Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria dell'Informazione

Tesi di laurea triennale

Analisi di Support Vector Machines per la classificazione automatica

Laureanda:
MARINA RUOTOLO

Relatore:
Prof. LUCA SCHENATO

Anno Accademico 2011–2012

Indice

Elenco delle figure

Introduzione

L'apprendimento automatico è una disciplina che si sta sviluppando sempre di più ai nostri giorni e consiste nella ricerca di tecniche che consentano alle macchine di migliorare l'esecuzione di un compito a partire da semplici osservazioni della realtà. Ispirandosi alle capacità della mente umana, esso permette alle macchine di accumulare conoscenza tramite l'esperienza e successivamente di capire in modo automatico come valutare nuove informazioni. Per questo motivo, esso si inserisce nel contesto dell'Intelligenza Artificiale e gioca un ruolo chiave in molte aree della scienza, della medicina, della finanza e dell'industria.

Le Support Vector Machines (SVM), che nascono negli anni '90 grazie alla ricerca di Vladimir Vapnik e dei suoi collaboratori, andranno a risolvere il problema dell'apprendimento a partire da un *training set* di dati sperimentali di cui sono noti i parametri caratteristici. L'obiettivo sarà quindi quello di costruire un sistema che impari da dati già correttamente classificati e che poi grazie ad essi riesca a costruire una funzione di classificazione in grado di catalogare dati anche al di fuori di questo insieme. La caratteristica principale delle SVM, che le ha portate al successo immediato, è data dal fatto che esse, basandosi su semplici idee, permettono di giungere ad elevate prestazioni nelle applicazioni pratiche: sono cioè abbastanza semplici da analizzare matematicamente ma consentono di analizzare modelli complessi. L'algoritmo che permette di addestrarle, che come vedremo in seguito si può ricondurre ad un problema di programmazione quadratica con vincoli lineari, trova applicazione in ambiti diversissimi tra loro, tra cui i più comuni sono il riconoscimento di pattern, la catalogazione di testi e l'identificazione di facce in immagini.

Dopo qualche esempio di introduzione al problema della classificazione, in questo lavoro verrà approfondito l'aspetto della classificazione binaria di dati tramite l'utilizzo di SVM, prima di tutto dal punto di vista teorico e poi con qualche semplice applicazione in Matlab.

Capitolo 1

Il problema della classificazione

La classificazione, intesa come assegnazione di un pattern ad una determinata classe già nota a priori, è un tema di straordinaria importanza per la risoluzione di problemi del mondo reale. Essa può essere utilizzata in ambiti molto diversi tra loro, anche per problemi che a prima vista potrebbero sembrare di altro genere.

Un esempio tipico di applicazione è quello del riconoscimento delle cifre scritte a mano, ed in particolare delle cifre dei codici postali che vengono scritti sul retro delle buste. Nella Figura 1.1 si vedono delle cifre provenienti dagli ZIP codes scritti su alcune buste della posta statunitense: ogni immagine proviene da un codice di cinque cifre e, dopo essere stata isolata, è stata ricondotta a delle misure standard per quanto riguarda l'orientazione e la dimensione. Il problema da risolvere, che è quello di riconoscere l'identità di ogni cifra a partire da una matrice di pixel di intensità diversa, è proprio un problema di classificazione, in cui lo scopo è di assegnare ogni simbolo ad una delle dieci categorie possibili: $\{0, 1, \ldots, 9\}$.

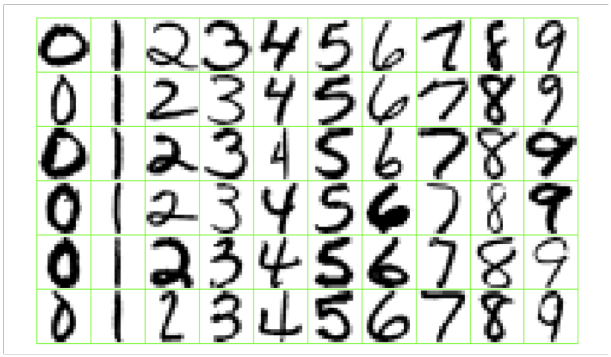


Figura 1.1: Esempio di cifre scritte a mano

Un altro esempio molto noto nell'ambito del riconoscimento e della classificazione è quello degli Iris di Fisher, noto matematico e biologo britannico. Il training set utilizzato nell'esperimento è formato dalle misure della lunghezza e della larghezza dei petali e dei sepali di 150 iris appartenenti a tre classi diverse (vedi Figura 1.2): 50 della specie *Setosa* (a), 50 della *Virginica* (b) e 50 della *Versicolor* (c). Nel problema di classificazione che Fisher affronta, l'obiettivo è di riuscire a classificare un iris nella giusta classe di appartenenza, avendo a disposizione solo le misure dei suoi petali o del suoi sepali. L'algoritmo risolutivo, come vedremo in seguito in una sua versione semplificata in cui ci si limiterà ad analizzare un sottoproblema di classificazione binaria, può essere costruito tramite l'uso delle SVM lineari, e permette, grazie all'apprendimento automatico dal training set di dati, di collocare un qualsiasi iris nella classe corretta.

(a) *Iris setosa* (b) *Iris virginica* (c) *Iris versicolor*

Figura 1.2: Gli Iris di Fisher

1.1 Errore e dimensione VC

Dal punto teorico, la classificazione consiste nella costruzione di una funzione $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ che, se applicata ad un campione x nello spazio $\mathbb{R}^n$ a cui è associato un parametro α reale, possa predirne la classe di appartenenza (usualmente indicata con un label $y \in \mathbb{R}$):

$$y_{pred} = f(x, \alpha) \tag{1.1}$$

Nel caso particolare in cui le possibili classi di appartenenza di un dato siano due, dato un insieme di campioni preclassificati presi da una distribuzione di probabilità sconosciuta $P(x, y)$, è importante scegliere tra tutte le funzioni $f(x, \alpha)$, quella che renda minimo l'errore teorico:

$$R(\alpha) = \int \frac{1}{2}|y - f(x, \alpha)| \, dP(x, y) \tag{1.2}$$

Esso, chiamato anche *rischio atteso*, è una misura di quanto l'ipotesi fatta sulla predizione di y_i per un punto x sia distante dal valore reale, ma, essendo $P(x, y)$ incognita, non può essere calcolato esplicitamente. Sapendo però che i campioni preclassificati del training set rappresentano un campione di $P(x, y)$, si può procedere ad un'approssimazione dell'errore:

$$R_{emp}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y - f(x, \alpha)| \tag{1.3}$$

Il *rischio empirico* $R_{emp(\alpha)}$ appena definito, per la legge dei grandi numeri converge a $R(\alpha)$ e, siccome nella sua espressione non compare la distribuzione di probabilità, può essere facilmente calcolato con i dati a disposizione. Per ottenere l'errore teorico minimo, però, non è sufficiente minimizzare l'errore empirico, ma è necessario anche rendere minima la dimensione VC (dal nome dei suoi ideatori Vapnik-Chervonenkis). Quest'ultima può essere intesa come la complessità del classificatore ed è pari al numero di vettori che il esso è in grado di separare in due classi (ad esempio per un classificatore che opera in $\mathbb{R}^2$ la dimensione VC è 3).

