UNIVERSITÀ DEGLI STUDI DI PALERMO

SCUOLA DELLE SCIENZE DI BASE E APPLICATE

DIPARTIMENTO DI MATEMATICA E INFORMATICA

Corso Di Laurea in Informatica

Classificazione Multiclasse con Support Vector Machine

TESI DI LAUREA DI

Federico Corrao

RELATORE

Ch.mo Prof. Giosuè Lo Bosco

ANNO ACCADEMICO 2013/2014

Indice

Sommario

Gli argomenti discussi nella presente tesi di laurea si collocano nel cosiddetto ambito dell'*apprendimento automatico* (*machine learning*) ed in particolare dell'*apprendimento supervisionato*, o *classificazione*.

Lo scopo ultimo dell'elaborato consiste nel presentare un metodo efficace di classificazione *multiclasse* basato su combinazione di classificatori binari. Il metodo mostrato risulterà essere un criterio di unificazione fra due schemi di riferimento che consentono di ottenere tale tipologia di classificatori k-ari: *one-versus-one* (o *one-against-one*) e *one-versus-all* (o *one-against-all*). Ci si propone quindi di fornire un criterio razionale per l'applicazione combinata dei suddetti paradigmi, così da ottenere un nuovo approccio alla classificazione le cui *performance* attese siano generalmente maggiori.

Verrà focalizzata l'attenzione sulle *SVM* (*Support Vector Machine*, *macchine a vettori di supporto*): una consolidata famiglia di classificatori che, avendo una formulazione naturale per problemi di separazione binari ($k = 2$ classi), si prestano bene ad una estensione al caso multiclasse ($k > 2$ classi). In virtù della loro versatilità — e delle buone prestazioni che raggiungono in applicazioni pratiche — le *SVM* sono ad oggi fra i metodi allo *stato dell'arte* più utilizzati.

L'elaborato non può quindi che iniziare con una rassegna introduttiva dei concetti fondamentali dell'apprendimento automatico, per poi discutere delle *SVM* e dei due sopracitati schemi di classificazione combinatori. Verrà quindi presentato il criterio di unificazione, motivando le scelte adottate nel tentativo di costruire un classificatore più robusto.

Infine, di tutti i metodi discussi saranno mostrati e comparati i relativi risultati rispetto ad una loro applicazione su *dataset* "reali". Si evidenzierà, in conclusione, che il criterio presentato consente effettivamente di ottenere — in tali casi particolari — un incremento di prestazioni durante la fase di test.

Nel **Capitolo 1** verrà introdotto l'apprendimento automatico, descrivendone brevemente i concetti fondamentali, presentandone una serie di applicazioni ed evidenziando l'importanza che esso ricopre all'interno del panorama tecnologico dell'informatica odierna. Si darà una breve descrizione della tipologia di *learning* inerente agli argomenti affrontati — l'apprendimento *supervisionato* — esponendone scopi e problematiche connesse.

Nel **Capitolo 2** verranno presentate le *Support Vector Machine*, esaminando il caso in cui i dati di *training* siano *linearmente separabili* e quello in cui non lo siano (il cosiddetto caso *soft-margin*). Saranno in introdotte le *funzioni kernel* e verrà mostrato come il loro impiego possa condurre ad una generalizzazione delle *SVM* al caso *non-lineare* — il *kernel trick*. Delle *SVM* verranno presentati vantaggi e svantaggi, nonché brevi considerazioni sulla loro complessità di calcolo e implementazione.

Nel **Capitolo 3** saranno quindi proposti i due metodi più utilizzati per la generalizzazione classificatori binari al caso multiclasse: *one-against-one* e *one-against-all*. Anche in questo caso verranno esposti *pro* e *contro*, considerazioni sulle rispettive complessità di calcolo, ed osservazioni sulla loro applicazione alle *SVM*. Verrà presentato inoltre il criterio di unificazione per tali schemi, evidenziando le intuizioni che hanno condotto alla sua produzione.

Nel **Capitolo 4**, infine, verranno presentati i sopracitati *dataset* e la natura dei dati in essi contenuti. Saranno quindi mostrati i risultati di classificazione ottenuti su di essi mediante l'applicazione dei metodi *k-NN*, *SVM one-vs-one*, *SVM one-vs-all* e del *metodo ibrido*. Alla luce di tali risultati, si arriverà alla conclusione che il metodo ibrido è — in alcune circostanze — effettivamente in grado di raggiungere *performance* migliori rispetto ai due metodi combinatori applicati separatamente.

La stesura della presente tesi di laurea è stata costantemente accompagnata dalla produzione di codice MATLAB (*r2014a Student Version*[1]) al fine di mettere in pratica quanto discusso. Nel corso dei vari capitoli sono presenti — ove ritenuto opportuno — figure e grafici.

[1] http://www.mathworks.it/academia/student_version/

Capitolo 1

Introduzione

1.1 Apprendimento automatico

L'**apprendimento automatico** (noto in letteratura come *machine learning*) è, come suggerito dal termine, un'area fondamentale dell'intelligenza artificiale il cui scopo consiste nello studio e implementazione di sistemi in grado di *apprendere* da insiemi di dati.

Per sistema si intende un algoritmo che, ricevuti dei dati in input (*osservazioni*), elabora gli stessi al fine di produrre conoscenza. Tale conoscenza è quindi inizialmente implicita, nascosta nelle relazioni fra i dati di interesse. Il compito del sistema è di analizzare questi ultimi, al fine di esplicitare informazione utile non nota a priori.

La conoscenza sintetizzata da questi algoritmi è in genere utilizzata da altri sistemi nel prendere *decisioni* autonomamente, durante l'esecuzione di specifiche attività (più o meno complesse). Ciò spiega la grande diffusione delle tecniche di *machine learning* in applicazioni pratiche, dove è cruciale la possibilità di effettuare scelte intelligenti senza la supervisione di un essere umano.

Di seguito una lista di applicazioni ben note delle suddette tecniche:

- **Individuazione di facce in immagini e video** (*face detection*); e.g. in sistemi real-time per il riconoscimento di un interlocutore posto di fronte un sensore (funzionalità introdotta di recente anche nelle moderne fotocamere digitali e negli smartphone) [4]

- **Riconoscimento di caratteri** (*OCR*, *Optical Character Recognition*); e.g. estrazione di informazioni da documenti cartacei; book scanning; conversione real-time di caratteri manoscritti (funzionalità sovente presente in dispositivi touchscreen); tecnologie assistive

- **Riconoscimento vocale** (*speech recognition*); e.g. in sistemi di scrittura automatica sotto dettatura (*dictation system*); interfacce uomo-macchina basate su comandi vocali; tecnologie assistive

- **Classificazione di testi** (*text categorization*); e.g. in *topics* per sistemi di *information retrieval* [4] (memorizzazione e recupero di documenti) e motori di ricerca; riconoscimento delle e-mail indesiderate nelle caselle di posta elettronica (*spam filtering*)

Altri campi di ricerca e sviluppo che fanno uso delle tecnologie offerte dal *machine learning* sono: *data mining* (estrazione di informazione e conoscenza utile da grandi quantità di dati) per *marketing* (ovvero in ricerche di mercato) e *business intelligence* (raccolta e analisi di dati in ambito aziendale); bioinformatica, medicina (e.g. diagnosi automatizzata), videoludica, robotica, sicurezza informatica e molto altro. È quindi naturale il manifestarsi di un interesse via via crescente per tali tecniche, che negli ultimi anni hanno avuto una diffusione capillare in qualsiasi contesto che preveda la presenza di dati e dispositivi di calcolo.

Una nota definizione di "apprendimento" è data da T. Mitchell in [1]:

> Un programma (algoritmo, sistema) che svolge un dato insieme di attività T apprende da una certa esperienza E se le sue prestazioni (misurate da una certa metrica P) nel compiere un'attività in T migliorano grazie all'esperienza E.

dove l'esperienza passata E è rappresentata sotto forma di dati in input dai quali apprendere.

I principali metodi di apprendimento automatico si dividono in due macro-categorie, a seconda della natura dei problemi che affrontano (ovvero dalla tipologia dei dati in esame): **apprendimento supervisionato** e **non supervisionato**.

L'apprendimento non supervisionato si rivolge a problemi di natura diversa da quelli affrontati nei prossimi capitoli; è quindi da considerarsi al di fuori degli scopi di questa tesi di laurea. Per una trattazione didattica dell'argomento si veda [2].

1.2 Apprendimento supervisionato

L'apprendimento **supervisionato** mira ad istruire un sistema cosicché, al verificarsi di un evento futuro (i.e. la ricezione di un'*osservazione* in input), esso sia in grado di prendere

una decisione in risposta all'evento — inferendo quest'ultima in base alla valutazione di una serie di *esempi*, passati o ideali.

Tale insieme di esempi iniziali è detto *training set* ("insieme di addestramento"). Un algoritmo di apprendimento supervisionato genera quindi, a partire dal *training set*, una *funzione di decisione* che, tenendo conto degli esempi già noti, è in grado di fornire (in output) una decisione per ogni nuovo evento (in input).

Una funzione di questo tipo è anche detta **classificatore**, e il problema in questione è detto **classificazione**. Per comprendere i motivi dietro la scelta di questa nomenclatura, si consideri una definizione più formale del problema.

