\boldsymbol{\alpha}$ siano soluzione.

Una immediata conseguenza di ciò sta nel fatto che, mentre $\mathbf{w}$ è determinato direttamente dalla procedura di training, la soglia b non lo è, sebbene lo sia implicitamente.

Come abbiamo infatti visto già nel paragrafo precedente, b può essere ricavato dalla condizione di "complementarietà" KKT (2.1.11), scegliendo ciascun n per cui $\alpha_n \neq 0$ e calcolando b.

A tal punto è consigliabile prendere il valore medio di tutti i valori di b ottenuti dalle equazioni (una per ogni $\alpha_n \neq 0$). [Burges(1998)]

Quello che si è fatto è considerare un problema di ottimizzazione dove i vincoli sono più gestibili dei vincoli (2.1.1), (2.1.2).

2.1.4 Fase di test

Una volta addestrata la SVM sarà sufficiente determinare a quale lato dell'intervallo di decisione appartiene un dato di test $\mathbf{x}$ e assegnargli l'etichetta della classe corrispondente. Per fare ciò è molto comune usare la funzione $\text{sign}\left(\mathbf{w} \cdot \mathbf{x} + b\right)$ che restituisce il valore -1 o $+1$ a seconda della classe a cui si suppone appartenga il vettore $\mathbf{x}$.

Per verificare l'accuratezza di una SVM si usa testare il modello su un set di

dati di cui si conosce la classe di appartenenza per poi confrontare se la classe ottenuta dal modello coincide con quella effettiva di appartenenza del dato.

2.1.5 Riepilogo

Quanto visto finora consente di definire una SVM in grado di risolvere il problema della classificazione di dati linearmente separabili, nel caso binario. Per concludere questa sezione riepiloghiamo brevemente i passi necessari.

- Trovare i coefficienti α_n che massimizzano

$$\sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n \cdot \mathbf{x}_m$$

  sotto il vincolo

$$\alpha_n \geq 0 \, \forall n \quad \text{e} \sum_{n=1}^{N} \alpha_n y_n = 0$$

  Ciò viene fatto risolvendo il problema di massimizzazione tramite un programma per la risoluzione di problemi di programmazione quadratica (QP solver).

- Calcolare la normale all'iperpiano, $\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n$
- Determinare l'insieme dei support vector tramite la verifica che i rispettivi moltiplicatori di Lagrange siano $\alpha_n > 0$
- Calcolare la soglia, $b = \frac{1}{N_S} \sum_{\mathbf{x}_s \in SV} \left(y_s - \sum_{\mathbf{x}_m \in SV} \alpha_m y_m \mathbf{x}_m \cdot \mathbf{x}_s \right)$
- Ogni nuovo punto $\mathbf{x}'$ viene classificato valutando $y' = \text{sign}\,(\mathbf{w} \cdot \mathbf{x}' + b)$.

2.2 Il caso di dati non separabili

Il problema che si pone nel caso di dati non separabili è quello di estendere tutto ciò che si è finora ottenuto a questa nuova situazione. Se infatti applicassimo l'algoritmo per dati separabili a un caso con dati non-separabili, non troveremmo alcuna soluzione.

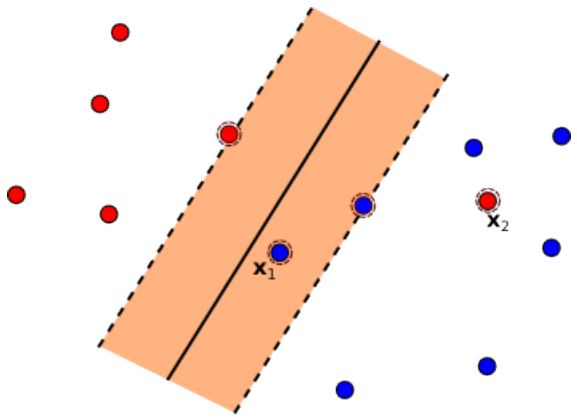

Figura 2.2.1: Il dato denominato $\mathbf{x}_1$ è correttamente classificato ma si trova tuttavia all'interno del margine. Il dato $\mathbf{x}_2$, invece, è classificato in modo errato.

Nell'esempio in Figura 2.2.1 il margine viene violato da un dato che però viene comunque classificato correttamente. La condizione che era sempre vera nel caso esaminato in precedenza, ossia $y_n (\mathbf{w} \cdot \mathbf{x}_n + b) \geq 1$, fallisce. Dobbiamo allora introdurre un metodo per quantificare l'errore che si ha nella violazione del margine.

Una prima idea è quella di rendere più flessibili le condizioni viste nel caso separabile, introducendo delle variabili positive dette "variabili di *slack*" (o allentamento). In questo modo saranno permesse le scorrette classificazioni dei punti. Le condizioni diventano:

$$\mathbf{x}_n \cdot \mathbf{w} + b \geq +1 - \xi_n \quad per \;\; y = +1$$

$$\mathbf{x}_n \cdot \mathbf{w} + b \leq -1 + \xi_n \quad per \;\; y_n = -1$$

con $\xi_n \geq 0 \quad n = 1, ..., N$.
che possono essere riassunte in un unica condizione:

$$y_n (\mathbf{w} \cdot \mathbf{x}_n + b) \geq 1 - \xi_n \quad \xi_n \geq 0$$

Se un generico dato $\mathbf{x}_i$ viene classificato in modo errato, allora ad esso corrisponde uno slack $\xi > 1$ in quanto per essere classificato erroneamente deve trovarsi per forza al di là dell'iperpiano separatore. Possiamo quindi considerare la violazione totale come $\sum_n \xi_n$.

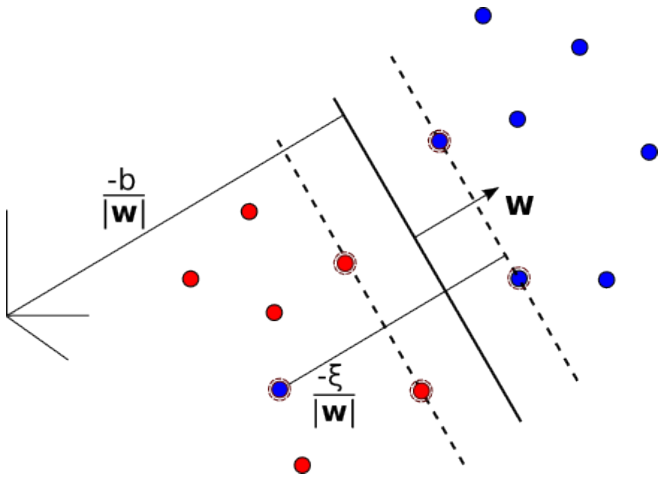

Figura 2.2.2: Distanza dell'iperpiano separatore dall'origine e distanza del punto classificato erroneamente dal rispettivo margine.

Ci troviamo davanti a un nuovo problema di ottimizzazione, per molti versi simile a quello visto nel caso di dati separabili:

$$\text{minimizzare} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\left(\sum_{n=1}^{N}\xi_n\right)^k \tag{2.2.1}$$

$$\text{con } y_n(\mathbf{w}\cdot\mathbf{x}_n + b) \geq 1 - \xi_n \quad \text{per } n = 1, ..., N \quad \text{e } \xi_n \geq 0;$$
$$n = 1, ..., N \; \mathbf{w} \in \mathbb{R}^D, \quad b \in \mathbb{R}$$

Il valore di C è scelto dall'utente e rappresenta il peso che si vuole attribuire agli errori (a un grande valore di tale parametro corrisponde un alta penalità di errore e sarà quindi preferibile avere il minor errore possibile).

Si tratta ancora una volta di un problema di programmazione convessa per qualsiasi intero positivo k. Per $k = 2$ e $k = 1$ è inoltre un problema di programmazione quadratica e la scelta di $k = 1$ ha il vantaggio che né ξ_n, né i rispettivi moltiplicatori di Lagrange, appaiono nel problema duale di Wolfe, che diventa un problema di massimizzazione.

2.2.1 Formulazione Lagrangiana del problema

Il problema di minimizzazione (2.2.1) va trasformato in un problema di Lagrange, ottenendo così il problema primale:

$$\mathcal{L}_P = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n - \sum \alpha_n \left[y_n\left(\mathbf{w}\cdot\mathbf{x}_n + b\right) - 1 + \xi_n\right] - \sum_{n=1}^{N}\beta_n\xi_n$$

Tale funzione va minimizzata rispetto a $\mathbf{w}$, b e ξ e massimizzata rispetto ad ogni $\alpha_n \geq 0$ e $\beta_n \geq 0$, dove i coefficienti α_n e β_n sono moltiplicatori di Lagrange.