Degli esempi di algoritmi in grado di risolvere questo problema sono proprio le SVM: esse infatti sono in grado di minimizzare il numero di errori sul training set ed allo stesso tempo di rendere piccola la dimensione VC.

1.2 La classificazione binaria

In generale, per poter utilizzare gli algoritmi di apprendimento automatico, la prima cosa da fare è trovare quale sia il miglior criterio per separare le classi di dati del training set (fase di training). Una volta ottenuto il separatore ottimo si potrà poi passare alla fase di test vera e propria, in cui, dato un campione, gli si assegnerà un pattern corrispondente alla sua classe di appartenenza.

Il caso più semplice da analizzare, che sta alla base del funzionamento delle SVM e a cui si potrà fare riferimento anche per la risoluzione di problemi più complessi, è quello della classificazione binaria. Formalizzando quindi il problema, in presenza di due sole alternative il training set sarà formato da coppie di dati e sarà del tipo: $\{(x_1, y_i), \ldots, (x_n, y_i)\} \in \mathbb{R}^n \times \{\pm 1\}$, dove con x_i vengono indicati i campioni e con y_i i *label* della corrispondente classe di appartenenza. Lo scopo sarà quello di utilizzare questi dati per trovare una funzione, il cui codominio sia $\{\pm 1\}$, che permetta di classificare nuovi dati: a seconda che essa valga +1 o -1 il punto risulterà appartenere all'una o all'altra classe.

1.2.1 Gli iperpiani separatori

Il primo caso da analizzare è quello in cui i campioni del training set sono linearmente separabili. Il metodo da utilizzare per risolvere il problema sarà quello di determinare un *iperpiano* che separi le classi di dati, in modo che i punti si possano suddividere tra due semispazi (vedi Figura 1.3). Chiamando $\mathbf{w}$ il vettore normale all'iperpiano e b l'intercetta all'origine, tutti i punti che giacciono sull'iperpiano separatore dovranno soddisfare l'equazione:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{1.4}$$

e la funzione di decisione corrispondente sarà:

$$f(x) = sign(\mathbf{w} \cdot \mathbf{x} + b) \tag{1.5}$$

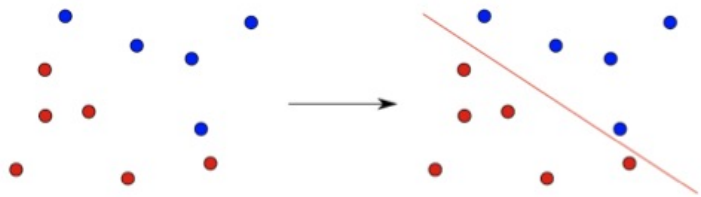


Figura 1.3: Classificazione binaria lineare

Nella maggior parte dei casi pratici bisogna anche tener conto che ci possono essere degli errori nei dati sperimentali. Se ci sono dei punti che risultano nel semipiano sbagliato, infatti, una classificazione lineare esatta è pressoché impossibile e sono necessarie delle tecniche di approssimazione che cerchino di rendere minimo il numero di errori.

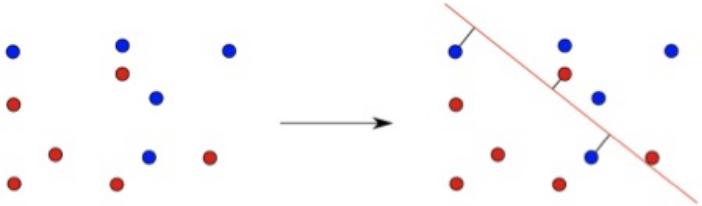


Figura 1.4: Classificazione binaria lineare con errori

1.2.2 Spazi immagine e kernel

Per utilizzare la classificazione tramite iperpiani anche per dati che avrebbero bisogno di funzioni non lineari per essere separati, è necessario ricorrere alla tecnica degli spazi immagine (*feature spaces*). Questo metodo, che sta alla base della teoria delle SVM, consiste nel mappare i dati iniziali in uno spazio di dimensione superiore. Presupponendo quindi $m > n$, per la mappa si utilizza una funzione

$$\Phi \colon \mathbb{R}^n \to \mathbb{R}^m \tag{1.6}$$

Come si può notare dall'esempio in Figura 1.5, le due classi nello spazio di input non sono linearmente separabili, ma attraverso la funzione Φ i dati vengono mappati in uno spazio in cui diventano linearmente separabili e in cui sarà possibile trovare un iperpiano che li separi.

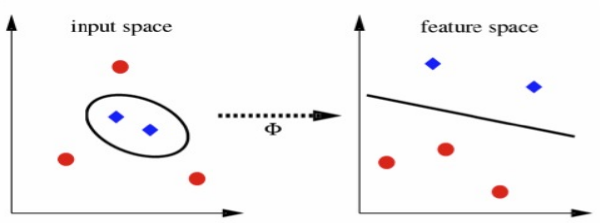


Figura 1.5: Esempio di feature space

La tecnica degli spazi immagine è particolarmente interessante per algoritmi che utilizzano i dati di training x_i solo attraverso prodotti scalari $\mathbf{x}_i \cdot \mathbf{x}_j$. In questo caso nello spazio $\mathbb{R}^m$ non si devono trovare esplicitamente $\Phi(\mathbf{x}_i)$ e $\Phi(\mathbf{x}_j)$ ma basta calcolare il loro prodotto scalare $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. Per rendere semplice quest'ultimo calcolo, che in spazi di dimensioni elevate diventa molto complicato, si utilizza una funzione detta *kernel* che restituisce direttamente il prodotto scalare delle immagini:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \tag{1.7}$$

In generale i kernel più utilizzati sono:

- Lineare: $K(\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y}$
- Polinomiale: $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$ oppure $K(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x} \cdot \mathbf{y})^d$
- Gaussian Radial Basis function: $K(\mathbf{x}, \mathbf{y}) = exp(-|\mathbf{x} - \mathbf{y}|^2)/(2\sigma^2)$
- Sigmoid: $K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x} \cdot \mathbf{y} - \delta)$

Capitolo 2

Support Vector Machines lineari

Le Support Vector Machines sono dei classificatori molto potenti che permettono di ricondursi alla classificazione lineare anche se si devono risolvere problemi di gran lunga più complessi. Nel caso binario, quando si ha a disposizione un training set di dati $\{(x_1, y_i), \ldots, (x_n, y_i)\} \in \mathbb{R}^n \times \{\pm 1\}$ e si cerca una funzione che possa classificarli al meglio, l'algoritmo risolutivo delle SVM, utilizzando la programmazione quadratica, permette di trovare i parametri caratteristici del miglior separatore delle due classi.