Sia T un *training set* di n elementi, composto da coppie ordinate:

$$T = \{(\boldsymbol{x}_i, y_i);\ i = 1, 2 \ldots n;\ y_i \in C\} \tag{1.1}$$

C è l'insieme delle *etichette di classe* (*class labels*). La coppia $(\boldsymbol{x}_i, y_i)$ indica che l'esempio $\boldsymbol{x}_i$ (la cui natura può essere varia, e.g. $\boldsymbol{x}_i \in \mathbb{R}^d$) appartiene alla *classe* y_i.

Sia quindi f funzione di decisione ottenuta (con un qualsiasi metodo di apprendimento) a partire dal *training set* T, che associa ad ogni osservazione $\boldsymbol{x}$ un'etichetta di classe $y \in C$. Si dice che f *classifica* $\boldsymbol{x}$ con etichetta y: la scelta di una decisione da prendere in risposta all'evento rappresentato dall'input $\boldsymbol{x}$ è una *predizione* del valore di y da parte di f.

Tale predizione è una valutazione di tipo qualitativo, dal momento che classificare l'osservazione x equivale ad associare x ad una categoria di dati, in qualche modo simili fra loro. Il problema della classificazione è perciò, intuitivamente, quello della "separazione" fra dati di diverso tipo. La predizione dell'etichetta per x sarà in genere tanto più precisa quanto lo sarà la facilità con cui si distinguono dati appartenenti a classi diverse. Se il classificatore non riesce ad assegnare l'etichetta corretta per l'osservazione $\boldsymbol{x}$ (cioè $f(\boldsymbol{x}) = y_{\text{predetta}} \neq y_{\text{reale}}$), si dice che essa è stata *misclassificata*.

Si indica con *test set* un insieme di osservazioni future, disgiunte dal *training set*, sulle quali applicare la funzione di decisione — ovvero per le quali si è interessati a predire l'etichetta di classe. Classificatori (e, per estensione, metodi di apprendimento) diversi vengono spesso comparati tramite parametri indicatori delle rispettive *performance* ottenute su un *test set*. Tali parametri sono generalmente calcolati ricorrendo ad una *matrice di confusione*, contenente informazioni sulle osservazioni misclassificate in fase di test (e.g *accuracy*).

È evidente che la scelta del *training set* per l'addestramento di un metodo di classificazione influenza la funzione di decisione f trovata, quindi le sue performance. Si discuteranno questi aspetti in un contesto pratico nel **Capitolo 4**, quando verranno confrontate le prestazioni di diversi metodi di classificazione multiclasse basati su SVM — e non solo.

Per adesso, ci si limiterà a dire che la fase di addestramento di un classificatore $f(\boldsymbol{x}; \alpha)$ può essere vista come la ricerca di un set di parametri α in modo da massimizzare le prestazioni di f, fissato il *training set* [3]. I motivi di questa considerazione saranno evidenti nel corso del **Capitolo 2**, dove mostreremo che l'addestramento di una Support Vector Machine (e quindi la ricerca di f) consiste sostanzialmente nella risoluzione di un problema di ottimizzazione che fornisca parametri α tali da soddisfare un criterio di "massimizzazione della separazione" fra due classi.

Come ultima considerazione di carattere generale sull'argomento, è necessario far presente che tutti i metodi di apprendimento supervisionato utilizzano una *assunzione* fondamentale: suppongono infatti che gli esempi del *training set* e le osservazioni da classificare provengano dalla stessa distribuzione di probabilità. Intuitivamente, affinché f riesca a classificare bene il *test set* (per un dato problema) occorre che sia addestrata con esempi sufficientemente *rappresentativi* del problema in esame — quindi delle sue classi [1]. Questa assunzione è spesso violata in una certa misura, con conseguente perdita di prestazioni da parte del classificatore [2].

Capitolo 2

Support Vector Machines

Verranno ora approfondite le tematiche brevemente introdotte nel **Capitolo 1** riguardo l'apprendimento supervisionato, focalizzando l'attenzione sulle *SVM — Support Vector Machine*, *macchine a vettori di supporto*.

La fase di apprendimento di tale tipologia di classificatori è basata su considerazioni di tipo geometrico, ed in particolare sul concetto di *iperpiano di separazione*. Verrà dapprima analizzato il caso banale in cui i dati di *training* siano linearmente separabili, per poi generalizzare i concetti presentati al caso *soft-margin*. Saranno infine introdotte le funzioni *kernel*, ponendo l'attenzione sul modo in cui quest'ultime si integrano nel formalismo delle *SVM*.

2.1 Dati linearmente separabili

Interpretazione geometrica Si introduce la trattazione delle *SVM* nel caso più semplice da illustrare: si supponga inizialmente che i dati del *training set* da cui ottenere la funzione di decisione ricercata siano *linearmente separabili* [5].

Sia T un *training set* — come definito nel **Capitolo 1** — costituito da n coppie ordinate:

$$T = \left\{(\boldsymbol{x}_i, y_i);\ i = 1, 2\ ...\ n;\ \boldsymbol{x}_i \in \mathbb{R}^d;\ y_i \in \{-1, +1\}\right\}. \tag{2.1}$$

La prima componente della i-esima coppia è l'osservazione $\boldsymbol{x}_i$, rappresentata come un *punto* in uno spazio continuo d-dimensionale ($X \in \mathbb{R}^d$). Ogni osservazione consta quindi di d componenti, o *feature*. La seconda componente della i-esima coppia rappresenta l'etichetta di classe y_i — nota a priori — associata alla osservazione $\boldsymbol{x_i}$.

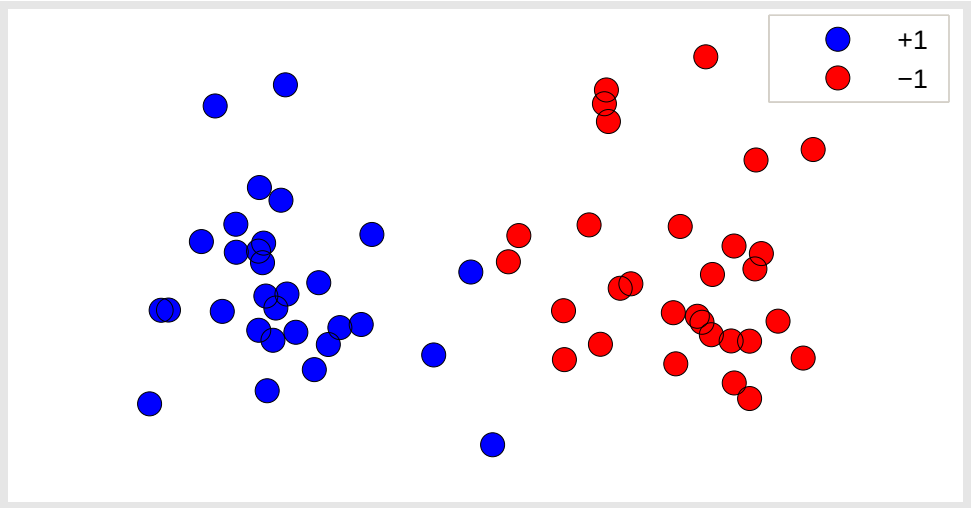

Figura A: *Due classi di punti positivi (T_+) e negativi (T_-) in $\mathbb{R}^2$ (scatter plot)*

Poiché $C = \{-1, +1\}$ risulta evidente come si stia trattando un problema si classificazione *binario*, ovvero con $k = |C| = 2$ classi. Denotiamo allora con

$$T_+ = \left\{ \boldsymbol{x}_+ \in \mathbb{R}^d \mid (\boldsymbol{x}_+, +1) \in T \right\} \tag{2.2}$$

$$T_- = \left\{ \boldsymbol{x}_- \in \mathbb{R}^d \mid (\boldsymbol{x}_-, -1) \in T \right\} \tag{2.3}$$

rispettivamente la *classe dei positivi* e la *classe dei negativi*.

L'idea fondamentale alla base della classificazione mediante *SVM* è quella di separare geometricamente le due classi (e per estensione i rispettivi punti) ricorrendo a un *iperpiano di separazione*. Ricordiamo che un *iperpiano* in $\mathbb{R}^d$ è così definito:

$$H = \left\{ x \in \mathbb{R}^d \,|\, w \cdot x + b = 0;\ w \in \mathbb{R}^d \right\} \tag{2.4}$$

ovvero come un insieme di punti x che soddisfano una equazione lineare caratterizzata dai coefficienti w e b, dove: w è un vettore ortogonale (*normale*) all'iperpiano H (e indica la sua direzione), b è il termine noto (spesso indicato in letteratura come *offset* o *bias*).

Ricordiamo inoltre che un iperpiano è una generalizzazione della nozione intuitiva di *piano* su spazi d-dimensionali con $d \neq 2$, $d \neq 3$; i.e. per $d = 1$ un iperpiano è rappresentato da un punto in un dato spazio uni-dimensionale, per $d = 2$ è una retta in uno spazio bi-dimensionale, per $d = 3$ è un piano (nella comune accezione del termine) in uno spazio tri-dimensionale, e così via.