Differenziando questa funzione rispetto a $\mathbf{w}$, b e ξ_n e ponendo queste derivate uguali a 0, otteniamo:

$$\nabla_{\mathbf{w}}\mathcal{L}_P = \mathbf{w} - \sum_{n=1}^{N}\alpha_n y_n \mathbf{x}_n = \mathbf{0} \quad \Rightarrow \mathbf{w} = \sum_{n=1}^{N}\alpha_n y_n \mathbf{x}_n \tag{2.2.2}$$

$$\frac{\partial \mathcal{L}_P}{\partial b} = -\sum_{n=1}^{N}\alpha_n y_n = 0 \quad \Rightarrow \sum_{n=1}^{N}\alpha_n y_n = 0 \tag{2.2.3}$$

$$\frac{\partial \mathcal{L}_P}{\partial \xi_n} = C - \alpha_n - \beta_n = 0 \quad \Rightarrow C = \alpha_n + \beta_n \tag{2.2.4}$$

a cui vanno aggiunte anche le seguenti condizioni per formare l'insieme delle condizioni KKT:

$$y_n\left(\mathbf{w}\cdot\mathbf{x}_n + b\right) - 1 + \xi_n \geq 0 \tag{2.2.5}$$

$$\xi_n \geq 0 \tag{2.2.6}$$

$$\alpha_n \geq 0 \tag{2.2.7}$$

$$\beta_n \geq 0 \tag{2.2.8}$$

$$\alpha_n\left[y_n\left(\mathbf{w}\cdot\mathbf{x}_n + b\right) - 1 + \xi_n\right] = 0 \tag{2.2.9}$$

$$\beta_n \xi_n = 0 \tag{2.2.10}$$

Sostituendo le (2.2.2)-(2.2.4) nella formula di $\mathcal{L}_P$ otteniamo il problema duale di Wolfe, esprimibile come:

$$\text{massimizzare} \quad \mathcal{L}_D = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n \cdot \mathbf{x}_m$$

$$\text{rispetto a } \alpha \quad \text{con } 0 \leq \alpha_n \leq C \quad \text{per } n = 1, ..., N \quad \text{e } \sum_{n=1}^{N} \alpha_n y_n = 0$$

Abbiamo posto la condizione $0 \leq \alpha_n \leq C$ perché altrimenti, affinché sia verificata la condizione $C - \alpha_n - \beta_n = 0$ servirebbe un valore di β_n negativo ma ciò non è possibile dovendo essere sempre $\beta_n \geq 0$ come stabilito in precedenza.

La soluzione di questo problema è data da

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n$$

che può essere ristretta alla sola sommatoria dei vettori di supporto, come visto in precedenza.

Osservando i dati del problema notiamo che l'unica differenza con il caso di dati linearmente separabili è che i moltiplicatori di Lagrange α_n sono limitati anche superiormente dalla costante C.

È inoltre possibile usare le condizioni KKT di complementarietà (2.2.9) e (2.2.10) per determinare la soglia b.

L'equazione (2.2.4) combinata con la (2.2.10) mostra che $\xi_n = 0$ se $\alpha_n < C$. Possiamo quindi prendere un qualsiasi punto per cui valga $0 < \alpha_n < C$ per usare la (2.2.9) (con $\xi_n = 0$) e calcolare b. Come valeva per il caso separabile, è conveniente usare la media di tutti i b calcolati tramite questa equazione.

2.2.2 Riepilogo

Anche in questo caso può tornare utile tracciare un sunto dei passi da seguire per descrivere una SVM in grado di agire su dati non separabili.

- Scegliere quanta penalità assegnare alla classificazione non corretta di un dato, scegliendo un opportuno valore per il parametro C.
- Trovare i coefficienti α in modo che

$$\sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n \cdot \mathbf{x}_m$$

  sia massimizzata, sotto il vincolo

$$0 \leq \alpha_n \leq C \; \forall n \text{ e } \sum_{n=1}^{N} \alpha_n y_n = 0$$

  Questo viene fatto tramite la risoluzione di un problema di programmazione quadratica.

- Calcolare la normale all'iperpiano separatore, $\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n$
- Determinare l'insieme dei support vector accertandosi che i rispettivi moltiplicatori di Lagrange verifichino la condizione $0 < \alpha_n \leq C$
- Calcolare la soglia, $b = \frac{1}{N_s} \sum_{\mathbf{x}_s \in SV} \left(y_s - \sum_{\mathbf{x}_m \in SV} \alpha_m y_m \mathbf{x}_m \cdot \mathbf{x}_s \right)$
- Classificare i nuovi dati $\mathbf{x}'$ tramite la funzione $y' = \text{sign}\,(\mathbf{w} \cdot \mathbf{x}' + b)$.

2.2.3 Tipi di support vector

Possiamo osservare due tipi di support vector:

- support vector marginali
- support vector non marginali

Il valore dei moltiplicatori di Lagrange rispettivi ai support vector marginali è ristretto a $0 \leq \alpha_n \leq C$, il valore della rispettiva variabile di slack è sempre $\xi_n = 0$ ed è verificata l'equazione $y_n \left(\mathbf{w} \cdot \mathbf{x}_n + b \right) = 1$

Nel caso dei support vector non marginali, i rispettivi moltiplicatori di Lagrange hanno valore $\alpha_n = C$, mentre le rispettive variabili di slack sono positive ($\xi_n > 0$) e tali SV verificano la diseguaglianza $y_n \left(\mathbf{w} \cdot \mathbf{x}_n + b \right) < 1$

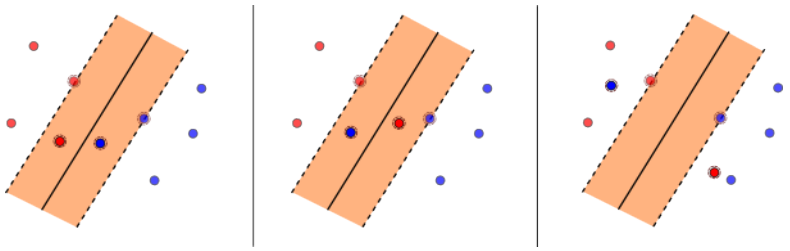

Figura 2.2.3: Diversi configurazioni di support vector non marginali

In Figura 2.2.3 vediamo tre possibili configurazioni. Nel primo caso i support vector si trovano dalla parte corretta rispetto all'iperpiano ma all'interno del margine di separazione. Nel secondo caso si trovano dalla parte errata rispetto all'iperpiano e all'interno del margine di separazione. Nel terzo caso i support vector si trovano dalla parte errata e all'esterno del margine.

2.3 Soluzioni globali e unicità

Possiamo chiederci quando la soluzione del problema di training è globale e quando essa sia unica. Per "globale" si intende che non esistono altri punti nella regione in cui è definita la soluzione in cui la funzione assuma un valore più piccolo di quello trovato.

Riguardo l'unicità possiamo trovarci davanti a due casi in cui essa non è verificata:

- soluzioni per cui $\{\mathbf{w}, b\}$ sono unici ma per cui l'espansione (2.1.7) non lo è;
- soluzioni dove $\{\mathbf{w}, b\}$ sono diverse.

Entrambi questi casi sono importanti: anche se $\{\mathbf{w}, b\}$ è unico, se i moltiplicatori α_n non lo sono, potrà esistere un'espansione equivalente di $\mathbf{w}$ che richiede un numero minore di support vector e che quindi, nella pratica, richiede un numero minore di istruzioni nella fase di test.

È dimostrabile che ogni soluzione locale è anche globale. Questa è una proprietà di ogni problema di programmazione convessa [Fletcher(1987)]. Inoltre la soluzione è unica se la funzione obiettivo (2.1.6) è strettamente convessa.

Riguardo il secondo caso teorizzato, ossia quando le soluzioni sono diverse, può tornare utile il seguente teorema che mostra come se vi sono soluzioni non

uniche, allora la soluzione in un punto ottimale è continuamente deformabile nella soluzione dell'altro punto ottimale, in modo che tutti i punti intermedi siano a loro volta soluzioni.

Teorema 1. *Sia* $\mathbf{X}$ *la variabile corrispondente alla coppia di variabili* $\{\mathbf{w}, b\}$. *La funzione obiettivo sia convessa. Siano* $\mathbf{X}_0$ *e* $\mathbf{X}_1$ *i due punti in cui la funzione obiettivo assume valore minimo. Allora esiste un cammino* $\mathbf{X} = \mathbf{X}(\tau) = (1-\tau)\mathbf{X}_0 + \tau\mathbf{X}_1, \quad \tau \in [0,1]$ *tale che* $\mathbf{X}(\tau)$ *è soluzione* $\forall\tau$

2.4 Osservazioni finali

A conclusione di questo capitolo rispondiamo a una domanda abbastanza intuitiva che ci si potrebbe rivolgere dopo aver letto la teoria riguardante questi due tipi di SVM.

Cosa accade se usiamo la SVM lineare per dati separabili, vista nel paragrafo 2.1, con un insieme di dati di training non separabili?

In questo caso il passaggio tra il problema primale (ossia minimizzare $\|\mathbf{w}\|^2$) e il problema duale (minimizzare il problema di Lagrange rispetto ad α) fallisce. Questo passaggio è infatti matematicamente valido solo se esiste un iperpiano che possa dividere i due insiemi.

Capitolo 3

Support Vector Machine non lineari

Abbiamo considerato finora solo il caso di funzioni di decisione lineari, ma è utile generalizzare quanto ottenuto al fine di risolvere il problema di ottimizzazione anche nel caso in cui la funzione di decisione non sia funzione lineare dei dati.

Il problema di base è che l'insieme dei dati di training non è separabile e che vogliamo trovare un metodo più performante rispetto a quello lineare con slack per suddividere le due classi. Si nota dal problema di training analizzato nel Capitolo 2 che i dati appaiono in esso esclusivamente nella forma di prodotti interni. L'idea è quindi quella di lavorare su uno spazio diverso dallo spazio di ingresso X in cui i dati siano linearmente separabili. Chiameremo questo spazio Z e la sua dimensione può anche essere infinita.

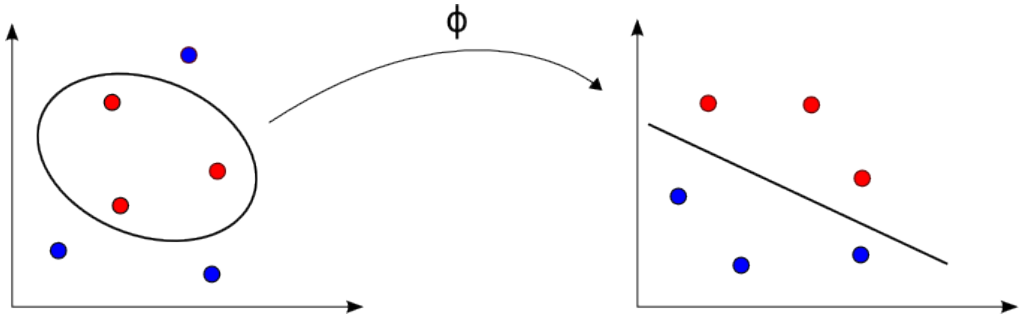

Figura 3.0.1: I dati non linearmente separabili vengono mappati tramite Φ in uno spazio dove sono linearmente separabili.