2.1 Dati linearmente separabili

Il caso più semplice da analizzare è quello in cui i dati del training set sono linearmente separabili. Come visto in precedenza, esiste sempre un iperpiano del tipo $\mathbf{w} \cdot \mathbf{x} + b = 0$ che separa perfettamente i campioni in due insiemi distinti. Si suppone cioè che tutti i dati del training set soddisfino le seguenti condizioni:

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq +1 \quad se \quad y_i = +1 \tag{2.1}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 \quad se \quad y_i = -1 \tag{2.2}$$

che si possono unire in un'unica disuguaglianza:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0 \quad \forall i \tag{2.3}$$

Teoricamente esistono infiniti iperpiani che soddisfano queste equazioni, ma per una classificazione corretta è necessario determinare i parametri caratteristici $(\mathbf{w}, b)$ dell'iperpiano che tra tutti separa i dati al meglio.

2.1.1 Margine di separazione massimo

In Figura 2.1 si vedono due classi di dati separate da due iperpiani diversi. A prima vista entrambe le soluzioni sembrerebbero buone, ma si può notare che mantenendo più ampio il corridoio tra le due classi il rischio di overfitting è minore e di conseguenza la classificazione di dati non appartenenti al training set risulterà più precisa.

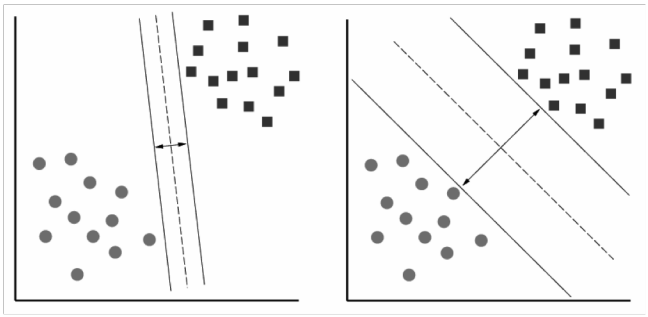


Figura 2.1: Classificazione tramite iperpiani diversi

Tenendo presente quanto detto, consideriamo i punti che soddisfano l'uguaglianza nell'Eq. (2.1): essi giacciono tutti sull'iperpiano $H_1 : \mathbf{w} \cdot \mathbf{x}_i + b = 1$. I punti per cui vale l'uguaglianza nell'Eq (2.2), invece, giacciono sull'iperpiano $H_2 : \mathbf{w} \cdot \mathbf{x}_i + b = -1$. H_1 e H_2 risultano paralleli e nello spazio che li separa non cadono punti del training. La distanza tra i due piani inoltre, che tramite la geometria si deduce essere uguale a $\frac{2}{\|\mathbf{w}\|}$, viene definita *margine*.

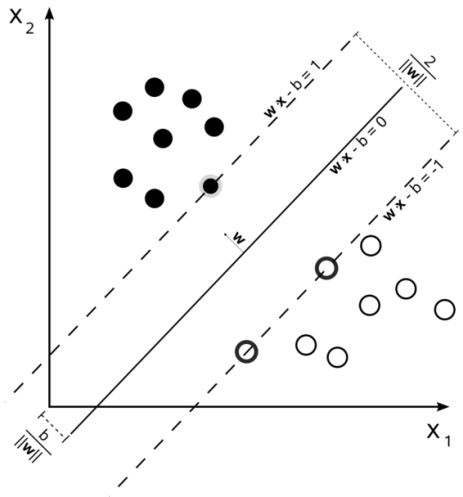


Figura 2.2: Esempio di separazione lineare con SVM

L'algoritmo che le SVM utilizzano per trovare i due parametri del separatore ottimo consiste proprio nel rendere massimo il margine appena definito, per fare in modo che il corridoio tra le due classi risulti il più ampio possibile. Un primo metodo può essere direttamente quello di cercare i due iperpiani che minimizzano $\|\mathbf{w}\|^2$, con $\mathbf{w}$ e b soggetti ai vincoli dell'Eq. (2.3). Il problema di minimo vincolato da risolvere diventa un problema di programmazione quadratica e può essere formalizzato nel modo seguente:

$$\begin{cases} \min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \\ y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq +1 \quad i = 1, \ldots, N \end{cases} \tag{2.4}$$

dove il termine $\frac{1}{2}$ è stato aggiunto per semplicità di calcolo.

2.1.2 Lagrangiana

Per rendere possibile la generalizzazione di quanto detto finora anche nel caso non lineare in cui verranno utilizzate le funzioni kernel, è necessaria una formulazione Lagrangiana del problema, grazie alla quale i dati appariranno solo sotto forma di prodotto scalare. Nella funzione da minimizzare vengono quindi introdotti dei moltiplicatori di Lagrange α_i, uno per ogni vincolo dell'Eq. (2.3). La Lagrangiana ottenuta è la seguente:

$$L_p(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b)] + \sum_{i=1}^{N} \alpha_i \tag{2.5}$$

In apparenza la funzione risulta molto complicata ma darà la possibilità di semplificare notevolmente i calcoli da eseguire. Per risolvere il problema di ottimizzazione, la prima cosa da fare è minimizzare L_p rispetto a $\mathbf{w}, b$ e contemporaneamente fare in modo che le derivate di L_p rispetto a tutti gli α_i si azzerino, il tutto soggetto al vincolo $\alpha_i > 0$ (questo particolare insieme di vincoli verrà chiamato Γ_1). Si nota che la funzione Lagrangiana in oggetto è una funzione quadratica convessa e che quindi il problema preso in esame anche in questo caso risulterà essere di programmazione quadratica ed avrà un'unica soluzione. Alla luce di ciò è possibile ricorrere ad una sua espressione duale: si può cioè massimizzare L_p, imponendo che il suo gradiente rispetto a $\mathbf{w}, b$ risulti 0 e che $\alpha_i > 0$ (insieme di vincoli Γ_2). Questa doppia formulazione del problema ha una proprietà fondamentale: il minimo di L_p soggetto al vincolo Γ_1 dà come risultato esattamente gli stessi $\alpha_i, \mathbf{w}, b$ del massimo di L_p soggetto al vincolo Γ_2.