Un iperpiano possiede la desiderabile proprietà di dividere i rimanenti punti dello spazio in due *semispazi* (semirette, semipiani, etc.) disgiunti. La strategia adoperata consiste dunque

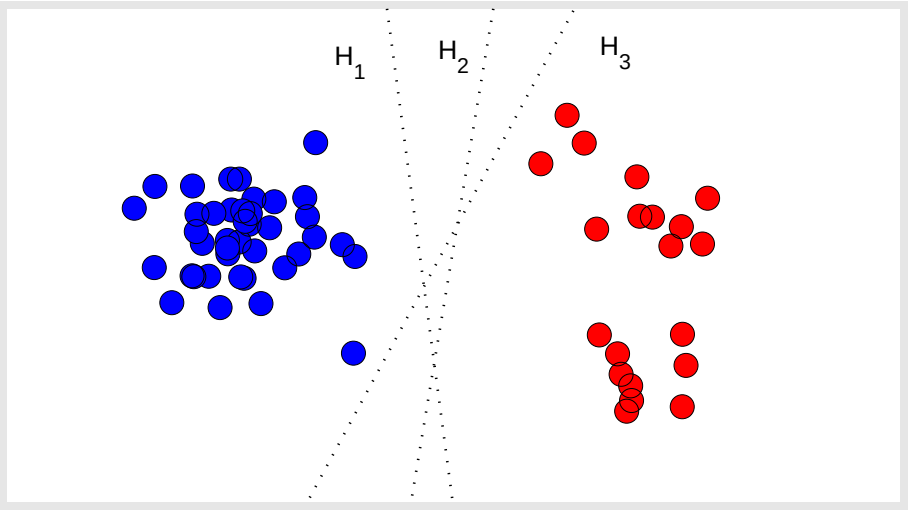

Figura B: *Tre iperpiani di separazione in $\mathbb{R}^2$ (rette), con dati linearmente separabili*

nell'individuare un iperpiano adatto alla costruzione di una funzione di decisione, al fine di classificare osservazioni successive. In prima istanza, si richiede l'*esistenza* di un iperpiano in grado di bipartire lo spazio $\mathbb{R}^d$ in *semispazio positivo* (contenente *solamente* i punti di T_+) e *semispazio negativo* (contenente *solamente* i punti di T_-).

Un tale iperpiano è detto *di separazione*. Fissato quindi un *dataset* T avente rispettive classi T_+ e T_-, se esiste un iperpiano di separazione per tali insiemi di punti si dice che essi sono *linearmente separabili*. In accordo a quanto mostrato nella sezione successiva, si chiarisce sin d'ora che la separabilità lineare non è strettamente necessaria per la costruzione di classificatori basati su *SVM*: essa rappresenta infatti una condizione molto restrittiva, difficilmente soddisfatta in *dataset* provenienti da applicazioni reali.

Ai fini della trattazione, si assumerà al momento che tale ipotesi irrealistica sia verificata.

Posto che gli insiemi di punti T_+ e T_- siano linearmente separabili, ci si propone si trovare per essi un iperpiano di separazione. Risulta evidente l'esistenza di un numero infinito di tali iperpiani: un criterio di scelta ragionevole prevede di selezionare il particolare iperpiano di separazione *a massimo margine*.

Sia H un qualunque iperpiano di separazione, d_+ (d_-) la distanza euclidea fra H e il più vicino positivo $\boldsymbol{x}_+ \in T_+$ (negativo $\boldsymbol{x}_- \in T_-$), banalmente calcolata ricorrendo alla formula

$$d_+ = d(\boldsymbol{x}_+, H) = \frac{|w \cdot \boldsymbol{x}_+ + b|}{||w||} \tag{2.5}$$

dove $||w||$ è la norma euclidea $\sqrt{w_1^2 + ... + w_d^2}$ (analogo per $\boldsymbol{x}_-$).

Si definisce il *margine* di un iperpiano di separazione H la quantità $d_+ + d_-$.

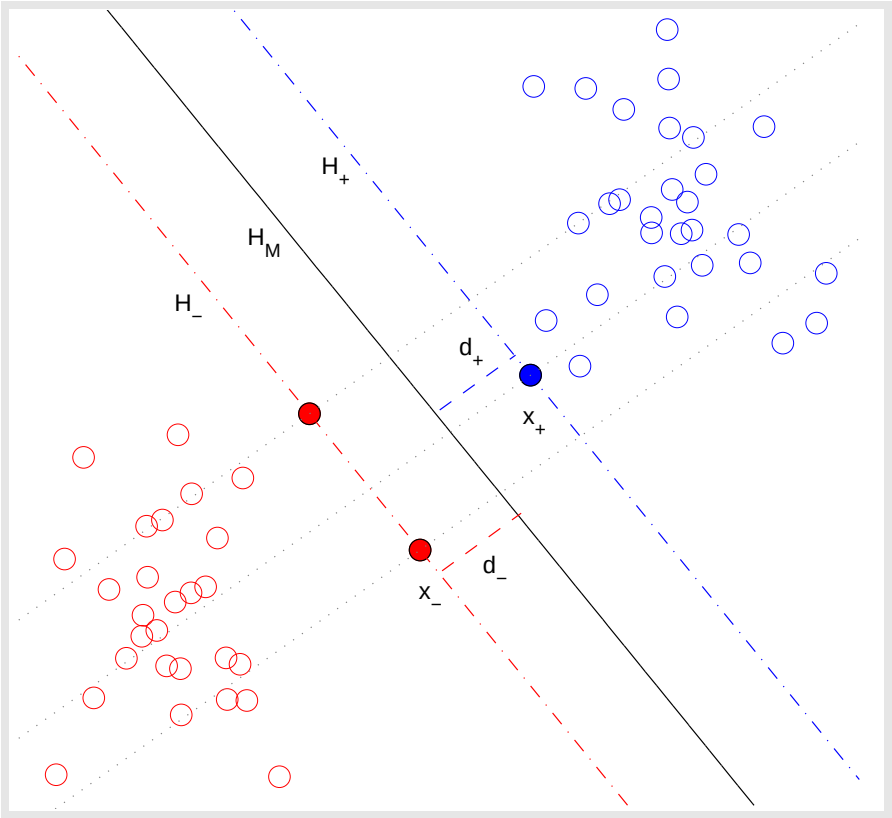

Figura C: *Iperpiano di separazione a massimo margine* H_M *in* $\mathbb{R}^2$

Il problema si traduce quindi nella ricerca di valori dei coefficienti w e b tali che — fra tutte le possibili scelte di w e b — essi permettano di ottenere l'iperpiano H_M di separazione con margine $d_+ + d_-$ massimo.

$$H_M : w \cdot \boldsymbol{x} + b = 0 \; . \tag{2.6}$$

Si supponga adesso di scegliere opportunamente un fattore di scala per w e b in modo che risultino vere, per l'ipotesi di separabilità lineare, le seguenti asserzioni:

$$w \cdot \boldsymbol{x_i} + b \geq +1 \;\; \text{per} \;\; y_i = +1, \;\; \forall \, i = 1, 2 \; ... \; n \tag{2.7}$$

$$w \cdot \boldsymbol{x_i} + b \leq -1 \;\; \text{per} \;\; y_i = -1, \;\; \forall \, i = 1, 2 \; ... \; n \tag{2.8}$$

(una per ogni elemento in T), riscrivibili in forma compatta come:

$$y_i(w \cdot \boldsymbol{x_i} + b) - 1 \geq 0, \;\; \forall \, i = 1, 2 \; ... \; n \; . \tag{2.9}$$

Allo scopo di inquadrare visivamente il significato geometrico del problema, si considerino inoltre due iperpiani ausiliari H_+ e H_-, così definiti:

$$H_+ : w \cdot \boldsymbol{x} + b = +1 \tag{2.10}$$

$$H_- : w \cdot \boldsymbol{x} + b = -1 \ . \tag{2.11}$$

È evidente che H_+ e H_- sono paralleli (poiché hanno lo stesso w) e differiscono per il termine noto. Inoltre, entrambi risultano paralleli all'iperpiano ricercato H_M.

L'insieme di vincoli (2.7) indica che gli esempi positivi ($\boldsymbol{x} \in T_+$), come si illustra in Figura C, sono posizionati nell'area "al di sopra" dell'iperpiano H_+, o eventualmente vi appartengono. Ciò equivale a dire che *nessun* positivo oltrepassa H_+ dal lato opposto. La stessa considerazione si applica — a segni inversi — con la (1.8) agli esempi negativi. In definitiva, tutte le condizioni (2.9) impongono che nessuno dei punti di *training* sia ricada nell'area compresa fra H_+ e H_-. I punti $\boldsymbol{x}_+$ e $\boldsymbol{x}_-$ (quelli più vicini, appartenenti a classi distinte) giacciono proprio su H_+ e H_-, rispettivamente. È da notare inoltre che l'iperpiano ricercato H_M sia posizionato esattamente "a metà strada" fra H_+ e H_-.

Si consideri adesso un qualsiasi punto $\boldsymbol{x_h} \in H_M$; dalle riflessioni appena fatte segue che il margine è ricavabile come la misura della distanza fra gli iperpiani H_+ e H_-. La distanza d_+ tra il punto $\boldsymbol{x_h}$ e l'iperpiano H_+ è:

$$d_+ = \frac{|\boldsymbol{x_h} \cdot w + (b-1)|}{||w||} \tag{2.12}$$

analogamente, la distanza d_- tra il punto $\boldsymbol{x_h}$ e l'iperpiano H_- è:

$$d_- = \frac{|\boldsymbol{x_h} \cdot w + (b+1)|}{||w||} \tag{2.13}$$

Ma poiché $\boldsymbol{x_h} \in H$, vale $\boldsymbol{x_h} \cdot w + b = 0$.
In definitiva, un'espressione per il margine è data da

$$d_+ + d_- = \frac{|-1|}{||w||} + \frac{|1|}{||w||} = \frac{2}{||w||} \ . \tag{2.14}$$

L'addestramento di una *SVM* coincide in pratica con la risoluzione del seguente problema di ottimizzazione: massimizzare la funzione obiettivo $2/||w||$ sotto i vincoli dati da (2.9), nelle variabili w e b. Ciò equivale a minimizzare $||w||/2$ sotto le stesse condizioni. Per semplicità verrà considerata invece la funzione obiettivo $||w||^2/2$, con

$$||w||^2 = \left(\sqrt{w_1^2 + ... + w_d^2}\right)^2 = w_1^2 + ... + w_d^2 = w \cdot w \tag{2.15}$$

la cui soluzione — sempre rispetto ai vincoli (2.9) — è la medesima. In questo modo viene eliminato lo svantaggio computazionale derivante dal calcolo di una radice quadrata. Il fattore di $1/2$ è similmente mantenuto per convenienza matematica (durante i calcoli).