Andiamo a modificare il problema di Lagrange ottenuto per dati separabili e adattiamolo al nuovo caso considerato, ossia all'operare nello spazio Z:

$$\mathcal{L}_D = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{z}_n \cdot \mathbf{z}_m$$

$$\text{con } 0 \leq \alpha_n \leq C \quad n = 1, ..., N \quad \text{e} \sum_{n=1}^{N} \alpha_n y_n = 0$$

A questo punto possiamo definire la funzione che restituirà la classe di appartenenza:

$$g(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{z} + b)$$

dove i valori di $\mathbf{w}$ e b possono essere facilmente calcolati:

$$\mathbf{w} = \sum_{\mathbf{z}_n \grave{\text{e}}\, SV} \alpha_n y_n \mathbf{z}_n \qquad b \text{ tale che } y_m (\mathbf{w} \cdot \mathbf{z}_m + b) = 1$$

Come si può notare, nel calcolo di $g(\mathbf{x})$ abbiamo bisogno del prodotto scalare $\mathbf{z}_n \cdot \mathbf{z}$ mentre nel calcolo di b abbiamo bisogno del prodotto scalare $\mathbf{z}_n \cdot \mathbf{z}_m$. Non è interessante a questo fine sapere "cosa sia" lo spazio Z ma basta sapere quale sia il risultato del prodotto scalare tra due vettori in esso.

Il trucco consiste quindi nel mappare i dati dallo spazio X a un altro spazio euclideo Z, usando una funzione che chiamiamo Φ:

$$\Phi : \mathbb{R}^D \mapsto Z$$

Ovviamente, in questa situazione, l'algoritmo di training dipende solo dai prodotti interni in Z, che sono: $\mathbf{z} \cdot \mathbf{z}' = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$.

3.1 Il Kernel

Da quanto visto nell'introduzione di questo capitolo, dati i punti $\mathbf{x}, \mathbf{x}' \in X$, vogliamo conoscere il valore di $\mathbf{z} \cdot \mathbf{z}'$.

Definizione. Sia $\mathbf{z} \cdot \mathbf{z}' = K(\mathbf{x}, \mathbf{x}')$ dove K, detto Kernel, è il prodotto interno di $\boldsymbol{\Phi}(\mathbf{x})$ e $\Phi(\mathbf{x}')$ in Z,.

A questo punto può essere utile considerare un esempio

Esempio 2. Il vettore $\mathbf{x} = (x_1, x_2)$ ha dimensione due e rappresenta un generico dato che vogliamo "mappare" nello spazio Z tramite la funzione Φ definita come $\mathbf{z} = \Phi(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$. Supponiamo allora di avere due dati $\mathbf{x}, \mathbf{x}'$ e che si voglia conoscere il loro prodotto interno nello spazio Z. Per definizione di Kernel, tale prodotto vale:

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{z} \cdot \mathbf{z}' = 1 + x_1 x_1' + x_2 x_2' + x_1^2 x_1'^2 + x_2^2 x_2'^2 + x_1 x_1' x_2 x_2'$$

Il trucco che interviene a semplificare di molto le cose consiste nel fatto che è possibile calcolare $K(\mathbf{x}, \mathbf{x}')$ senza dover trasformare $\mathbf{x}$ e $\mathbf{x}'$ in vettori di Z e quindi senza dover conoscere la funzione Φ .

Per chiarire questo fatto consideriamo l'esempio appena visto:

Esempio 3. Invece di calcolare il kernel come il prodotto di due vettori in Z, lo definiamo come una funzione che opera direttamente su vettori in X, ossia

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x} \cdot \mathbf{x}')^2 &= (1 + x_1 x_1' + x_2 x_2')^2 \\ &= 1 + x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2 x_1 x_1' + 2 x_2 x_2' + 2 x_1 x_1' x_2 x_2' \end{aligned}$$

che somiglia molto al prodotto calcolato nell'Esempio precedente a parte la presenza di alcuni coefficienti uguali a 2. La cosa però non ci sconvolge perché si tratta ad ogni modo di un prodotto interno!

La base dello spazio Z in questo caso è $\left(1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2\right)$.

In sostanza il punto di forza di questo approccio è che ci basta sapere che il risultato ottenuto dalla funzione kernel è un prodotto interno in uno spazio Z e non ci interessa, invece, conoscere come è fatto questo spazio, o cosa accade in esso.

3.2 La condizione di Mercer

La condizione di Mercer consente di stabilire se un dato kernel sia ammissibile, ossia se corrisponda effettivamente al prodotto interno di due vettori in uno spazio.

Definizione 4. Esiste una funzione Φ che mappa i dati in uno spazio Z e una espansione

$$K(\mathbf{x},\mathbf{y}) = \sum_i \Phi(\mathbf{x})_i \, \Phi(\mathbf{y})_i$$

se e solo se, per ogni $g(\mathbf{x})$ tale che $\int g(\mathbf{x})^2 \, d\mathbf{x}$ è finito, vale

$$\int K(\mathbf{x},\mathbf{y}) \, g(\mathbf{x}) \, g(\mathbf{y}) \, d\mathbf{x} d\mathbf{y} \geq 0 \tag{3.2.1}$$

L'equazione (3.2.1) deve valere per qualsiasi g con norma finita. In alcuni casi specifici potrebbe non essere semplice verificare che la condizione di Mercer sia rispettata. Ad ogni modo è possibile dimostrare che questa condizione è soddisfatta per integrali positivi di potenze del prodotto interno: $K(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})^p$. [Burges(1998)]

Una semplice conseguenza di questo fatto è che ogni kernel che possa essere espresso come $K(\mathbf{x},\mathbf{y}) = \sum_{p=0}^{\infty} c_p (\mathbf{x}\cdot\mathbf{y})^p$, dove i c_p sono coefficienti reali positivi e la serie è uniformemente convergente, soddisfa la condizione di Mercer.

3.3 Esempi di possibili Kernel

In [Poggio(1975)] viene mostrato come il kernel polinomiale omogeneo K con $p \in \mathbb{N}$ e

$$K(\mathbf{x},\mathbf{x}') = (\mathbf{x}\cdot\mathbf{x}')^p$$

sia un kernel ammissibile. Da questa osservazione, si può concludere immediatamente [Boser et al.(1992b)Boser, Guyon, and Vapnik] che kernel del tipo

$$K(\mathbf{x},\mathbf{x}') = (\mathbf{x}\cdot\mathbf{x}' + c)^p$$

ossia kernel polinomiali non omogenei con $p \in \mathbb{N}, c > 0$ sono anch'essi ammissibili. Questo può essere mostrato riscrivendo K come la somma di kernel omogenei e applicando un corollario del Teorema di Mercer che afferma che la combinazione lineare di funzioni kernel ammissibili è a sua volta una funzione kernel ammissibile [Smola and Schölkopf(2004)].

Un altro kernel che può essere utilizzato nel caso non lineare, è il kernel tangente iperbolica

$$K(\mathbf{x}, \mathbf{x}') = \tanh\left(\vartheta + \phi\left(\mathbf{x} \cdot \mathbf{x}'\right)\right)$$

che però non soddisfa la condizione di Mercer per $\vartheta < 0$ o $\phi < 0$.

Sono molto diffusi anche kernel invarianti rispetto la traslazione ($K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}, -\mathbf{x}')$).

In [Aizerman et al.(1964)Aizerman, Braverman, and Rozoner] viene mostrato come

$$K(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}}$$

sia un kernel ammissibile.

Come è facile intuire, è possibile implementare diversi tipi di kernel nelle SVM non lineari, in modo da ottenere il miglior risultato possibile a partire dall'insieme di dati di training.

3.4 Ipotesi e problema di ottimizzazione

Partiamo dal caso a dati separabili visto nel capitolo precedente.

Quello che vogliamo fare è esprimere la funzione che assegna la classe a un dato, $g\left(\mathbf{x}\right) = \text{sign}\left(\mathbf{w} \cdot \mathbf{x} + b\right)$ in funzione del Kernel.

Tenendo presente che:

$$\mathbf{w} = \sum_{\mathbf{z}_n \text{è } SV} \alpha_n y_n \mathbf{z}_n$$

definiamo la nuova funzione come:

$$g\left(\mathbf{x}\right) = \text{sign}\left(\sum_{\mathbf{x}_n \in SV} \alpha_n y_n K\left(\mathbf{x}_n, \mathbf{x}\right) + b\right)$$

dove

$$b = \frac{1}{N_S} \sum_{\mathbf{x}_s \in SV} \left(y_s - \sum_{\mathbf{x}_m \in SV} \alpha_m y_m K\left(\mathbf{x}_m, \mathbf{x}_s\right)\right)$$

Finora abbiamo detto che non ci interessa cosa sia e cosa accada nello

spazio Z ma è ovvio che deve esistere il prodotto interno di due vettori in esso. Possiamo affrontare il problema dell'esistenza usando 2 diversi approcci:

1. per costruzione (ad esempio nel caso del kernel polinomiale)

2. usando proprietà matematiche (Condizioni di Mercer)

3.5 Riepilogo

Per usare una SVM al fine di risolvere un problema di classificazione su dati non linearmente separabili, dobbiamo prima scegliere un kernel e i relativi parametri che ci aspettiamo consentano di mappare i dati non linearmente separabili in uno spazio dove invece lo siano.

Come nei casi precedenti vediamo quali sono, in modo sommario, i passi da seguire per definire la SVM e per eseguirne il test.

- Scegliere la penalità da attribuire agli errori di classificazione scegliendo un opportuno valore per il parametro C.