Chiamato $\Lambda = (\alpha_1, \ldots, \alpha_N)$ il vettore dei moltiplicatori, si può formalizzare il problema di massimo descritto: per prima cosa si calcolano le derivate

parziali dell'Eq. (2.5) rispetto a $\mathbf{w}, b$ e si pongono uguali a 0:

$$\frac{\partial L_p(\mathbf{w}, b, \alpha_i)}{\partial(\mathbf{w})} = \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i = 0 \tag{2.6}$$

$$\frac{\partial L_p(\mathbf{w}, b, \alpha_i)}{\partial(b)} = \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.7}$$

Da queste si ottengono le seguenti condizioni:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{2.8}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.9}$$

Sostituendole poi nell'Eq. (2.5), si arriva ad una nuova espressione:

$$L_d(\Lambda) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \tag{2.10}$$

La funzione L_d così ottenuta è la formulazione duale del problema (per sottolinearlo è stato cambiato il pedice della funzione da p a d): la soluzione infatti può essere trovata massimizzando L_d rispetto a Λ, imponendo che il vincolo (2.9) sia verificato e che tutti i moltiplicatori di Lagrange rispettino la condizione $\alpha_i > 0$. In questo modo si otterrà il vettore $\mathbf{w}$ dato dall'espressione (2.8).

Per trovare esplicitamente l'altro parametro (il valore della soglia b) sarebbe necessario ricorrere ad una delle condizioni di Karush-Kuhn-Tucker per i problemi di ottimizzazione, che però vanno al di là della trattazione di questo lavoro.

2.1.3 I Support Vectors

Tutti i punti che nella soluzione precedente soddisfano la condizione $\alpha_i > 0$ vengono chiamati vettori di supporto (*support vectors*) e giacciono su uno dei due iperpiani H_1, H_2. Essi, essendo i più vicini alla frontiera di decisione, sono punti critici per le SVM. Tutti gli altri punti, invece, hanno il corrispondente $\alpha_i = 0$ e non influenzano in alcun modo il classificatore: se infatti venissero rimossi o spostati senza oltrepassare la soglia di H_1 e H_2 e l'algoritmo fosse ripetuto, si otterrebbe esattamente lo stesso iperpiano separatore.

2.2 Dati non linearmente separabili

Se nel training set di dati sono presenti dei campioni anomali che si trovano nel semipiano sbagliato e la classificazione binaria tramite iperpiani non può essere eseguita in maniera corretta, si parla di dati non linearmente separabili. Per estendere il metodo di risoluzione visto in precedenza anche in questo caso, è necessario rendere più flessibili i vincoli definendo delle variabili *slack* $\xi_i \geq 0$, tanto maggiori quanto più lontani sono i punti anomali. Si suppone quindi che i dati del training set rispettino i seguenti vincoli

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq +1 - \xi_i \quad se \quad y_i = +1 \tag{2.11}$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 + \xi_i \quad se \quad y_i = -1 \tag{2.12}$$

riassunti in:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i \tag{2.13}$$

Per ogni campione anomalo, la corrispondente ξ_i deve essere più grande dell'unità ($\xi_i \geq 1$) e $\sum_i \xi_i$ risulterà quindi un limite superiore per il numero di errori possibili nel training set.

Anche la funzione costo da minimizzare varia rispetto al caso di dati linearmente separabili e diventa:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\Big(\sum_i \xi_i\Big)^k \tag{2.14}$$

con C e k parametri scelti a priori. É noto che ad un alto valore di C corrisponde un elevato peso degli errori commessi e che per $k = 1, 2$ il problema rimane di programmazione quadratica e quindi per risolverlo sarà possibile utilizzare il metodo della Lagrangiana precedentemente descritto. Se si sceglie $k = 1$ si ha il vantaggio che le variabili ξ_i non compaiono nella formulazione duale del problema, per cui la funzione L_d da massimizzare risulta identica a quella già analizzata nel caso in cui i dati del training set erano separabili linearmente:

$$L_d(\Lambda) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \tag{2.15}$$

L'unica differenza da sottolineare si ha nei coefficienti di Lagrange; essi infati, oltre ad essere limitati inferiormente, sono limitati anche superiormente dal parametro C. I vincoli a cui è soggetta l'equazione diventano quindi:

$$0 \leq \alpha_i \leq C \tag{2.16}$$

$$\sum_i \alpha_i y_i = 0 \tag{2.17}$$

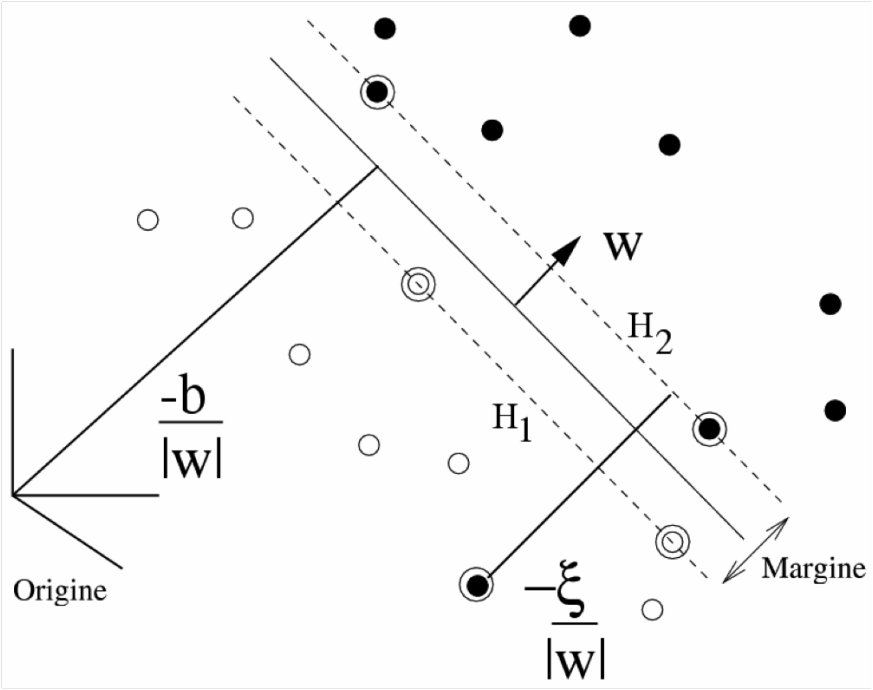


Figura 2.3: Classificatore lineare per campioni non linearmente separabili

In Figura 2.3 è rappresentato un esempio di un iperpiano separatore per un un insieme di dati non linearmente separabili. Il piano viene determinato tramite i support vectors (punti cerchiati) e ha distanza $-\frac{b}{\|\mathbf{w}\|}$ dall'origine. Ogni punto che risulta essere mal classificato, invece, ha distanza $-\frac{\xi}{\|\mathbf{w}\|}$ dalla sua classe di appartenenza.