Più formalmente, ci si pone in definitiva il seguente problema:

$$\begin{aligned} &\arg\min_{w,b} \frac{1}{2}||w||^2 \\ &y_i(w \cdot \boldsymbol{x_i} + b) - 1 \geq 0, \ \ \forall\, i = 1, 2 \ ... \ n \ . \end{aligned} \tag{2.16}$$

I punti $\boldsymbol{x_s}$ che appartengono a H_+ o a H_+ sono detti **vettori di supporto** (*support vectors*, da qui il nome del modello); una loro eventuale rimozione dal *training set* cambia la soluzione del problema di ottimizzazione. Secondo le definizioni date in precedenza, $\boldsymbol{x}_+$ e $\boldsymbol{x}_-$ sono *dei* vettori di supporto: è infatti ammissibile l'esistenza di più punti giacenti sugli iperpiani H_+ e H_+; segue che il numero di vettori di supporto può essere $s > 2$.

Rimuovere dal *training set* un qualsiasi punto — sia esso positivo o negativo — che non fa parte dei vettori di supporto non influenza invece la soluzione trovata (ovvero i valori desiderati di w e b).

Ottimizzazione Si accennerà ora alla formulazione del suddetto problema di ottimizzazione; la traccia della seguente discussione è ispirata a [3].

Si consideri la forma Lagrangiana del problema — ottenuta applicando quindi il metodo dei moltiplicatori di Lagrange alla funzione obiettivo $||w||^2/2$ — la quale permette di semplificare la notazione di esso, operando con vincoli meglio trattabili.

Siano n moltiplicatori di Lagrange positivi $\alpha_i \in \mathbb{R}, \ i = 1, 2 \ ... \ n$; uno per ciascuna delle n condizioni in (2.9) (ovvero uno per ogni elemento del *training set*). Ricordiamo che, per vincoli della forma $c_i \geq 0$, il metodo prevede di sottrarre alla funzione obiettivo le espressioni dei vincoli moltiplicate ai coefficienti positivi $\alpha_i \geq 0$. Ciò conduce alla forma:

$$L_P = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} [\alpha_i y_i (w \cdot \boldsymbol{x_i} + b) - 1] \tag{2.17}$$

dove il pedice P in L_P indica che il problema è espresso in forma primale.

Il problema di ottimizzazione è quindi stato trasformato nella equivalente formulazione: minimizzare L_P rispetto a w e b, imponendo che $\alpha_i \geq 0$ e richiedendo che le i derivate parziali $\partial L_p / \partial \alpha_i$ si annullino — si denoti questo insieme di vincoli con C_1. Esso risulta essere un problema di programmazione quadratica convesso, visto che (a) la funzione obiettivo L_P è convessa (b) l'insieme dei punti che soddisfano i vincoli (i.e. l'insieme ammissibile) è un insieme convesso (poiché ogni vincolo lineare della (2.9) descrive un insieme convesso e l'intersezione di n insiemi convessi è ancora un insieme convesso).

Ciò permette di passare ad una formulazione duale del problema (nota come "duale di Wolfe"): *massimizzare* L_P (sempre rispetto a w e b) imponendo che $\alpha_i \geq 0$ e richiedendo stavolta che siano le derivate parziali $\partial L_P/\partial w$ e $\partial L_P/\partial b$ ad annullarsi — si denoti questo insieme di vincoli C_2. La formulazione primale e quella duale sono equivalenti: la minimizzazione di L_P rispetto ai vincoli C_1 e la massimizzazione di L_P rispetto ai vincoli C_2 conducono alla stessa soluzione (ovvero agli stessi valori di w e b).

Dai vincoli sulle derivate parziali in C_2

$$\frac{\partial L_P}{\partial w} = 0 \tag{2.18}$$

$$\frac{\partial L_P}{\partial b} = 0 \tag{2.19}$$

derivando L_P nei due casi e risolvendo, si trova rispettivamente:

$$w = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x_i} \tag{2.20}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.21}$$

e sostituendo nella (2.17) si ottiene da L_P una espressione per L_D:

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{\substack{i=1 \\ j=1}}^{N} \alpha_i \alpha_j y_i y_j \ \boldsymbol{x_i} \cdot \boldsymbol{x_j} \tag{2.22}$$

dove il pedice D in L_D indica che il problema è espresso nella sua forma duale.

Come già affermato, L_P e L_D corrispondono a problemi di ottimizzazione differenti (sia in termini di funzione obiettivo sia di vincoli) ma che conducono alla stessa soluzione. È da notare in particolare come nella (2.22) non figuri più la dipendenza dalle variabili w e b, quanto piuttosto dai moltiplicatori α_i. La fase di *training* di una *SVM* è quindi sintetizzabile, in ultima istanza, come la massimizzazione di L_D rispetto ai coefficienti α_i.

Sebbene sia presente un moltiplicatore α_i per ciascun punto di training $\boldsymbol{x_i}$, generalmente la maggior parte di essi sarà nullo: vale infatti che $\alpha_i > 0$ (strettamente positivo) solo per i vettori di supporto (che indicheremo d'ora in poi con $\boldsymbol{x_s} \in SV$). Ciò è coerente con quanto affermato dalla espressione (2.20) — la quale fornisce un modo diretto per ricavare il valore di w ricercato — dove è evidente come i punti $\boldsymbol{x} \notin SV$ non incidano nel calcolo della soluzione trovata. I vettori di supporto sono dunque gli elementi critici del *training set*, dalla cui entità dipende direttamente il calcolo della funzione di decisione f.

Nonostante sia stata ricavata una espressione immediata per il calcolo di w, l'iperpiano di separazione a massimo margine ricercato non è ancora univocamente determinato — non è ancora noto infatti il valore di b. Di seguito si fornisce una espressione per ricavare b a partire dalle condizioni di Karush-Kuhn-Tucker.

Condizioni di Karush-Kuhn-Tucker Tali condizioni rivestono un ruolo fondamentale sia nella teoria che nella pratica della risoluzione di problemi di ottimizzazione vincolata. Essendo una loro trattazione teorica ben oltre gli scopi di questo elaborato, ne verrà fornita una presentazione concisa e non esaustiva.

Riguardo il particolare caso affrontato nei paragrafi precedenti, per il problema di ottimizzazione espresso in forma primale L_P valgono le seguenti condizioni:

$$\frac{\partial}{\partial w_j} L_P = w_j - \sum_{i=1}^{n} \alpha_i y_i x_{ij} = 0 \quad \forall\, j = 1, 2, \ \ldots\ d \tag{2.23}$$

$$\frac{\partial}{\partial b} L_P = -\sum_{i=1}^{n} \alpha_i \boldsymbol{x_i} = 0 \tag{2.24}$$

$$y_i(w \cdot \boldsymbol{x_i} + b) - 1 \geq 0 \quad \forall\, i = 1, 2, \ \ldots\ n \tag{2.25}$$

$$\alpha_i \geq 0 \quad \forall\, i = 1, 2, \ \ldots\ n \tag{2.26}$$

$$\alpha_i(y_i(w \cdot \boldsymbol{x_i} + b) - 1) = 0 \quad \forall\, i = 1, 2, \ \ldots\ n\ . \tag{2.27}$$

che risultano essere soddisfatte dalla soluzione del problema di ottimizzazione. Inoltre, in presenza di problemi convessi — ed è questo il caso — tali condizioni risultano essere *necessarie e sufficienti* affinché w, b e gli α_i siano soluzioni del problema. Si utilizzi, per lo scopo preposto, il vincolo (2.27) detto "condizione di complementarità". Esso consente di ricavare di b a partire dalla conoscenza di w e di almeno un vettore di supporto. Riscrivendo b in forma esplicita si ha:

$$b = \frac{1}{y_i} - w \cdot \boldsymbol{x_s} \tag{2.28}$$

comunque scelto $\boldsymbol{x_s} \in SV$ (ovvero un qualsiasi vettore di supporto). È comunque buona pratica (adottata peraltro in sistemi automatici che implementano il *training SVM*) calcolare un b_i per ogni vettore di supporto ed assumere come valore definitivo la media dei b_i ottenuti, in modo da ovviare ad inesattezze derivanti da approssimazioni intermedie nel calcolo di b.

Test Una volta ricavati sia w che b, non rimane che esplicitare la funzione di decisione f da utilizzare per predire le etichette di osservazioni successive in input. f è quindi fortemente

legata all'iperpiano di separazione a massimo margine identificato dai coefficienti w e b. L'espressione finale per f è data da:

$$f(\boldsymbol{x}; w, b) = sign(w \cdot \boldsymbol{x} + b) \tag{2.29}$$

dove banalmente si classifica $\boldsymbol{x}$ a seconda della sua posizione rispetto all'iperpiano H_M.