- Trovare i moltiplicatori di Lagrange in modo che

$$\mathcal{L}_D = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{z}_n \cdot \mathbf{z}_m$$

  sia massimizzata, sotto il vincolo

$$0 \leq \alpha_n \leq C \; \forall n \quad e \; \sum_{n=1}^{N} \alpha_n y_n = 0$$

  Ciò si ottiene risolvendo un problema di programmazione quadratica.

- Determinare l'insieme dei vettori di supporto verificando che i loro moltiplicatori di Lagrange siano $0 \leq \alpha_n \leq C$

- Calcolare la soglia, $b = \frac{1}{N_S} \sum_{\mathbf{x}_s \in SV} \left(y_s - \sum_{\mathbf{x}_m \in SV} \alpha_m y_m K\left(\mathbf{x}_m, \mathbf{x}_s\right) \right)$

- Classificare i nuovi dati $\mathbf{x}'$ valutando $y' = \text{sign}\left(\sum_{\mathbf{x}_n \in SV} \alpha_n y_n K\left(\mathbf{x}_n, \mathbf{x}\right) + b \right)$

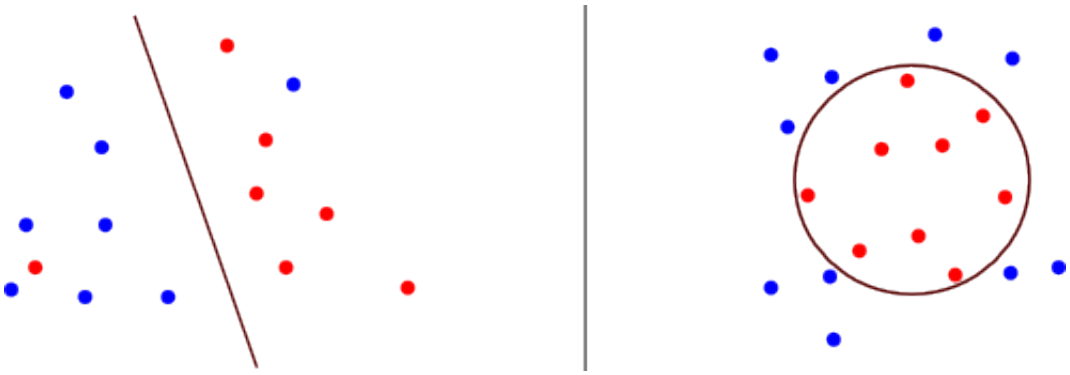

Figura 3.6.1: Dati leggermente separati (a sinistra) e dati molto separati (a destra)

3.6 Conclusioni

Abbiamo visto in questo capitolo come implementare SVM non lineari che consentano di rispondere al problema della classificazione nel caso di dati non separabili. Nel capitolo 2 ci siamo trovati di fronte allo stesso problema che abbiamo però risolto usando SVM lineari con slack.

È utile distinguere due diversi tipi di non-separabilità, e a seconda del tipo scegliere quale delle due SVM usare.

Nel caso di dati leggermente separati è conveniente usare una SVM lineare con slack mentre nel caso di dati molto separati conviene usare una SVM non lineare.

Capitolo 4

Applicazioni di SVM in MATLAB

In questo capitolo cercheremo di applicare quanto mostrato riguardo i vari tipi di SVM per la classificazione di dati. Affronteremo il problema in due modi. Inizialmente vedremo come sia possibile utilizzare la funzione *quadprog* messa a disposizione da MATLAB per la risoluzione di problemi di programmazione quadratica. Useremo questo strumento per realizzare una SVM lineare.

Successivamente analizzeremo due funzioni di MATLAB (*svmtrain* e *svmclassify*) che consentono di implementare SVM di vario tipo più agevolmente.

4.1 Risoluzione di un problema di programmazione quadratica

Lo strumento a più basso livello che MATLAB mette a disposizione per lo sviluppo di Support Vector Machine è la funzione *quadprog*. Essa permette la risoluzione di problemi di programmazione quadratica ed è definita come:

```
x=quadprog(H,f,A,b)
```

Sebbene essa possa accettare un numero maggiore di parametri, è sufficiente invocarla in questo modo per risolvere un problema di programmazione quadratica tipico per le SVM.

In questo caso la funzione minimizza

$$\frac{1}{2}x^T H \cdot x + f^T x \tag{4.1.1}$$

$$\textit{con vincolo} \quad Ax \leq b$$

dove A e b sono rispettivamente matrice e vettore di valori *double*.

Il risultato dato in output dalla funzione quadprog è un vettore x che minimizza (4.1.1). Il vettore x può essere un minimo locale per problemi non convessi mentre è un minimo globale per problemi convessi.

Notiamo subito che è necessario apportare delle opportune correzioni per poter applicare questa funzione ai casi illustrati nei capitoli precedenti.

Nel nostro caso il termine di primo grado $f^T x$ non è presente quindi dovremo porre nullo il vettore f. Inoltre la diseguaglianza del vincolo deve essere di verso opposto in quanto la condizione che vogliamo soddisfare è

$$y_n \left(\mathbf{w} \cdot \mathbf{x}_n + b \right) \geq 1 \tag{4.1.2}$$

Presupposto che nel codice MATLAB indicheremo i dati all'interno di una matrice in cui ogni riga corrisponderà ad un dato, le prime due colonne conterranno gli attributi del dato e l'ultima colonna la classe di appartenenza (1 o -1), procediamo col definire:

$$f = (0, 0, 0)$$

$$H = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$b = (-1, -1, ..., -1)^T$$

dove b ha dimensione pari al numero di dati di training.

La matrice A deve essere costruita in modo che il primo membro della diseguaglianza sia l'opposto di quanto si ha a primo membro nel vincolo (4.1.2).

La soluzione del problema di programmazione quadratica ottenuta dalla funzione quadprog sarà nella forma (w_1, w_2, b).

4.1.1 Realizzazione SVM lineare in MATLAB

Innanzitutto è fondamentale definire due matrici contenenti i dati con relativi attributi e classe di appartenenza così come descritto in precedenza

```
dati1=[1  3  1;2  -4  1;3  6  1];
dati2=[7  7  -1;9  10  -1;8  -3  -1];
```

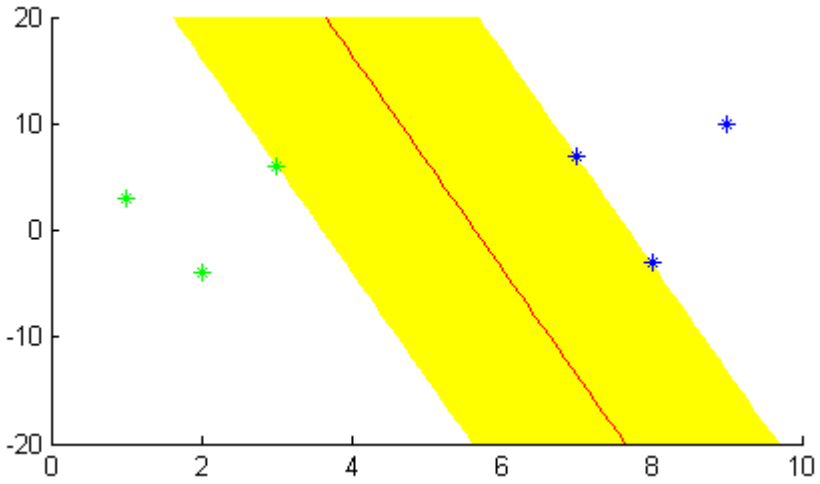

Figura 4.1.1: Rappresentazione dei dati di training e dell'iperpiano separatore calcolato con *quadprog*

A questo punto è necessario definire le matrici e i vettori così come si è visto

```
f=[0 0 0];
H=[1 0 0;0 1 0;0 0 0];

dati=[dati1;dati2];
tot=size(dati,1);
b=-ones(tot,1);

for i=1:tot
  A(i,1)=-dati(i,1)*dati(i,3);
  A(i,2)=-dati(i,2)*dati(i,3);
  A(i,3)=-dati(i,3);
end
```

Una volta definiti tutti i parametri richiesti dalla funzione quadprog, non resta che invocarla

```
w=quadprog(H,f,A,b);
```

In Figura 4.1.1 è illustrato il risultato ottenuto con i dati utilizzati in questo esempio. Il codice completo comprendente quello utilizzato per disegnare il grafico è riportato in Appendice.

4.2 Support Vector Machine in MATLAB

Per semplificare lo sviluppo di SVM e integrare la possibilità di utilizzare svariati tipi di kernel, MATLAB mette a disposizione gli oggetti *svmtrain* e *svmclassify* che consentono rispettivamente di addestrare una SVM e di utilizzarla per la classificazione.

Prima di utilizzare una SVM è necessario definire una struttura

```
SVMStruct = svmtrain(Training,Group);
```

La funzione svmtrain ritorna una struttura contenente informazioni riguardo la SVM addestrata.

Le variabili di input sono:

- Training: matrice dei dati di training, dove ogni riga corrisponde a un'osservazione (a un singolo elemento) mentre ogni colonna corrisponde a una variabile che caratterizza il dato.
- Group: vettore delle classi. L'elemento i-esimo di questo vettore indica la classe a cui appartiene il dato riportato nella riga i-esima della matrice Training.

Tra i vari parametri che è possibile passare alla funzione, ne mostriamo alcuni degni di nota:

- 'kernel_function': permette di impostare la funzione kernel che la funzione *svmtrain* dovrà usare per effettuare il training dei dati. La funzione kernel di default è quella corrispondente al kernel lineare, ossia il prodotto interno. È tuttavia possibile impostare uno dei seguenti kernel:
  - 'quadratic' - kernel quadratico;
  - 'polynomial' - kernel polinomiale. Di default viene usato un kernel polinomiale di ordine 3 ma è possibile impostare l'ordine usando il parametro *polyorder;*
  - 'rbf' - Gaussian Radial Basis Function, con un fattore di scala, sigma, impostato di default a 1. È possibile modificare questo valore tramite il parametro *rbf_sigma.*
- method: consente di impostare il metodo utilizzato per cercare l'iperpiano separatore. Le opzioni disponibili sono:

  - 'QP' - Programmazione quadratica;
  - 'SMO' - Sequential Minimal Optimization. Impostato di default;
  - 'LS' - Minimi quadrati.