2.3 Fase di test

Dopo la fase di apprendimento in cui, grazie alle tecniche descritte, si ottengono i parametri dell'iperpiano ottimo di separazione, le SVM procedono alla classificazione vera e propria di nuovi dati. Questa fase viene chiamata fase di test e consiste nel collocare nella giusta classe un campione arbitrario $\mathbf{x}$, anche non appartenente ai dati di training, proprio come farebbe la mente umana dopo aver immagazzinato informazioni tramite l'esperienza. Sempre rimanendo nell'ambito della classificazione binaria, per raggiungere lo scopo in genere si utilizza la seguente funzione di decisione:

$$f(x) = sign(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.18}$$

Essa permette di assegnare ad ogni dato preso in esame un label (± 1) a seconda della classe di appartenenza e, se si sostituisce al posto di $\mathbf{w}$ l'espressione ottenuta nell'Eq. (2.8), si ottiene una funzione di decisione più generale che contiene i campioni solo sotto forma di prodotto scalare:

$$f(x) = sign\Big(\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b\Big) \tag{2.19}$$

Quest'ultima espressione, come si verrà più in particolare nel capitolo successivo, risulta molto utile nel caso in cui la funzione da utilizzare per separare i dati del training set non sia lineare.

Capitolo 3

Support Vector Machines non lineari

Nel caso in cui si abbia a disposizione un training set di dati appartenenti a due classi che possono essere separate in maniera non lineare (ad esempio tramite una funzione quadratica, cubica o addirittura di grado superiore) la situazione va analizzata in maniera diversa. A prima vista le tecniche di risoluzione discusse finora sembrano insufficienti per poter classificare correttamente questi dati, ma grazie allo stratagemma matematico dei kernel, le SVM renderanno possibile la generalizzazione di quanto visto in precedenza anche per questa classe di problemi.

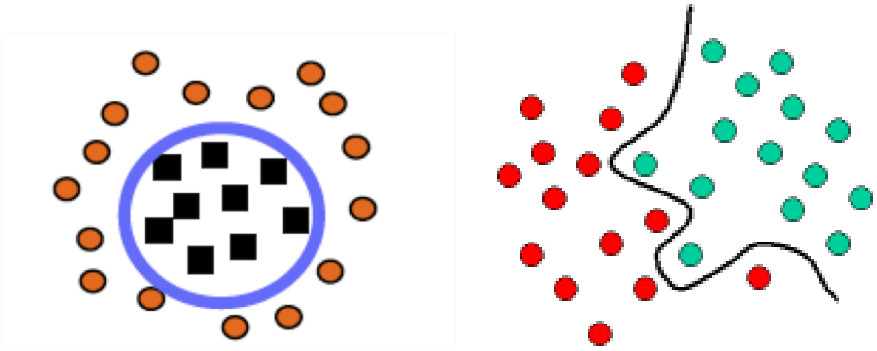


Figura 3.1: Dati separabili non linearmente

Come già visto, se si mappano i dati in uno spazio di dimensione superiore grazie ad una funzione $\Phi\colon \mathbb{R}^n \to \mathbb{R}^m$ con $m > n$, i dati diventano linearmente separabili nel nuovo spazio e sarà quindi possibile trovare i parametri di un iperpiano che li separi in $\mathbb{R}^m$. A questo punto il procedimento da utilizzare risulta lo stesso dei dati linearmente separabili, ma per rendere agevoli i

calcoli del problema di programmazione quadratica nello spazio $\mathbb{R}^m$, sarà necessario utilizzare la funzione Lagrangiana L_d del problema duale, nella quale i dati del training set appaiono solo sotto forma di prodotto scalare. Nel nuovo spazio $\mathbf{x}_i \cdot \mathbf{x}_j$ sarà sostituito da una forma del tipo $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$, ma, se m è molto grande, i vettori $\Phi(\mathbf{x})$ aumenteranno notevolmente di dimensione e le operazioni da fare diventeranno troppo laboriose. L'idea innovativa delle SVM sta proprio nel non eseguire esplicitamente questi calcoli, utilizzando lo strumento delle funzioni kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \tag{3.1}$$

Esse infatti, restituendo il prodotto delle immagini senza mai eseguire il prodotto tra i vettori e ignorando la forma esplicita della funzione di mapping, permettono di semplificare notevolmente i calcoli e di effettuare efficacemente il training anche nello spazio di dimensione m. Per verificare quanto detto è sufficiente un semplice esempio pratico: supponendo di avere due vettori $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$ con un kernel polinomiale del tipo

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^2 \tag{3.2}$$

e scegliendo come funzione di mapping $\Phi \colon \mathbb{R}^2 \to \mathbb{R}^3$:

$$\Phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2^2 \\ x_2^2 \end{pmatrix}$$

se si calcola la funzione kernel e si ottiene la seguente funzione che si nota essere uguale proprio al prodotto delle immagini $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$:

$$(\mathbf{x} \cdot \mathbf{y})^2 = \left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \right)^2 = (x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}) \tag{3.3}$$

Non tutti i kernel però sono validi e potrebbe capitare che in alcuni casi il problema di ottimizzazione non abbia soluzioni. Per accertarsi che la funzione kernel scelta sia appropriata bisogna controllare che esso soddisfi la condizione dettata dal *Teorema di Mercer*. Esso assicura l'esistenza di una mappa Φ ed un'espansione

$$K(\mathbf{x}, \mathbf{y}) = \sum_i \Phi(\mathbf{x})_i \cdot \Phi(\mathbf{y})_i \tag{3.4}$$

se e solo se, data una $g(\mathbf{x})$ per cui l'integrale $\int g(\mathbf{x})^2\, dx$ sia finito, si ha:

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y})\, dx\, dy \geq 0 \tag{3.5}$$

3.1 Fase di test

Dopo aver risolto il problema di ottimizzazione, si avranno a disposizione i parametri del separatore e si avranno quindi tutti gli strumenti per costruire un classificatore binario non lineare. La funzione di decisione da utilizzare è la stessa del caso precedentemente analizzato (Eq. (2.19)) dove però tutti i prodotti scalari verranno sostituiti dai kernel corrispondenti. Si otterrà quindi il seguente classificatore:

$$f(x) = sign\Big(\sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_j) + b\Big) \tag{3.6}$$

3.2 Algoritmo delle SVM in sintesi

Riassumendo quanto visto si possono dedurre delle regole generali che le SVM utilizzano per la classificazione non lineare dei dati:

- Scegliere il parametro C che rappresenta un compromesso tra la minimizzazione dell'errore sul training set e la massimizzazione del margine
- Scegliere la funzione kernel da utilizzare
- Risolvere il problema di programmazione quadratica che, essendo la funzione da minimizzare convessa, darà sicuramente una soluzione unica e globale
- Classificare nuovi campioni tramite la funzione di decisione

Negli anni sono stati sviluppati diversi algoritmi per risolvere il problema di programmazione quadratica. Negli esempi di applicazione del capitolo successivo è stato deciso di utilizzare una funzione di *Matlab* che, con un'opportuna programmazione, consente di minimizzare al meglio la funzione in oggetto in maniera efficiente.