Nel corso del **Capitolo 3** verrà mostrato come alcune tecniche di estensione di classificatori binari (quindi *SVM*) al caso multiclasse richiedano la conoscenza di informazioni ulteriori rispetto alla sola etichetta predetta (i.e. la distanza dell'osservazione $\boldsymbol{x}$ dall'iperpiano H_M, utilizzata come *score*, "punteggio").

2.2 Dati non linearmente separabili

Interpretazione geometrica Ci si ponga adesso nel caso in cui i punti in $T_+ \cup T_-$ *non* siano linearmente separabili, ovvero non esista alcun iperpiano H di separazione tale che i semispazi da esso individuati contengano esclusivamente punti di una sola classe. Questo è ovviamente lo scenario più comune nei casi in cui si abbia a che fare con *dataset* provenienti da applicazioni reali (e non meramente didattici).

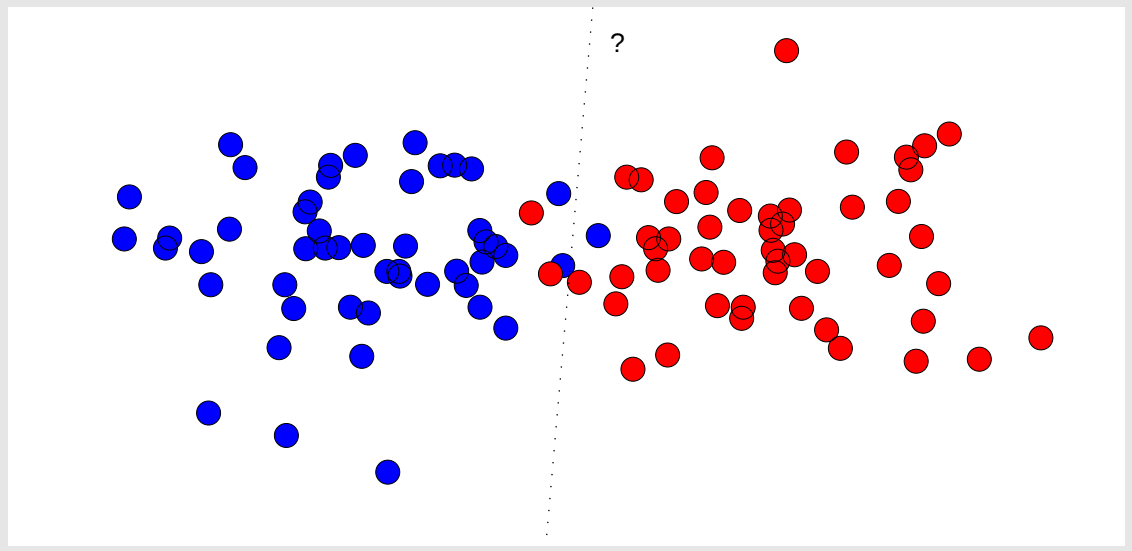

Figura D: *Due classi di punti non linearmente separabili in H_M in $\mathbb{R}^2$*

Sotto queste ipotesi, non valgono le considerazioni fatte nella sezione precedente per la ricerca di una soluzione al problema di ottimizzazione quadratica, visto che esse dipendevano dall'insieme di vincoli (2.9) — i quali non sono più verificati. Il meccanismo, in questo caso, consiste nel riformulare il problema originario così da consentire la presenza di "errori" in fase di *training*, ovvero di quei punti la cui presenza determina la non-separabilità delle

classi. Un classificatore *SVM* addestrato in questo modo cercherà comunque di ottenere la separazione più "pulita" possibile, cercando ancora una volta di massimizzare la separazione fra T_+ e T_-.

Al fine di ottenere un tale classificatore, si introducano n variabili ausiliarie di *slack* positive $\xi_i \geq 0; i = 1, 2 \ldots n$ (una per ogni elemento di *training*), in modo da riscrivere i vincoli (2.9) nella nuova forma:

$$w \cdot \boldsymbol{x_i} + b \geq +1 - \xi_i \quad \text{per} \quad y_i = +1, \quad \forall\, i = 1, 2 \ldots n \tag{2.30}$$

$$w \cdot \boldsymbol{x_i} + b \leq -1 + \xi_i \quad \text{per} \quad y_i = -1, \quad \forall\, i = 1, 2 \ldots n\,. \tag{2.31}$$

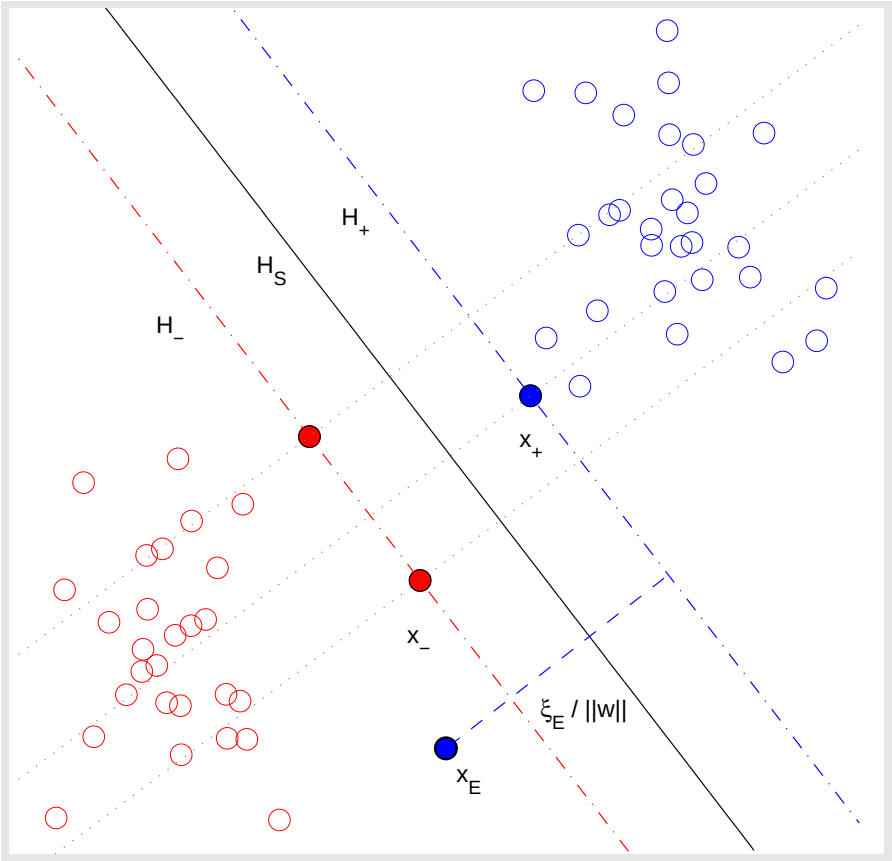

Figura E: $\boldsymbol{x_E}$ *è un punto "in errore" poiché è situato oltre l'iperpiano* H_S

Si supponga di aver trovato un iperpiano H_S per tale problema di separazione. Si osservi che — come indicato in figura B — affinché si verifichi un errore per $\boldsymbol{x}_E$ il rispettivo ξ_E associato debba essere maggiore di 1 (e.g. se $\boldsymbol{x}_E$ è un positivo, ricade nel semispazio negativo individuato da H_S; ovvero la distanza tra $\boldsymbol{x}_E$ e H_+ deve essere superiore al margine). Gli ξ_i rappresentano quindi dei coefficienti di penalità associati ai punti; e per quanto detto prima la somma degli n ξ_i è una limitazione superiore al numero di punti di errore in fase di *training*.

Si desidera dunque trovare un iperpiano in modo tale massimizzare il margine di separazione fra le due classi (la distanza tra i due iperpiani H_+ e H_- paralleli ad H_S e passanti rispettivamente per $\boldsymbol{x}_+$ e $\boldsymbol{x}_-$, i più vicini punti non affetti da errore), cercando al contempo di minimizzare l'errore complessivo commesso.

La funzione obiettivo da minimizzare adatta allo scopo è:

$$\frac{||w||^2}{2} + C\left(\sum_{i=1}^{n} \xi_i\right)^k \tag{2.32}$$

dove C, scelto dall'utente, controlla la penalità associata all'errore. Per ogni intero positivo k il problema risulta essere ancora convesso; e per $k = 1$ o $k = 2$ è ancora un problema di programmazione quadratica.

Fatte le considerazioni analoghe al caso linearmente separabile, L_P diventa:

$$L_P = \left[\frac{||w||^2}{2} + C\left(\sum_{i=1}^{n} \xi_i\right)^k\right] - \sum_{i=1}^{n} [\alpha_i y_i (w \cdot \boldsymbol{x_i} + b) - 1 + \xi_i] - \sum_{i=1}^{n} \mu_i \xi_i \tag{2.33}$$

dove i $\mu_i \geq 0$ sono gli n ulteriori moltiplicatori di Lagrange relativi alle variabili ξ_i. La scelta di $k = 1$ ha il vantaggio che, nel passaggio alla forma duale del problema, né gli ξ_i né i coefficienti μ_i compaiono in L_p, che rimane:

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{\substack{i=1 \\ j=1}}^{N} \alpha_i \alpha_j y_i y_j \ \boldsymbol{x_i} \cdot \boldsymbol{x_j} \tag{2.34}$$

soggetta alla ulteriore condizione:

$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \tag{2.35}$$

da cui $\alpha_i = C - \mu_i$, e poiché $\mu_i \geq 0$, si ha $\alpha_i \leq C$. In definitiva, l'unica differenza rispetto alla formulazione orginaria è che il vincolo sui moltiplicatori α_i è adesso $0 \leq \alpha_i \leq C$. Questa formulazione del problema è anche detta *soft-margin* [6].