- 'showplot': se impostato a *true* permette di visualizzare i dati e l'iperpiano separatore. Tale opzione è valida solo se ogni dato ha due colonne (ossia è caratterizzato da due attributi)

La funzione *svmtrain* restituiste un oggetto che in questo caso è stato chiamato SVMStruct e che contiene tutti i dati che caratterizzano la Support Vector Machine.

Una volta creata e addestrata la SVM, essa può essere utilizzata per classificare un insieme di elementi. Per fare ciò è necessario utilizzare la funzione svmclassify in questo modo:

```
Group = svmclassify(SVMStruct, Sample)
```

Ogni riga della matrice Sample contiene un dato che viene classificato dalla funzione svmclassify usando le informazioni presenti nella struttura SVMStruct creata per mezzo della funzione svmtrain. Come nel caso della matrice Training, Sample è una matrice in cui ogni riga corrisponde a un'osservazione (singolo elemento) e ogni colonna corrisponde a una variabile che caratterizza il dato. Di conseguenza, Sample deve avere lo stesso numero di colonne dei dati di training dal momento che il numero di colonne definisce il numero di attributi caratteristici del dato.

Il vettore colonna Group indica in ogni riga la classe che viene assegnata, tramite classificazione, ad ogni riga di Sample.

4.2.1 Un semplice esempio: gli Iris di Fisher

Il caso degli Iris di Fisher è tra gli esempi più usati nella letteratura riguardante le Support Vector Machine. Nel 1916 Ronald Aylmer Fisher, matematico e biologo britannico, raccolse 150 iris appartenenti a 3 specie diverse (*setosa*, *virginica*, *versicolor*). Egli fu in grado di classificare correttamente ciascun iris e successivamente accostò a questo dato le misurazioni fatte sui petali e sui sepali del fiore. In sostanza abbiamo a disposizione un dataset contenente per ciascun fiore i seguenti dati:

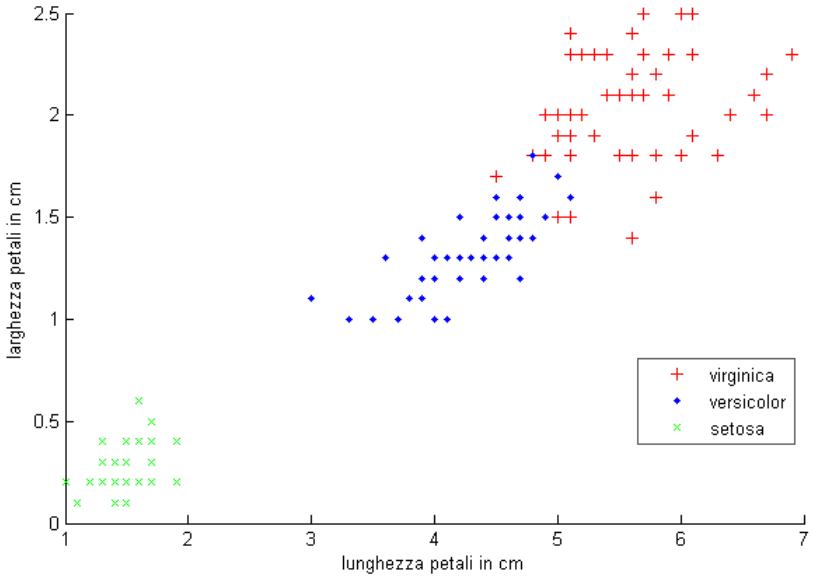

Figura 4.2.1: Dati di training relativi alla lunghezza e alla larghezza dei petali

1	2	3	4	5
lunghezza sepalo	larghezza sepalo	lunghezza petalo	larghezza petalo	classe

Sono presenti 50 esemplari di ogni specie.

A questo punto sorge spontanea una domanda: è possibile classificare correttamente una specie considerando solo le 4 misure fatte sul fiore?

In figura 4.2.1 è presente il grafico relativo alle informazioni riguardanti i petali ed è chiaramente visibile come i dati relativi alla specie "setosa" siano ben separati da quelli relativi alle altre due specie. Si tratta di dati separabili e una semplice SVM lineare permette di dividerli correttamente. Anche nel caso dei dati riguardanti le misurazioni fatte sui sepali, la specie *setosa* è ben divisibile dalle altre due specie. Successivamente vedremo come si comportano le macchine facenti uso delle funzioni kernel viste nei capitoli precedenti, per dividere il piano quando consideriamo solo le specie *versicolor* e *virginica*.

Per tracciare l'iperpiano separatore che divide la classe setosa dalle altre è stato innanzitutto necessario unire le classi “versicolor” e “virginica” in un unica classe “virginica/versicolor”. Successivamente è stata addestrata una SVM lineare con tutti i 150 dati a disposizione.

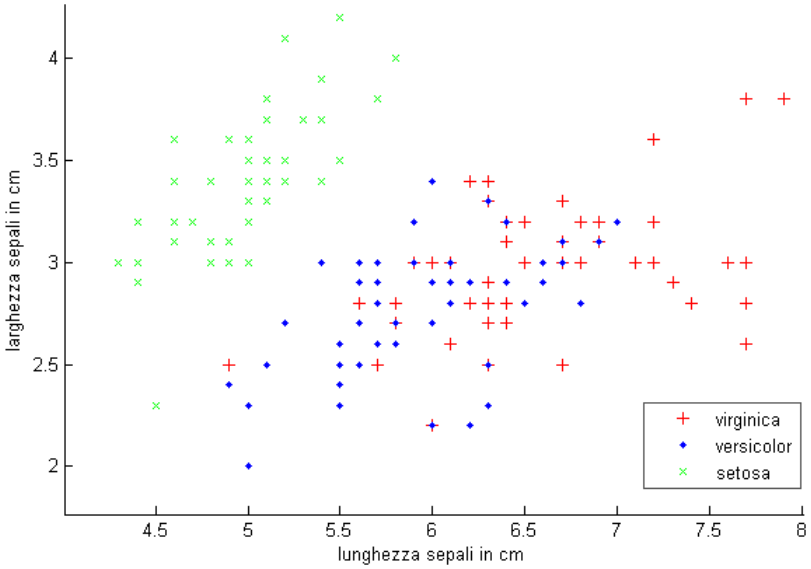

Figura 4.2.2: Dati di training relativi alla lunghezza e alla larghezza dei sepali

Algoritmo 4.1 Divide le specie in due categorie: setosa e virginica/versicolor in modo da poter utilizzare una SVM lineare per separare le due classi

```
load fisheriris
xdata = meas(1:end,3:4);
group = species(1:end);

for i=1:size(group)
    if(strcmp(species(i),'virginica') || strcmp(species(i),'
        versicolor'))
         group(i)={'virginica/versicolor'};
    end
end

svmstruct=svmtrain(xdata,group,'showplot',true,'kernel_function','
    linear','method','QP');
```

La matrice meas contiene tutte le misurazioni fatte sul fiore (Ogni riga corrisponde ad un fiore). Vengono considerate solo la 3° e 4° colonna di tale matrice, corrispondenti alle misurazioni fatte sui petali. Come si vede in figura 4.2.3, le due classi vengono correttamente divise. I dati cerchiati corrispondono ai support vector.

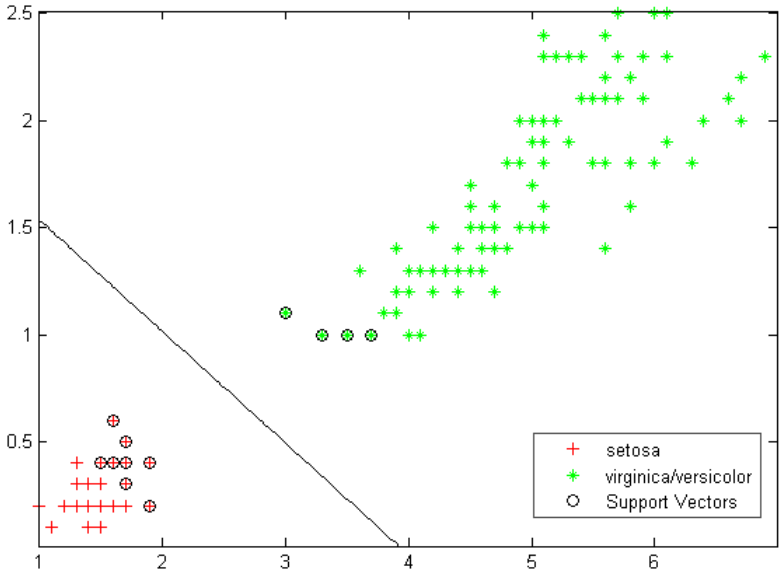

Figura 4.2.3: SVM lineare separa la specie "setosa" dalle altre basandosi sulle informazioni relative ai petali

Tornando ad osservare il grafico relativo alle misurazioni sui petali, si può portare la propria attenzione sulla ricerca di un metodo per suddividere nel modo migliore la specie virginica dalla specie versicolor.