Capitolo 4

Applicazioni

Dopo aver discusso del funzionamento delle SVM dal punto di vista teorico, ora, grazie all'utilizzo di *Matlab*, verrà analizzata qualche applicazione. Lo scopo sarà quello di effettuare la fase di training, trovando e visualizzando in due dimensioni la retta che divide al meglio due classi di dati. Per risolvere in maniera efficiente il problema di programmazione quadratica si utilizzerà la funzione *quadprog* che permetterà di trovare i parametri dell'iperpiano ottimo di separazione prima per classi che si possono separare linearmente in maniera precisa e poi, con i dovuti aggiustamenti, anche in presenza di errori. Questi parametri poi, se sostituiti nella funzione di decisione, permetteranno di capire la classe di appartenenza di un qualsiasi campione di input, anche non appartenente al training set.

4.1 Dati linearmente separabili

Come primo esempio è stata presa in esame una classe formata da tre punti rossi ed un'altra da tre punti blu, pensati in modo da essere linearmente separabili tra di loro. Per poterle rappresentare in un grafico di *Matlab* sono state espresse con delle matrici di tre colonne, le prime due indicanti le coordinate dei punti sul piano e la terza il label della classe corrispondente (1 per la classe R dei punti rossi e -1 per la classe B dei punti blu):

```
R=[7 9 1;
   3 8 1;
   4 6 1]
B=[-3 4 -1;
   4 2  -1;
   1 -3 -1]
```

Rappresentando in due dimensioni il training set di dati si ottiene quindi il seguente grafico:

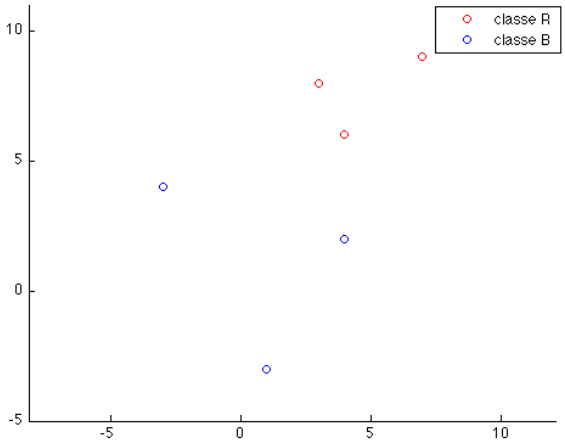


Figura 4.1: Due classi linearmente separabili

Ora, vista la separabilità delle due classi, per simulare il comportamento delle SVM, è necessario trovare i parametri $\mathbf{w} = \begin{bmatrix} w_1 & w_2 \end{bmatrix}$ e b della retta $\mathbf{w} \cdot \mathbf{x} + b = 0$ che massimizza il margine. Come già visto, il problema è quello di minimizzare la funzione $\frac{1}{2}\|\mathbf{w}\|^2$ soggetta al vincolo $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ e risolverlo sarà molto semplice grazie all'utilizzo della funzione *quadprog* di *Matlab*. Una volta inseriti parametri opportuni, essa permette di trovare il minimo di una funzione del tipo:

$$\frac{1}{2} z^T H z + f^T z \tag{4.1}$$

soggetta ad un vincolo:

$$Az \leq b \tag{4.2}$$

Si nota però che la funzione da minimizzare ed i vincoli presenti nel problema delle SVM risultano essere di forma leggermente diversa da quella richiesta da *Matlab*; chiamato quindi $\mathbf{z} = \begin{bmatrix} w_1 & w_2 & b \end{bmatrix}$ il vettore delle incognite, è necessario definire con molta attenzione i vettori e le matrici da inserire nella funzione *quadprog*. Tenendo conto del fatto che nella funzione da minimizzare non è presente il termine di primo grado e che i vincoli sono espressi da una disequazione nel verso opposto rispetto a quella di *Matlab*, il codice che è stato utilizzato per impostare il problema di programmazione quadratica è il seguente:

```
tot=size(R,1)+size(B,1)
H=[1 0 0;
   0 1 0;
   0 0 0]
f=[0 0 0]
campioni=[R;
          B]
for i=1:tot
    A(i,1)=-campioni(i,1)*campioni(i,3);
    A(i,2)=-campioni(i,2)*campioni(i,3);
    A(i,3)=-campioni(i,3);
end
b=-ones(tot,1)
```

A questo punto si può finalmente utilizzare la funzione sopracitata per trovare i parametri del separatore ottimo imponendo:

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

Trovate poi le intersezioni della retta separatrice con gli assi, la si può disegnare tramite la funzione *line*. Per verifica è stato deciso di disegnare anche le rette che delimitano il margine massimo: per farlo si può usare sempre la funzione *line*, ricordandosi che la loro distanza dalla retta separatrice è $\frac{1}{\|\mathbf{w}\|}$. In Figura 4.2 si nota come esse, equidistanti dal separatore e passanti per i vettori di supporto, formino un corridoio al cui interno non compaiono dati.

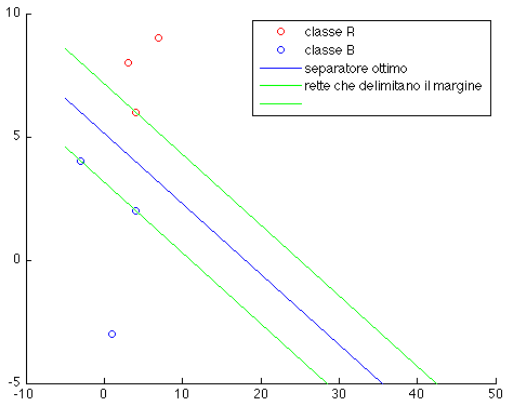
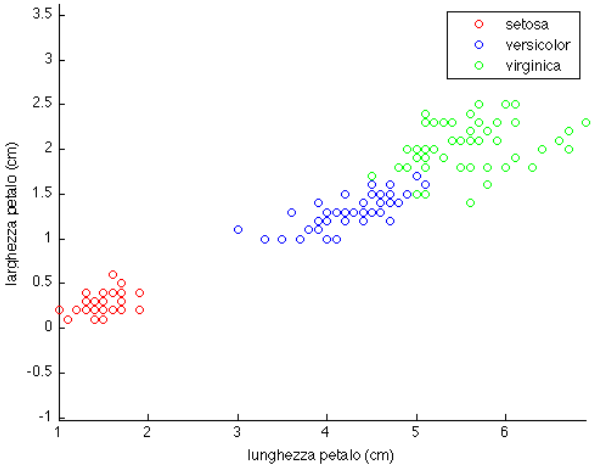