Per brevità, verranno omesse le condizioni di Karush-Kuhn-Tucker relative al nuovo L_P, delle quali si può trovare un elenco in [3]. Ci si limiterà a dire, in questa sede, che la condizione di complementarità per il calcolo di b assume la forma:

$$\alpha_i (y_i (w \cdot \boldsymbol{x_i} + b) - 1 + \xi_i) = 0 \quad \forall\, i = 1, 2, \ \ldots\ n\ . \tag{2.36}$$

da cui:

$$b = \frac{1 - \xi_i}{y_i} - w \cdot \boldsymbol{x_s} \tag{2.37}$$

per la quale valgono le stesse considerazioni del caso linearmente separabile.

Risolto il nuovo problema di ottimizzazione e trovati w e b opportuni, la fase di test è analoga al caso precedente.

2.3 Funzioni kernel e SVM non lineari

L'estensione del *training SVM* al caso *soft-margin* si propone di rimediare al caso in cui i punti su cui operare non siano perfettamente separabili tramite un iperpiano. Tuttavia sono comuni i casi in cui, *non solo* essi *non* sono linearmente separabili, ma in aggiunta presentano configurazioni più complesse; cosicché ha poco senso cercare di separarli ricorrendo ad iperpiani. Si veda un esempio in Figura F, che illustra graficamente il problema in $\mathbb{R}^2$.

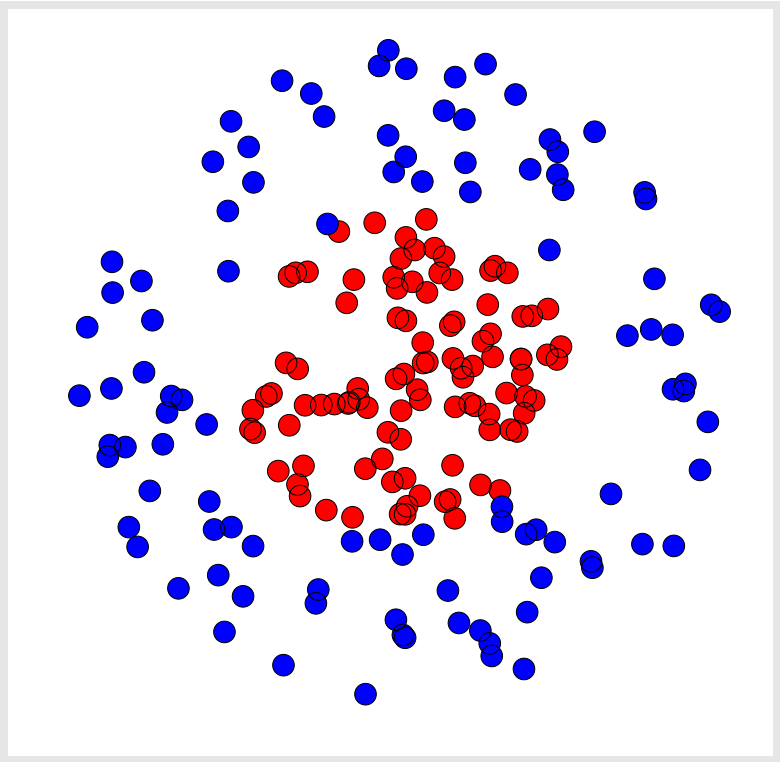

Figura F: *Due classi non linearmente separabili in* $\mathbb{R}^2$

Il modo in cui tale problematica viene elegantemente ed efficacemente affrontata dalle *SVM* è uno dei punti di forza a favore del loro utilizzo in applicazioni pratiche. L'idea di base è la seguente: si supponga di conoscere una funzione ϕ che trasforma i punti $\boldsymbol{x}_i$, mappandoli dal loro dominio $T_+ \cup T_-$ (lo spazio dei dati) ad uno spazio F (lo spazio delle

feature) a dimensione maggiore. Sia quindi ϕ:

$$\begin{aligned} \phi : & D \subseteq \mathbb{R}^d \longrightarrow F \subseteq \mathbb{R}^m \\ & x \longmapsto \phi(x) \end{aligned} \tag{2.38}$$

con $m > d$. L'intuizione è quella di separare *non* i punti $\boldsymbol{x_i}$ con un iperpiano in $\mathbb{R}^d$, quanto piuttosto le immagini $\phi(\boldsymbol{x_i})$ con un iperpiano in F; dove esse risultano più facilmente (e in alcuni casi linearmente) separabili. Ciò significa che punti soggetti al problema di ottimizzazione *non* sono gli x_i ma i relativi $\phi(\boldsymbol{x_i})$. Per predire l'etichetta di una nuova osservazione $\boldsymbol{x}$, è necessario valutare quindi la posizione di $\phi(\boldsymbol{x})$ rispetto all'iperpiano di separazione che risiede nello spazio delle features F.

Questo approccio sembra ragionevole, ma presenta uno svantaggio: osservando le equazioni che caratterizzano il problema di ottimizzazione — e in particolare la (2.22) — si intuisce come la fase di addestramento sia basata fondamentalmente sul calcolo di prodotti scalari del tipo $\phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j})$ (essendo i dati mappati in F prima di effettuare l'ottimizzazione). Poiché F ha in genere un elevato numero di dimensioni, ciò comporta che la fase di addestramento può risultare computazionalmente pesante da effettuare. Tale potenziale inefficienza è aggirata facendo uso del cosiddetto **kernel trick** [7].

Si consideri la funzione $\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ la quale mappa un punto $x \in \mathbb{R}^2$ (x_1 e x_2 sono le sue componenti) nello spazio $\mathbb{R}^3$ (si assume al momento per semplicità di trattazione che m non sia molto grande).

Si calcoli il prodotto scalare $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$:

$$\begin{aligned} \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y}) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2) \cdot (y_1^2, \sqrt{2}y_1y_2, y_2^2) \\ &= x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2 \\ &= (x_1y_1 + x_2y_2)^2 \\ &= ((x_1, x_2) \cdot (y_1, y_2))^2 = (\boldsymbol{x} \cdot \boldsymbol{y})^2 \ . \end{aligned} \tag{2.39}$$

Si evince, dai passaggi sopraelencati, come sia stato ricavato un modo per calcolare il suddetto prodotto scalare senza dover ricorrere necessariamente al calcolo esplicito di ϕ, e quindi di dover operare all'interno dello spazio F, con le sue numerose m componenti.

Il "trucco" (letteralmente, *trick*) consiste quindi nell'individuare una **funzione kernel** K tale che $K(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$, ovvero una differente espressione per il calcolo del prodotto scalare che consenta di ottenere tale valore senza "passare" per ϕ.

Il kernel $K_2 = (\boldsymbol{x} \cdot \boldsymbol{y})^2$ utilizzato nel precedente esempio è detto kernel *quadratico*; una sua generalizzazione è data dal kernel *polinomiale* $K_p = (\boldsymbol{x} \cdot \boldsymbol{y})^p$. Sebbene nell'esempio fornito non sia immediatamente evidente l'alta dimensionalità di F si consideri che, dato K_p, la ϕ associata produce una esplosione combinatoria del numero di dimensioni in F, che risulta uguale a [2] [3]

$$\binom{d+p-1}{p} \tag{2.40}$$

Per quanto riguarda le *SVM*, fissato appositamente un kernel K (la sui scelta deriva da un'analisi dei dati da trattare), è possibile sostituire $K(\boldsymbol{x}, \boldsymbol{y})$ ovunque si presenti di un prodotto scalare del tipo $\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$. Si consideri ad esempio il calcolo della funzione di decisione (2.29) in presenza di una trasformazione ϕ:

$$\begin{aligned} f(\boldsymbol{x}; w, b) &= sign(w \cdot \boldsymbol{x} + b) \\ &= sign\left(\left(\sum_{i=1}^{n} \alpha_i y_i \phi(\boldsymbol{x_i})\right) \cdot \phi(\boldsymbol{x}) + b\right) \\ &= sign\left(\sum_{i=1}^{n} \alpha_i y_i \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x}) + b\right) \\ &= sign\left(\sum_{i=1}^{n} \alpha_i y_i K(\boldsymbol{x_i}, \boldsymbol{x}) + b\right) \end{aligned} \tag{2.41}$$

Notare come in tale formulazione sia possibile predire l'etichetta dell'osservazione $\boldsymbol{x}$ senza ricavare direttamente il valore di w. Verrà ripreso un ragionamento simile nel **Capitolo 3**, quando si manifesterà il problema di dover calcolare la distanza tra $\boldsymbol{x}$ e l'iperpiano di separazione in F; con w incognito.

Per avere un'idea visuale di come agisce il kernel nello spazio dei dati, si veda la Figura G, la quale mostra che, data la natura non lineare di ϕ, all'iperpiano nello spazio delle *features* vi corrisponde una superficie di separazione (*decision boundary*). Essa, a differenza di quanto succede nel caso lineare (e.g con una retta in $\mathbb{R}^2$, si presta meglio nell'adattarsi — in un certo senso — alla forma delle classi. Il kernel utilizzato in questo caso — e negli esperimenti del **Capitolo 4** — è:

$$K(\boldsymbol{x}, \boldsymbol{y}) = e^{-\frac{||\boldsymbol{x}-\boldsymbol{y}||^2}{2\sigma^2}} \tag{2.42}$$

noto come *RBF* kernel (*radial basis function*, funzione a base radiale) [3].