Per fare ciò è stato utilizzato il seguente codice

Algoritmo 4.2 Divide la specie virginica dalla specie versicolor

```
load fisheriris
xdata = meas(51:end,3:4);
group = species(51:end);
%la seguente riga di codice varia a seconda della SVM che si vuole
     utilizzare
svmstruct=svmtrain(xdata,group,'showplot',true,'kernel_function','
    polynomial','polyorder',10,'method','QP');
```

La riga di codice evidenziata dal commento può essere modificata in modo da implementare i vari kernel a disposizione. Nella realizzazione della Figura 4.2.4 sono state utilizzate queste SVM:

- ```
  svmstruct=svmtrain(xdata,group,'showplot',true,'
      kernel_function','linear','method','QP');
  ```

  ```
  svmstruct=svmtrain(xdata,group,'showplot',true,'
      kernel_function','quadratic','method','QP');
  ```

- ```
  svmstruct=svmtrain(xdata,group,'showplot',true,'
      kernel_function','polynomial','polyorder',3,'method','QP')
      ;
  ```

  ```
  svmstruct=svmtrain(xdata,group,'showplot',true,'
      kernel_function','polynomial','polyorder',10,'method','QP'
      );
  ```

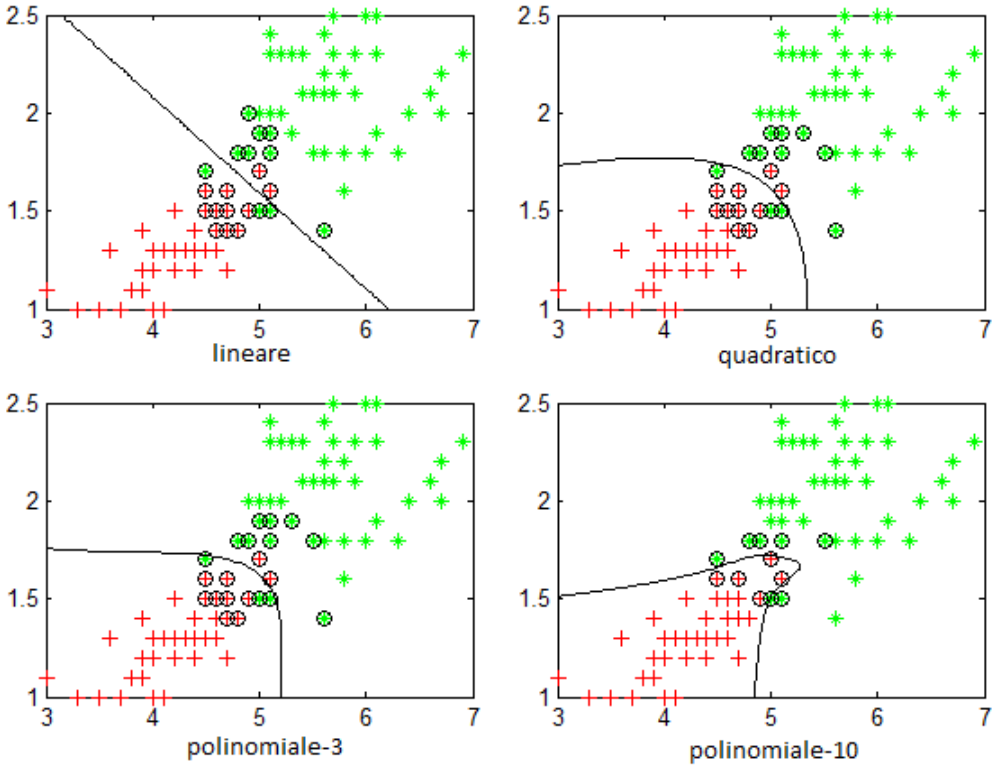

Figura 4.2.4: Quattro diversi tipi di SVM utilizzate per la divisione delle classi "virginica" (in verde) e "versicolor" (in rosso) basandosi su larghezza e lunghezza dei petali

Lo studio di SVM applicate al caso degli iris di Fisher è un buon esempio in quanto, essendo necessarie due sole variabili (lunghezza/larghezza petalo) per caratterizzare un dato, è possibile tracciare un grafico 2D che rende la suddivisione delle due classi molto intuitiva.

Nel prossimo caso non sarà possibile tracciare questo grafico in quanto le variabili prese in considerazione saranno più di due.

4.3 Applicazione di SVM allo studio di dati di una SPECT

La tomografia ad emissione di fotone singolo è una tecnica tomografica di imaging medico che utilizza la radiazione ionizzante nota come "raggi gamma". L'imaging SPECT viene eseguito utilizzando una *gamma camera* per acquisire, da più angoli, molteplici immagini 2D (dette proiezioni). Successivamente viene utilizzato un computer per eseguire un algoritmo di ricostruzione tomografica partendo dalle varie proiezioni, dando origine ad un dataset tridimensionale. La gamma camera consente l'acquisizione di immagini scintigrafiche che rappresentano visivamente la distribuzione, nel corpo, della radioattività emessa dai radiofarmaci iniettati nel paziente a scopo diagnostico o terapeutico.

Questi dati possono essere in seguito manipolati per mostrare sezioni sottili lungo un qualsiasi asse del corpo. Per catturare le immagini SPECT, la gamma camera viene ruotata attorno al paziente. Si prendono quindi diverse immagini planari nelle diverse proiezioni ottenute in punti definiti durante la rotazione (solitamente 3-6 gradi d'arco). In molti casi si esegue una rotazione di 360 gradi che consente di ottenere una ricostruzione tridimensionale ottimale.

La SPECT può essere utilizzata per qualsiasi studio di gamma imaging in cui può essere utile una reale rappresentazione 3D. È il caso di imaging tumorale, imaging della tiroide o delle ossa. Dal momento che la SPECT permette un'accurata localizzazione nello spazio 3D, può essere usata per fornire informazioni sulle funzioni localizzate degli organi interni, come la funzionalità cardiaca o l'imaging del cervello.

Il dataset a disposizione descrive tramite un insieme di parametri continui i risultati di analisi SPECT (Single Proton Emission Computed Tomography) del cuore effettuate su 267 diversi pazienti. Ogni paziente è classificato in

due categorie: normale e anormale, secondo quanto definito dal medico che ha analizzato le immagini.

Il risultato dell'elaborazione di queste immagini sono quindi 44 valori caratteristici continui compresi tra 0 e 100 (F1R - F22S) che vanno a descrivere la SPECT di ciascun paziente. Ad ogni paziente è inoltre assegnato un valore booleano 0 o 1 (OVERALL_DIAGNOSIS) per indicare se ad esso è stato diagnosticato o meno un problema.

Il set di dati è stato diviso in due: i dati relativi ad 80 pazienti sono stati usati per addestrare la SVM mentre i dati dei rimanenti 187 pazienti sono stati usati per testarla. A seconda del tipo di SVM impiegata si sono ottenuti differenti risultati di correttezza della previsione.

1	2	3	4	5	...	42	43	45
OVERALL_DIAGNOSIS	F1R	F1S	F2R	F2S	...	F21S	F22R	F22S

Il codice utilizzato è il seguente:

Algoritmo 4.3

```
load spectftrain.dat
load spectftest.dat

trainset=spectftrain(:,2:end);
class=[spectftrain(:,1)]';
testset=spectftest(:,2:end);
SVMStruct = svmtrain(trainset,class,'Kernel_Function','linear','
    method','QP');
Group = svmclassify(SVMStruct,testset);
```

Il programma completo, riportato in Appendice, dà come output il livello di accuratezza della SVM confrontando i dati ricavati nella fase di test con quelli reali:

```
corrette: 131, errate:56
accuratezza:70%
```

Utilizzando diversi kernel per la SVM si sono ottenuti i seguenti risultati di accuratezza:

- Lineare: 70%

- Quadratico: 45%
- Polinomiale-3: 45%
- Gaussiano: 86%

Le percentuali di accuratezza si riferiscono al confronto tra il risultato di classificazione dato dalla SVM e la diagnosi del medico. È interessante considerare che l'errore di classificazione può verificarsi per cause dovute all'addestramento o alla tipologia della SVM ma anche a causa di una una diagnosi medica non corretta.

Capitolo 5

Support Vector Machine per la regressione

Le Support Vector Machine possono essere applicate non solo a problemi di classificazione ma anche al caso della regressione. Una versione di SVM per la regressione è stata proposta nel 1996 da Vladimir N. Vapnik, Harris Drucker, Christopher J. C. Burges, Linda Kaufman e Alexander J. Smola e tale metodo è detto Support Vector Regression (SVR). Il modello prodotto dalla classificazione tramite SV dipende solo da un sottoinsieme dei dati di training, visto che la funzione di costo nella costruzione del modello non considera i punti di training che stanno oltre il margine. In modo analogo, il modello prodotto da SVR dipende solo da un sottoinsieme dei dati di training, perché la funzione di costo usata per costruire il modello ignora ogni errore che si trovi al di sotto di una soglia minima.

Supponiamo di avere una serie di dati di training:

$$\{\mathbf{x}_n, y_n\}\, n = 1, ..., N \quad \mathbf{x}_n \in \mathbb{R}^D \quad y_n \in \mathbb{R}$$

In una ε-SVM per la regressione il nostro scopo è quello di trovare una funzione $f(x)$ che abbia un grado di precisione ε ,con cui si vuole approssimare la funzione rappresentata dai dati di training, per mezzo del modello f.

In parole povere, non ci preoccupiamo degli errori fin quando questi sono più piccoli di ε, ma non accetteremo nessun errore più grande di esso. Il modello lineare è rappresentato dalla funzione:

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w} + b \quad w \in \mathbb{R}^D \quad b \in \mathbb{R}$$

Come nel caso delle SVM per la classificazione, lo scopo è quello di minimizzare la norma euclidea del vettore $\mathbf{w}$. Vogliamo che l'errore sia più piccolo di ε quindi dovrà valere:

$$|y_n - \mathbf{x}_n \cdot \mathbf{w} - b| \leq \varepsilon \tag{5.0.1}$$

Ci troviamo davanti a un problema di ottimizzazione convessa:

$$\text{minimizzare } \frac{1}{2} \|\mathbf{w}\|^2$$

$$\text{con vincoli } \begin{cases} y_n - \mathbf{x}_n \cdot \mathbf{w} - b \leq \varepsilon \\ \mathbf{w} \cdot \mathbf{x}_n + b - y_n \leq \varepsilon \end{cases}$$

dove i vincoli sono stati ottenuti a partire dalla condizione (5.0.1).