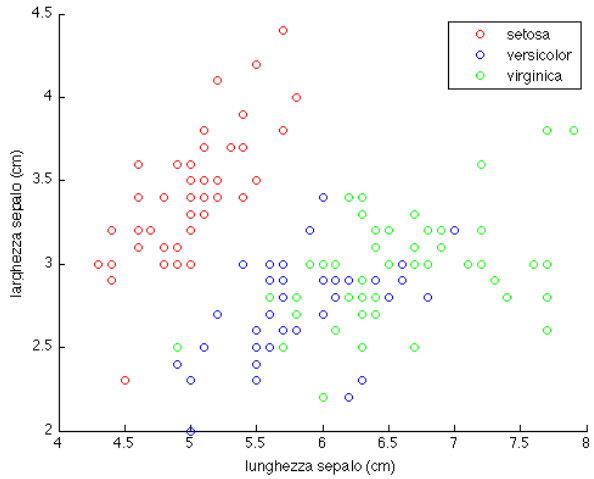
Figura 4.2: Separatore ottimo per dati linearmente separabili

4.2 Gli iris di Fisher

Viene considerato ora il data set degli iris di Fisher [5], già presentata nel Capitolo 1. Visualizzando in rosso gli iris della specie *setosa*, in blu quelli della specie *versicolor* ed in verde quelli della *virginica*, si può notare che la classe *setosa* risulta perfettamente separabile dalle altre due, sia per quanto riguarda le dimensioni dei petali, sia per quelle dei sepali.



(a) *Petali*

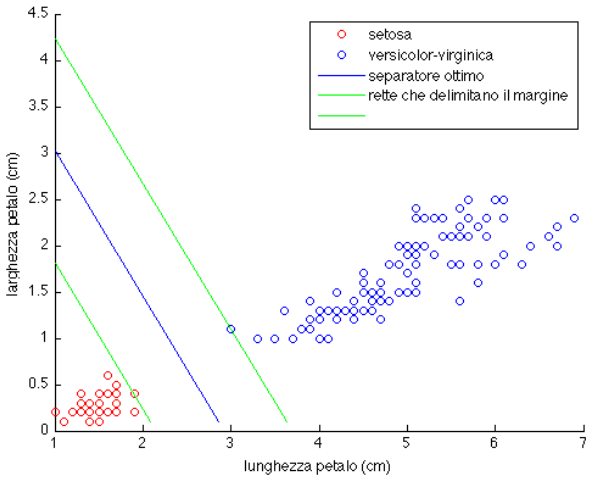


(b) *Sepali*

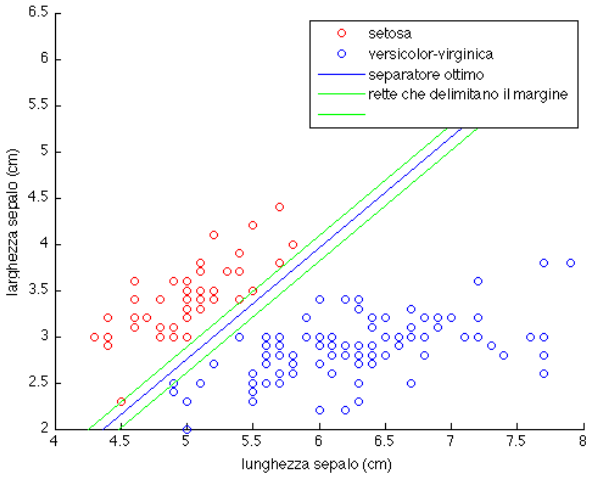
Figura 4.3: Training set degli iris di Fisher

È possibile quindi creare due macro-classi, una formata dagli iris della specie *setosa* (caratterizzata dal label 1) ed un'altra in cui vengono unite la *versi-*

color e la *virginica* (di label -1). Il procedimento per trovare il separatore ottimo tra le due classi è esattamente lo stesso dell'esempio precedente, come anche i vettori e le matrici H, f, A, b da utilizzare; dopo averli inseriti nella funzione *quadprog* ed aver trovato le intercette all'origine per le rette separatrici e per le rette che delimitano il margine massimo, si può procedere come prima ad una loro rappresentazione tramite la funzione *line*. I grafici ottenuti sono i seguenti:



(a) *Petali*



(b) *Sepali*

Figura 4.4: Separatori ottimi per gli iris di Fisher

A questo punto si passa alla fase di test, che permette di collocare un qualsiasi iris (anche non appartenente al training set) nella macro-classe corretta.

Graficamente, date le dimensioni del petalo o del sepalo, si può vedere subito in quale dei due semipiani rappresentati nelle Figure 4.4 sarà contenuto l'iris in questione e si potrà quindi determinare la sua specie di appartenenza. Procedendo analiticamente, invece, si può utilizzare la funzione di classificazione:

$$f(x) = sign(\mathbf{w} \cdot \mathbf{x} + b) \tag{4.3}$$

Se in questa si sostituiscono i parametri del separatore ottimo trovati grazie alla funzione *quadprog* e, al posto del vettore $\mathbf{x}$, le dimensioni del petalo o del sepalo dell'iris in oggetto, si ottiene come risultato un label ± 1. Se si ottiene 1 si potrà quindi affermare che l'iris appartiene alla specie *setosa*, altrimenti sarà sicuramente o della specie *versicolor* oppure di quella *virginica*.