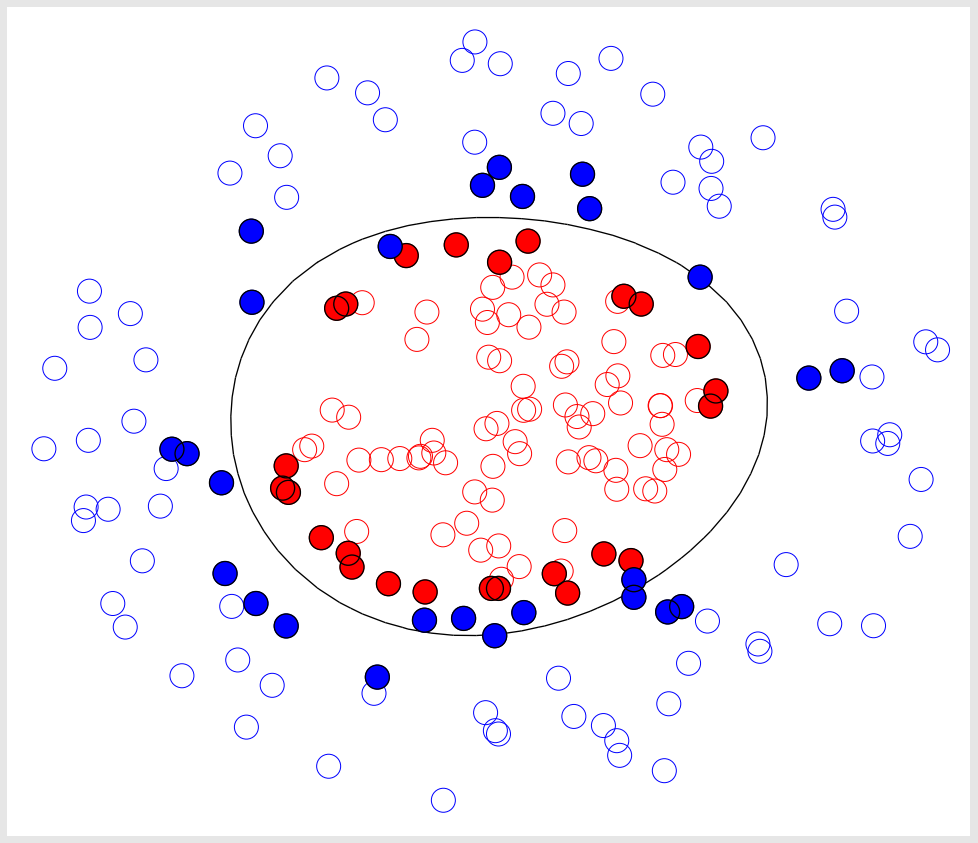

Figura G: *Decision boundary per due classi non linearmente separabili in* $\mathbb{R}^2$*, identificato tramite kernel rbf. Sono evidenziati i vettori di supporto*

È inoltre importante notare come il kernel *trick* non si applichi solamente alle *SVM*, ma potenzialmente a tutti i metodi che fanno uso di prodotti scalari nella suddetta forma (si parla così di *metodi kernel*). Per una trattazione approfondita di tali applicazioni, si veda [10].

2.4 Ulteriori considerazioni

Complessità Nel completare infine la presentazione delle *SVM*, oggetto di questo capitolo, verranno esposte alcune considerazioni relative al loro utilizzo in applicazioni reali. Il nucleo dell'utilizzo delle *SVM* risiede nella risoluzione del problema di programmazione quadratica già discusso; esistono numerose librerie e dei pacchetti software per l'ottimizzazione vincolata (*general purpose*) che, in casi semplici (ovvero per *dataset* di dimensione contenuta) danno risultati soddisfacenti.

Quando la dimensione del problema da risolvere è grande, occorrono tecniche più raffinate per poter lavorare sui dati in modo efficiente. Discutere dell'implementazione concreta di suddette tecniche SVM va ben oltre gli scopi del presente lavoro; si accenna in questa

sede solo al fatto che alcuni algoritmi dedicati operano suddividendo i dati in sottoproblemi (si veda "*chunking*") rendendo il *training* parallelizzabile. [3]

In genere, la complessità computazionale di tali algoritmi si colloca fra $O(n^3)$ e $O(n^2)$ [3]. Per quanto riguarda il *testing* di una singola osservazione si ha invece, nel caso banale, una complessità di $O(ks)$, dove k indica il numero di operazioni effettuate nella valutazione del kernel K scelto, s è il numero di vettori di supporto — vedasi la (2.41). Nel caso lineare, dove w è calcolabile esplicitamente (senza ricorrere a K) dopo l'addestramento, la complessità di *testing* si riduce invece a $O(1)$.

Implementazioni Fra le implementazioni che si ritengono maggiormente degne di nota (in quanto più popolari) vi sono: *LIBSVM* [1] (una libreria *open-source* scritta in C++ e *portata* su un'innumerevole quantità di linguaggi e piattaforme), SVMLight [2], scikit-learn [3] (framework per *machine learning* in Python), WEKA [4] (software per *machine learning* scritto in Java), e infine le API dello *Statistics Toolbox* presente in *MATLAB* [5] (la cui *Student Version r2014a* è stata utilizzata per lo sviluppo di questa tesi). Una lista di implementazioni si trova in [9].

Svantaggi Come già fatto presente, la fase di training può essere computazionalmente lenta quando si trattano dati dalla grande entità (si immagini di avere a che fare con *dataset* di milioni di elementi). In tali casi, non è quindi da trascurare neanche il costo in termini di memoria occupata, sia in fase di training che di testing — in virtù di quanto detto prima, se il kernel è non lineare, occorre memorizzare tutti i vettori di supporto per il calcolo di f.

Le *performance* di classificazione di una *SVM* sono inoltre fortemente legate alla scelta del kernel, che può essere difficile da identificare. Tutto ciò ricordando sempre che una *SVM* può solamente essere utilizzata in presenza di $k = 2$ classi. Esistono comunque lavori volti a generalizzare le *SVM* per la gestione del caso multiclasse secondo un approccio *single machine*, ovvero risolvendo un unico problema di ottimizzazione (quindi un singolo *training*) per tutte le k classi [3]. Nel corso del **Capitolo 3** si tratteranno invece metodi per l'estensione di classificatori binari al caso multiclasse tramite aggregazione di più "macchine" (approccio certamente più semplice).

[1] http://www.csie.ntu.edu.tw/ cjlin/libsvm/

[2] http://svmlight.joachims.org/

[3] http://scikit-learn.org/stable/

[4] http://www.cs.waikato.ac.nz/ml/weka/

[5] http://www.mathworks.it/it/help/stats/support-vector-machines.html

Capitolo 3

Classificazione Multiclasse

Di seguito una presentazione dei due schemi di aggregazione per classificatori binari che consentono di ottenere classificatori k-ari, nonché il criterio di unificazione fra essi — oggetto di questo elaborato.

3.1 Metodo one-vs-all

Sia T un *dataset* di k classi di punti ($k = |C|$). Si supponga per semplicità di notazione che le etichette di classe siano $C = \{1, 2, \ ... \ , k\}$. Questo metodo consente di costruire un classificatore k-ario a partire da k classificatori binari (e.g. *SVM*); ciò è realizzato — come ben espresso dal nome alternativo *one-vs-rest* — considerando di volta in volta la i-esima classe ($i \in C$) "contro" il resto dei punti. Più formalmente, per il classificatore i-esimo, la classe dei positivi e quella dei negativi sono così definite:

$$T_{i_+} = \left\{ \boldsymbol{x}_+ \in \mathbb{R}^d \mid (\boldsymbol{x}_+, i) \in T \right\} \tag{3.1}$$

$$T_{i_-} = \left\{ \boldsymbol{x}_- \in \mathbb{R}^d \mid (\boldsymbol{x}_-, j) \in T; \ j \neq i \right\} \tag{3.2}$$

Il classificatore i-esimo è addestrato dunque su di esse. I positivi T_{i_+} sono tutti i punti di classe i; i negativi T_{i_-} sono tutti i punti di classe diversa da i. Una descrizione grafica di questa selezione è data in Figura A.