L'assunzione tacita nel problema di minimizzazione consiste nell'ipotizzare che tale funzione esista e che approssimi tutte le coppie $(\mathbf{x}_n, y_n)$ con una precisione ε. In alcuni casi questa assunzione non è vera ed è necessario permettere alcuni errori, introducendo le variabili di slack.

In modo analogo a quanto realizzato nel caso di dati non separabili in [Cortes and Vapnik(1995)] e illustrato nel capitolo 2.2, introduciamo allora le variabili di slack ξ_n, ξ_n^* per far fronte alle condizioni impossibili nel caso di ottimizzazione dovute alla non esistenza della funzione f. Arriviamo dunque a questa formulazione del problema:

$$\text{minimizzare } \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{n=1}^{N} (\xi_n + \xi_n^*)$$

$$\text{con vincoli } \begin{cases} y_n - \mathbf{x}_n \cdot \mathbf{w} - b \leq \varepsilon + \xi_n \\ \mathbf{x}_n \cdot \mathbf{w} + b - y_n \leq \varepsilon + \xi_n^* \\ \xi_n, \xi_n^* \geq 0 \end{cases}$$

La costante $C > 0$ pesa quanto gli errori di training (ossia gli errori più grandi di ε) sono tollerati. Questa formulazione è equivalente al considerare una particolare funzione di loss detta ε-insensitive $|\xi|_\varepsilon := \begin{cases} 0 & |\xi| \leq \varepsilon \\ |\xi| - \varepsilon & \text{altrimenti} \end{cases}$

La Figura 5.0.1 descrive la situazione graficamente. Solo i punti all'esterno della regione colorata corrispondono al costo. Si scopre che il problema di

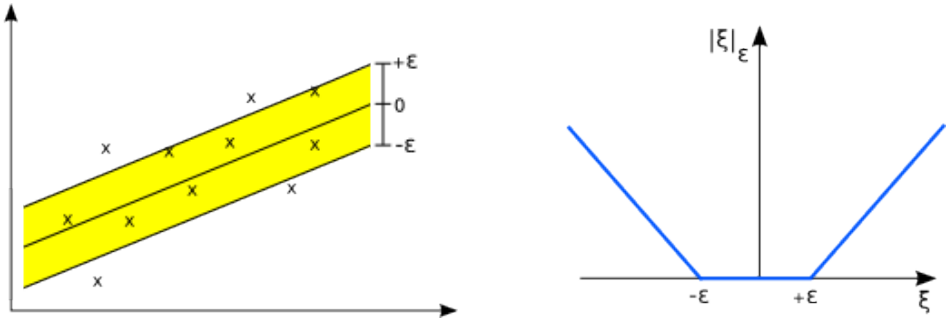

Figura 5.0.1: Soft margin e funzione $|\xi|_\varepsilon$

ottimizzazione appena visto può essere risolto più facilmente nella sua formulazione duale. Inoltre tale formulazione fornirà la chiave per estendere questa Support Vector Machine al caso di funzioni non lineari.

5.1 Formulazione duale e Programmazione Quadratica

L'idea basilare è quella di costruire una funzione di Lagrange a partire da entrambe le funzioni obiettivo e le corrispondenti condizioni, introducendo un insieme di variabili duali. Questa funzione di Lagrange viene detta primale. Si può mostrare che tale funzione ha un punto di sella rispetto alle variabili primali e duali nella soluzione ottimale [Goldstein(1986)].

Procediamo come segue:

$$L = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}(\xi_n + \xi_n^*) - \sum_{n=1}^{N}\alpha_n\left(\varepsilon + \xi_n - y_n + \mathbf{x}_n\cdot\mathbf{w} + b\right) - \sum_{n=1}^{N}\alpha_n^*\left(\varepsilon + \xi_n^* + y_n - \mathbf{x}_n\cdot\mathbf{w} - b\right) - \sum_{n=1}^{N}\left(\eta_n\xi_n + \eta_n^*\xi_n^*\right) \quad (5.1.1)$$

Anche in questo caso, tutti i moltiplicatori di Lagrange devono soddisfare la condizione di positività, ossia $\alpha_n, \alpha_n^*, \eta_n, \eta_n^* \geq 0$. Dalla condizione di punto di sella otteniamo che le derivate parziali di L rispetto alle variabili primali $\mathbf{w}, b, \xi_n, \xi_n^*$ devono essere nulle nel caso ottimale.

$$\partial_b L = \sum_{n=1}^{N} (\alpha_n^* - \alpha_n) = 0 \tag{5.1.2}$$

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{n=1}^{N} (\alpha_n - \alpha_n^*) \mathbf{x}_n = 0 \tag{5.1.3}$$

$$\partial_{\xi_n} = C - \alpha_n - \eta_n = 0 \tag{5.1.4}$$

$$\partial_{\xi_n^*} = C - \alpha_n^* - \eta_n^* = 0 \tag{5.1.5}$$

Sostituendo (5.1.2)-(5.1.5) nella formula (5.1.1) si ottiene il problema di ottimizzazione duale.

$$\text{massimizzare} \quad \begin{cases} -\frac{1}{2} \sum_{i,j=1}^{N} (\alpha_i - \alpha_i^*) \left(\alpha_j - \alpha_j^*\right) \mathbf{x}_i \cdot \mathbf{x}_j \\ -\varepsilon \sum_{n=1}^{N} (\alpha_n + \alpha_n^*) + \sum_{n=1}^{N} y_n (\alpha_n - \alpha_n^*) \end{cases}$$

$$\text{con vincoli} \quad \begin{cases} \sum_{n=1}^{N} (\alpha_n - \alpha_n^*) = 0 \\ \alpha_n, \alpha_n^* \in [0, C] \end{cases}$$

Nella derivazione di quest'ultimo problema di ottimizzazione sono state eliminate le variabili duali η_n, η_n^* secondo le condizioni (5.1.4) e (5.1.5), dal momento che queste variabili non apparivano più nella funzione obiettivo duale ma solo nelle condizioni. L'equazione (5.1.3) può essere riscritta come:

$$\mathbf{w} = \sum_{n=1}^{N} (\alpha_n - \alpha_n^*) \mathbf{x}_n$$

e quindi

$$f(\mathbf{x}) = \sum_{n=1}^{N} (\alpha_n - \alpha_n^*) \mathbf{x}_n \cdot \mathbf{x} + b \tag{5.1.6}$$

Sono utili a questo punto alcune considerazioni:

1. come si nota la funzione f è descritta come combinazione lineare dei dati di training $\mathbf{x}_n$;

2. la complessità della rappresentazione di una funzione tramite support

vector è indipendente da dimensione dello spazio di ingresso e dipende esclusivamente dal numero di support vector;

3. l'algoritmo può essere descritto in termini di prodotti cartesiani dei dati;

4. quando valutiamo $f(\mathbf{x})$ non dobbiamo calcolare esplicitamente $\mathbf{w}$.

Queste osservazioni torneranno utili nella formulazione del caso non lineare.

5.2 Calcolo di b

Finora abbiamo trascurato il problema di calcolare il valore del termine b. Ciò può essere fatto sfruttando le condizioni di Karush-Kuhn-Tucker che affermano, come già visto, che nella soluzione ottimale il prodotto tra le variabili duali e i vincoli deve essere nullo. Nel caso in questione questo fatto implica:

$$\alpha_n \left(\varepsilon + \xi_n - y_n + \mathbf{x}_n \cdot \mathbf{w} + b\right) = 0 \tag{5.2.1}$$

$$\alpha_n^* \left(\varepsilon + \xi_n^* + y_n - \mathbf{x}_n \cdot \mathbf{w} - b\right) = 0 \tag{5.2.2}$$

e

$$\left(C - \alpha_n\right) \xi_n = 0$$

$$\left(C - \alpha_n^*\right) \xi_n = 0$$

Questo permette di fare diverse conclusioni.

1. Solo i dati di training $(\mathbf{x}_n, y_n)$ corrispondenti a $\alpha_n = C$ e $\alpha_n^* = C$ stazionano all'esterno della zona ε-insensitive attorno a f.

2. Vale $\alpha_n \alpha_n^* = 0$, ossia non potrà mai esserci un insieme di variabili duali α_n, α_n^* che siano allo stesso tempo diverse da zero perché ciò richiederebbe variabili di slack diverse da zero in entrambe le direzioni.

3. Per $\alpha_n \in (0, C), \alpha_n^* \in (0, C)$ abbiamo $\xi_n = 0, \xi_n^* = 0$ e inoltre il secondo fattore in (5.2.1) e (5.2.2) è nullo.

Di conseguenza il parametro b può essere calcolato come segue:

$$b = y_n - \mathbf{x}_n \cdot \mathbf{w} - \varepsilon \quad \text{per } \alpha_n \in (0, C)$$

$$b = y_n - \mathbf{x}_n \cdot \mathbf{w} + \varepsilon \quad \text{per } \alpha_n^* \in (0, C)$$

In questo modo b risulta essere un sottoprodotto del processo di ottimizzazione.

5.3 Note conclusive

Dalle uguaglianze (5.2.1) e (5.2.2) segue che solo per $|f(\mathbf{x}_n) - y_n| \geq \varepsilon$ i moltiplicatori di Lagrange possono essere diversi da zero. Ossia per tutti i dati all'interno della zona ε-insensitive α_n, α_n^* sono nulli.

Per $|f(\mathbf{x}_n) - y_n| \leq \varepsilon$ il secondo fattore in (5.2.1) e (5.2.2) è non nullo quindi α_n, α_n^* devono per forza essere nulli per verificare le condizioni KKT. In sostanza abbiamo un'espansione di $\mathbf{w}$ in termini di $\mathbf{x}_n$ ma non abbiamo bisogno di tutti i dati $\mathbf{x}_n$ per descrivere $\mathbf{w}$.