4.3 Dati non linearmente separabili

Si considerino ora due classi non linearmente separabili (Figura 4.5)

```
R=[10 9 1;
   6 10 1;
   15 4 1;
   2 2 1]
 B=[-3 5 -1;
    -2 1 -1;
    8 2 -1;
    2 8 -1]
```

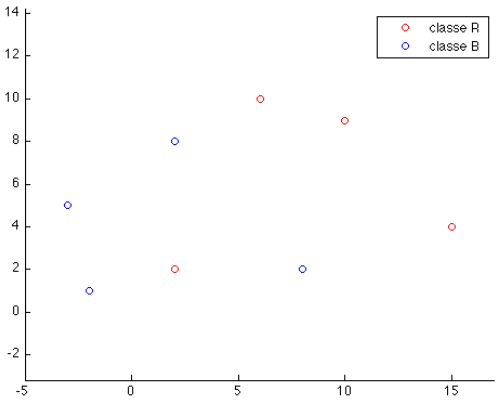


Figura 4.5: Due classi non linearmente separabili

L'analisi sarà un po' diversa dai casi precedenti, in quanto, nel definire matrici e vettori da inserire nella funzione *quadprog*, è necessario tener conto delle variabili slack e della costante C. Considerando il parametro $k = 1$, la funzione da minimizzare è:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\Big(\sum_i \xi_i\Big) \tag{4.4}$$

soggetta a:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad con \quad \xi_i \geq 0 \quad \forall i \tag{4.5}$$

Definito $\mathbf{z} = \begin{bmatrix} w_1 & w_2 & b & e_1 & \dots & e_m \end{bmatrix}$ il vettore delle incognite, dove gli e_i rappresentano gli errori corrispondenti ad ogni dato (m è il numero totale dei dati), il codice *Matlab* utilizzato per definire H, f, A, b è il seguente:

```
tot=size(R,1)+size(B,1)
H=zeros(3+tot)
H(1,1)=1
H(2,2)=1

C=2       %scelto a piacere

f=C*ones(3+tot,1)
f(1)=0
f(2)=0
f(3)=0

campioni=[R;
          B]
A=zeros(tot+tot,3+tot)
for i=1:tot
    A(i,1)=-campioni(i,1)*campioni(i,3)
    A(i,2)=-campioni(i,2)*campioni(i,3)
    A(i,3)=-campioni(i,3)
    A(i,3+i)=-1
end

for i=tot+1:2*tot
    A(i,i+3-tot)=-1
end

b=[-ones(tot,1);zeros(tot,1)]
```

Il vettore **z** ottenuto dall'applicazione della funzione *quadprog* è:

```
Z =
    0.2000
    0.3000
   -3.2000
         0
         0
         0
    3.2000
         0
         0
         0
    0.6000
```

Si nota che solo un dato ha errore superiore ad 1, ed è proprio il caso del dato mal classificato che possiamo osservare in Figura 4.6. Per il resto, si può notare che anche in questo caso la retta separatrice divide perfettamente la classe dei punti rossi da quella dei punti blu ed arrivati a questo punto sarà molto semplice passare alla fase di test per nuovi dati.

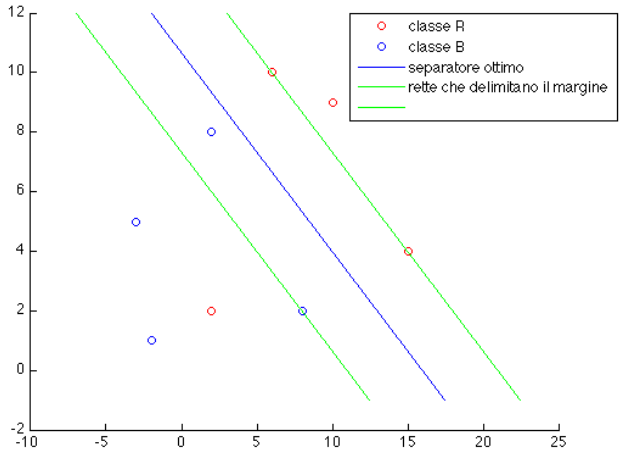


Figura 4.6: Separatore ottimo in presenza di errori

Conclusioni

Grazie alle applicazioni pratiche viste nel Capitolo 4 si è potuto confermare che, quando si ha a disposizione un metodo opportuno per minimizzare le funzioni, l'algoritmo delle Support Vector Machines è molto semplice da utilizzare. Con pochi passaggi, infatti, è stato possibile effettuare la fase di training ed ottenere i parametri necessari per la classificazione di qualsiasi dato appartenente ad una delle due classi in oggetto. In generale quindi le SVM forniscono un approccio completamente nuovo alla classificazione ed eliminano molti dei problemi tipici di altri algoritmi di apprendimento. I pregi di questi nuovi classificatori, infatti, sono molteplici:

- A differenza delle reti neurali che nel caso non lineare risultano complicate da addestrare, l'algoritmo delle SVM, con l'utilizzo degli spazi immagine e dei kernel, risulta essere molto efficiente e, in fase di ottimizzazione, non presenta problemi di minimi locali. Come visto in precedenza, infatti, il metodo di risoluzione si riconduce ad un problema di programmazione quadratica che permette sempre di trovare un minimo globale.

- I risultati che si ottengono sono indipendenti dall'algoritmo specifico utilizzato per l'ottimizzazione della funzione: se due utenti utilizzano le SVM per una classificazione, con gli stessi parametri e gli stessi dati, otterranno sempre la stessa soluzione, qualunque sia il metodo usato per risolvere il problema di minimizzazione.

- Le tecniche di ottimizzazione sono in continuo sviluppo e al giorno d'oggi si stanno affinando moltissimo, permettendo così alle SVM di diventare sempre più veloci nella classificazione di quantità ingenti di dati.

- L'algoritmo delle SVM è inoltre robusto al rumore e, grazie all'uso delle variabili slack, funziona bene anche nel caso in cui sono presenti dei dati anomali nel training set.

In questo lavoro sono stati analizzati solamente problemi di classificazione binaria, ma potrebbe capitare che le possibili classi di appartenenza di un dato siano più di due (come nel caso del riconoscimento delle cifre scritte a mano citate nel Capitolo 1). In queste situazioni è necessario estendere quanto detto finora al caso multiclasse. L'idea sta nel risolvere uno alla volta più problemi di classificazione binaria, separando ogni volta la classe in oggetto da tutte le altre: se le classi sono perfettamente separabili, per ogni dato in ingresso solo una di queste classificazioni darà risultato positivo e permetterà di assegnare il campione nella classe corretta.

Nonostante il divario tra intelligenza artificiale e intelligenza naturale rimanga consistente, è sorprendente vedere come le macchine oggi, grazie al perfezionamento delle tecniche di apprendimento automatico, riescano in qualche modo ad emulare le capacità della mente umana. Gli algoritmi delle SVM per la classificazione, che se estesi al caso multiclasse possono essere usati nei campi più disparati, ne sono proprio uno dei principali esempi.

Bibliografia

[1] C. Burges (1998), *A Tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery, vol 2: pp. 121–167, Kluwer Academic Publishers, Boston.

[2] M. Hearst (1998), Support Vector Machines, *IEEE Intelligent Systems*, vol. 13, no. 4: pp. 18-28.

[3] T. Hastie et al. (2009), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd edition: pp. 1-8, pp. 417-458, Springer.

[4] K.P. Bennett e C. Campbell (2000), *Support Vector Machines: Hype or Hallelujah?*, SIGKDD Explorations, vol. 2, no. 2 , pp. 1–13

[5] http://archive.ics.uci.edu/ml/datasets/Iris