I dati i cui rispettivi coefficienti sono non nulli, sono detti *support vector* e per essi valgono le stesse caratteristiche illustrate nel Capitolo 2 per la classificazione.

5.4 Non linearità e funzioni kernel

Dopo aver analizzato l'applicazione di una SVM lineare al caso della regressione, viene spontaneo chiedersi come estendere il tutto al caso non lineare, allo stesso modo di quanto si è visto nello studio della classificazione.

Anche in questo caso viene definita una funzione kernel $\phi : \chi \to \mathcal{F}$ che mappa i dati $\mathbf{x}_n$ dallo spazio di ingresso χ a uno spazio detto "feature space" $\mathcal{F}$. Successivamente sui dati mappati viene applicato l'algoritmo standard per la SVM per regressione.

Come già detto, l'algoritmo SVM dipende solo dal prodotto cartesiano tra i vettori dato. È quindi sufficiente conoscere ed utilizzare la funzione $K(\mathbf{x}, \mathbf{x}') := \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$ invece di conoscere esplicitamente $\Phi(\cdot)$. Questo permette di riscrivere l'algoritmo SVM per il caso della regressione in questo modo:

$$\text{massimizzare} \quad \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \alpha_i^*)\left(\alpha_j - \alpha_j^*\right) K(\mathbf{x}_i, \mathbf{x}_j) \\ -\varepsilon\sum_{n=1}^{N}(\alpha_n + \alpha_n^*) + \sum_{n=1}^{N} y_n(\alpha_n - \alpha_n^*) \end{cases}$$

$$\text{con vincoli} \quad \begin{cases} \sum_{n=1}^{N}(\alpha_n + \alpha_n^*) = 0 \\ \alpha_n, \alpha_n^* \in [0, C] \end{cases}$$

In questo modo l'espansione di f (5.1.6) può essere scritta come:

$$\mathbf{w} = \sum_{n=1}^{N}(\alpha_n - \alpha_n^*)\,\Phi(\mathbf{x}_n)$$

e quindi

$$f(\mathbf{x}) = \sum_{n=1}^{N}(\alpha_n - \alpha_n^*)\,K(\mathbf{x}_n, \mathbf{x}) + b$$

La differenza rispetto al caso lineare consiste nel fatto che $\mathbf{w}$ non è più un dato esplicito. Va evidenziato che nel caso non lineare il problema di ottimizzazione consiste nel cercare la funzione f nel *feature space* e non nello spazio di ingresso.

Gli algoritmi di addestramento delle SVM per la regressione e per la classificazione sono tuttavia simili. Nel caso pratico, l'addestramento di SVM per problemi di regressione è più complesso dell'addestramento di SVM per la classificazione, a causa del fatto che nel primo caso è necessario determinare simultaneamente i valori ottimali di due parametri, cioè di ε e di C.

Conclusioni

Le SVM sono uno degli algoritmi di apprendimento più promettenti nel campo dell'apprendimento automatico per via della loro semplicità. Esse possono essere utilizzate in svariati campi che vanno dalla classificazione di testi alla bioinformatica. Come abbiamo visto possono essere addestrate in modo semplice sia nel caso lineare che nel caso non lineare grazie all'utilizzo di funzioni kernel, lavorando in spazi di dimensioni maggiore.

Possiamo citare in conclusione alcune possibili estensioni di quanto mostrato.

In questa tesi sono state considerate solo macchine in grado di attuare una classificazione di tipo binaria. È possibile estendere quanto visto per introdurre SVM multiclasse che effettuano la classificazione di dati appartenenti a un insieme finito di classi. L'approccio utilizzato è quello di ridurre il problema multiclasse in diversi problemi di classificazione binaria ed applicare a ciascuno di essi una SVM binaria.

Un altro tipo di SVM sviluppata è la Trasductive SVM che è in grado di lavorare su un insieme di dati parzialmente classificati applicando un principio logico detto *transduction*.

Il campo è in continua evoluzione anche per quanto riguarda le tecniche computazionali per la risoluzione dei problemi di ottimizzazione delle funzioni.

Appendice

In questa appendice sono riportati i listati di codice MATLAB completi utilizzati negli esempi e nelle raffigurazioni del Capitolo 4.

SVM lineare realizzata con funzione quadprog

```
clear
dati1=[1 3 1;2 -4 1;3 6 1];
dati2=[7 7 -1;9 10 -1;8 -3 -1];
f=[0 0 0];
H=[1 0 0;0 1 0;0 0 0];

dati=[dati1;dati2];
tot=size(dati,1);
b=-ones(tot,1);
for i=1:tot
  A(i,1)=-dati(i,1)*dati(i,3);
  A(i,2)=-dati(i,2)*dati(i,3);
  A(i,3)=-dati(i,3);
end
w=quadprog(H,f,A,b);
hold on
x=-w(3)/w(1);
y=-w(3)/w(2);
m=-w(1)/w(2);
t=-2:0.1:2;
q=w(1)*x+w(3)-m*x;
k = 1/((w(1)^2+w(2)^2))^(1/2);

plot(x+t,(m*(t+x)+q),'red')

plot(dati1(:,1),dati1(:,2),'*','Color','g')
plot(dati2(:,1),dati2(:,2),'*','Color','b')
```

Codice MATLAB utilizzato per disegnare i dati di training relativi alle specie di iris

```
%Programma disegna un grafico per petali/sepali di tutte e 3 le
    specie di %iris
load fisheriris
%impostare 3:4 per i petali e 1:2 per i sepali
setg='petali';
if strcmp(setg,'petali')
  xdata = meas(1:end,3:4);
else
  xdata = meas(1:end,1:2);
end
group = species(51:end);

hold on
virg=1;
set=1;
vers=1;

for i=1:size(species)
  if (strcmp(species(i),'virginica'))
    virginica(virg,1)=xdata(i,1);
        virginica(virg,2)=xdata(i,2);
        virg=virg+1;
    end

    if (strcmp(species(i),'versicolor'))
    versicolor(vers,1)=xdata(i,1);
    versicolor(vers,2)=xdata(i,2);
        vers=vers+1;
    end

  if (strcmp(species(i),'setosa'))
    setosa(set,1)=xdata(i,1);
        setosa(set,2)=xdata(i,2);
        set=set+1;
  end
end

plot(virginica(:,1),virginica(:,2),'+','Color','red');
plot(versicolor(:,1),versicolor(:,2),'.','Color','blue');
plot(setosa(:,1),setosa(:,2),'x','Color','green');
```

```
legend('virginica','versicolor','setosa');
if strcmp(setg,'petali')
    xlabel('lunghezza petali in cm')
    ylabel('larghezza petali in cm')
else
  xlabel('lunghezza sepali in cm')
  ylabel('larghezza sepali in cm')
end
```

SVM per analisi dati SPECT

```
clear
load spectftrain.dat
load spectftest.dat

trainset=spectftrain(:,2:end);
class=[spectftrain(:,1)]';
testset=spectftest(:,2:end);

SVMStruct = svmtrain(trainset,class,'Kernel_Function','linear','
    method','QP');
Group = svmclassify(SVMStruct,testset);

for i=1:size(testset)
  A(i,1)=spectftest(i,1);
  A(i,2)=Group1(i);
end

dim=size(A);
dim=dim(1);
cor=0;
err=0;
fneg=0;
fpos=0;
for i=1:dim
  if(A(i,1)==A(i,2))
    cor=cor+1;
    else
        err=err+1;
    end
    if(A(i,1)==1 && A(i,2)==0)
        fneg=fneg+1;
    end
```

```
        if(A(i,1)==0 && A(i,2)==1)
            fpos=fpos+1;
        end
    end
    fprintf('accuratezza:%d%%\n',round(cor*100/(dim)));
```

Elenco delle figure

Bibliografia

[Abu-Mostafa et al.(2012)Abu-Mostafa, Magdon-Ismail, and Lin] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012. ISBN 1600490069, 9781600490064.

[Aizerman et al.(1964)Aizerman, Braverman, and Rozoner] A Aizerman, Emmanuel M Braverman, and LI Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25:821–837, 1964.

[Bache and Lichman(2013)] K. Bache and M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[Boser et al.(1992a)Boser, Guyon, and Vapnik] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992a.

[Boser et al.(1992b)Boser, Guyon, and Vapnik] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992b.

[Burges(1998)] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[Burges(1999)] Christopher JC Burges. Geometry and invariance in kernel based methods. *Advances in kernel methods-support vector learning*, pages 89–116, 1999.

[Cortes and Vapnik(1995)] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[Fletcher(1987)] Roger Fletcher. Practical methods of optimization. 1987.

[Fletcher(2009)] Tristan Fletcher. Support Vector Machines Explained. 2009.

[Goldstein(1986)] Herbert Goldstein. *Classical mechanics*. Addison-Wesley, Reading, MA, 1986.

[Hastie et al.(2001)Hastie, Tibshirani, and Friedman] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction: with 200 full-color illustrations*. New York: Springer-Verlag, 2001.

[Hearst et al.(1998)Hearst, Dumais, Osman, Platt, and Scholkopf] Marti A. Hearst, ST Dumais, E Osman, John Platt, and Bernhard Scholkopf. Support vector machines. *Intelligent Systems and their Applications, IEEE*, 13(4):18–28, 1998.

[MATLAB(2013a)] MATLAB. Classify using support vector machine (svm), 2013a. URL `http://www.mathworks.it/it/help/stats/svmclassify.html`.

[MATLAB(2013b)] MATLAB. Train support vector machine classifier, 2013b. URL `http://www.mathworks.it/it/help/stats/svmtrain.html`.

[Poggio(1975)] T Poggio. On optimal nonlinear associative recall. *Biological Cybernetics*, 19(4):201–209, 1975.

[Smola and Schölkopf(2004)] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[Wikipedia(2013)] Wikipedia. Single-photon emission computed tomography, 2013.