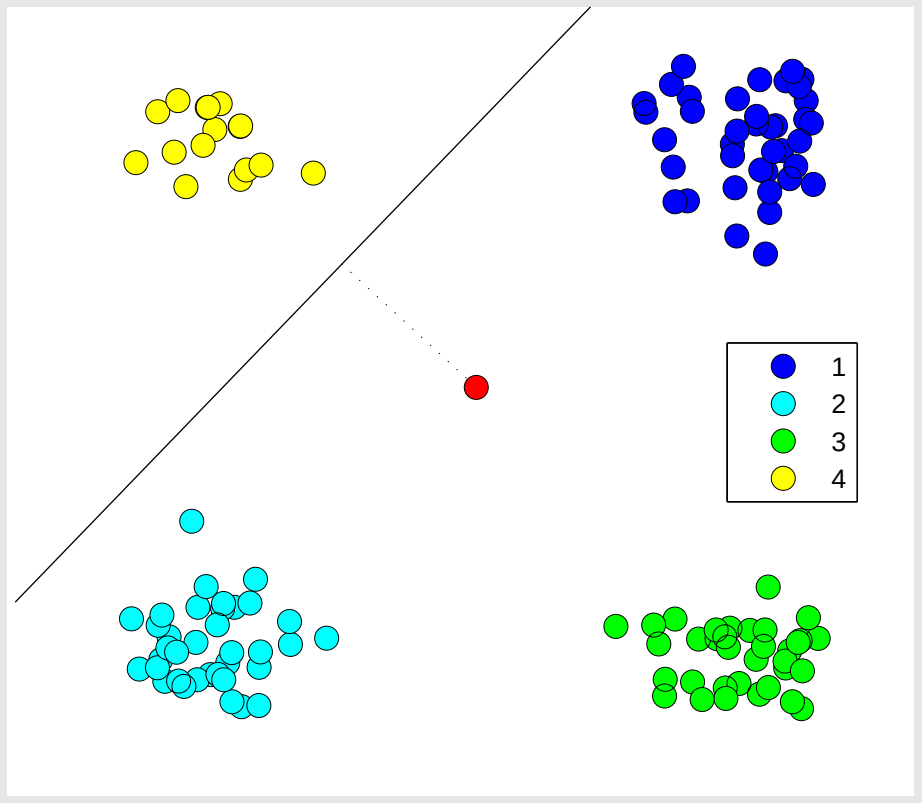

Figura A: *Esempio di iperpiano per classe 4 vs rest in* $\mathbb{R}^2$. *(con kernel lineare) L'elemento in rosso rappresenta un vettore di test*

Completata la fase di training, resta da definire in che modo venga classificata una nuova osservazione $\boldsymbol{x}$. L'idea è di mandare in input $\boldsymbol{x}$ ad ogni singolo classificatore (e.g. SVM_i), quindi effettuando k test sulle k macchine binarie. Ognuna di tali macchine classificherà l'osservazione con etichetta $+1$ o -1 (quindi come di classe i-esima o "resto"). Non si ha dunque un modo immediato per ottenere l'etichetta corretta relativa all'intero insieme C; non è cioè sufficiente il solo calcolo delle funzioni di decisione f_i.

Un criterio ragionevole (ed in pratica utilizzato) è il seguente: si supponga che i k classificatori binari siano *SVM*. Per ogni macchina SVM_i si ottiene, dopo il *training*, un relativo iperpiano di separazione H_i — sia esso nello spazio dei dati $\mathbb{R}^d$ o nello spazio delle features F, a seconda del kernel utilizzato.

Si considerino allora le distanze euclidee dell'osservazione $\boldsymbol{x}$ da ogni iperpiano H_i (calcolate tramite la formula già indicata nel **Capitolo 2**). Il metodo prevede di assegnare $\boldsymbol{x}$ alla classe positiva relativa alla SVM il cui iperpiano di separazione abbia distanza massima da $\boldsymbol{x}$. Siano le suddette distanze $\delta_i = d(\boldsymbol{x}, H_i)$. Più formalmente, la classe assegnata a $\boldsymbol{x}$ sarà:

$$\arg\max_i \delta_i \tag{3.3}$$

Si noti che, mentre nel caso lineare tali distanze siano facili da calcolare (poiché w è ricavato esplicitamente), per quanto detto nella **Sezione 2.3** (equazione 2.41) se il kernel non è lineare, il calcolo di tali distanze non è banale. In questo caso infatti il w dell'iperpiano di separazione nello spazio F non è direttamente calcolato (sebbene sia possibile valutare ancora f per mezzo di K); poiché:

$$w = \sum_{i=1}^{n} \alpha_i y_i \phi(\boldsymbol{x_i}) \tag{3.4}$$

e, come precedentemente visto, non è desiderabile lavorare esplicitamente con ϕ.

Il problema è affrontato nel seguente modo: si desidera calcolare la distanza tra $\phi(\boldsymbol{x})$ e l'iperpiano di separazione H_F nello spazio delle feature F (sia w il vettore ad esso normale):

$$d(\phi(\boldsymbol{x}), H_F) = \frac{|w \cdot \phi(\boldsymbol{x}) + b|}{||w||} \tag{3.5}$$

che può essere così riscritta esplicitando w:

$$d(\phi(\boldsymbol{x}), H_F) = \frac{\left|\sum_{i=1}^{n} \alpha_i y_i \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x}) + b\right|}{||w||} \tag{3.6}$$

(si ricorda che gli $\boldsymbol{x_i}$ sono i punti di *training*) e utilizzando il relativo kernel K:

$$d(\phi(\boldsymbol{x}), H_F) = \frac{\left|\sum_{i=1}^{n} \alpha_i y_i K(\boldsymbol{x_i}, \boldsymbol{x}) + b\right|}{||w||} \,. \tag{3.7}$$

Il calcolo del denominatore della (3.6) diventa adesso possibile grazie all'uso del kernel. Per quanto riguarda il denominatore, si consideri $||w||^2 = w_1^2 + \, ... \, w_d^2 = w \cdot w$.

$$\begin{aligned} w \cdot w &= \sum_{i=1}^{n} \alpha_i y_i \phi(\boldsymbol{x_i}) \cdot \sum_{j=1}^{n} \alpha_j y_j \phi(\boldsymbol{x_j}) = \sum_{i=i}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \phi(\boldsymbol{x_i}) \cdot \phi(\boldsymbol{x_j}) \\ &= \sum_{i=i}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(\boldsymbol{x_i}, \boldsymbol{x_j}) \end{aligned} \tag{3.8}$$

Ancora una volta è stata trovata un'espressione alternativa (per il calcolo di $||w||$) che non richieda necessariamente l'uso di ϕ.

Pro e Contro In riferimento a quanto verrà detto nella prossima sezione, il metodo *one-vs-all* ha complessità computazionale relativamente contenuta: il *training* richiede $O(k \cdot SVM)$, dove con *SVM* si indica la complessità del *training* di una singola macchina binaria. Considerazioni analoghe valgono per il test di una singola osservazione (lineare in k).

Si tenga presente, comunque, che in questo metodo ogni *SVM* effettua l'addestramento su *tutti* i dati in input, se pur con etichette diverse. Ciò implica che — a differenza di quanto avviene nel metodo presentato a breve — il *training* dei singoli classificatori binari possa risultare computazionalmente dispendioso. Si consideri anche che, in pratica, un numero elevato di punti di training può comportare un alto numero di vettori di supporto: ciò può ripercuotersi negativamente, in termini di calcolo, anche durante il test.

Naturalmente, quanto detto per il metodo *one-vs-all* può applicarsi a classificatori binari che non siano basati su *SVM*, purché sia possibile calcolare opportuni valori di *score* per l'osservazione in input (in questo caso le distanze δ_i).

3.2 Metodo one-vs-one

Si utilizzi la stessa notazione definita per il metodo *one-vs-all*. Come indicato dal nome, il metodo *one-vs-one* prevede invece di confrontare le varie coppie di classi "uno contro uno". Se il *dataset* T ha k classi, viene addestrato un classificatore per ogni coppia di etichette (i, j) con $i, j = 1, ..., k$ e $i \neq j$. Il numero totale di coppie di questo tipo è dato da:

$$R = \binom{k}{2} = \frac{k(k-1)}{2} \, . \tag{3.9}$$

Fissata quindi la coppia (i, j), il classificatore r-esimo ($r = 1, ..., R$) è addestrato con positivi e negativi dati da:

$$T_{r_+} = \left\{ \boldsymbol{x}_+ \in \mathbb{R}^d \mid (\boldsymbol{x}_+, i) \in T \right\} \tag{3.10}$$

$$T_{r_-} = \left\{ \boldsymbol{x}_- \in \mathbb{R}^d \mid (\boldsymbol{x}_-, j) \in T \right\} \tag{3.11}$$

Si noti che, poiché l'output della funzione di decisione f_r è $+1$ o -1, vi è la necessità di "rinominare" le etichette (scegliendo indifferentemente quale classe rappresenti i positivi e quale i negativi), per poi ritornare a quelle originarie. A differenza di quanto avviene in *one-vs-all* comunque, se pur con etichette diverse, le classi T_{r_+} e T_{r_-} dei singoli classificatori binari sono ancora classi distinte di T.

L'output del classificatore r-esimo per la coppia (i, j) corrisponde quindi univocamente all'etichetta i oppure a j — si può cioè supporre direttamente che $f_r(\boldsymbol{x}) \in \{i, j\}$. In tale metodo, una predizione di questo tipo è interpretata come una *votazione* a favore di una delle due classi.

Addestrati gli R classificatori, per ogni singola osservazione $\boldsymbol{x}$ di test è possibile costruire una matrice M siffatta:

$$M_{k,k} = \begin{pmatrix} 0 & y_{1,2} & \cdots & y_{1,k} \\ y_{1,2} & 0 & \cdots & y_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ y_{1,k} & y_{2,k} & \cdots & 0 \end{pmatrix} \tag{3.12}$$

dove $y_{i,j}$ indica l'etichetta in C predetta per $\boldsymbol{x}$ dal classificatore addestrato sulla coppia di classi (i, j). M è simmetrica (dato che non conta l'ordine delle classi, i.e. per (i, j) e (j, i) viene costruito un solo classificatore). È possibile considerare solo la parte triangolare superiore o inferiore di M. I valori sulla diagonale sono posti a 0 poiché non esiste alcun classificatore per (i, i) (tale f sarebbe banalmente costante: non farebbe alcuna scelta).

Un criterio per la selezione della classe cui assegnare all'osservazione $\boldsymbol{x}$ potrebbe ad esempio essere la selezione dell'etichetta che occorre con più frequenza in M: la classe che ha ricevuto numero più alto di voti. Questa è la strategia adottata nell'implementazione del codice MATLAB, utilizzata per effettuare gli esperimenti presentati nel **Capitolo 4**. È comunque possibile modificare M in modo da considerare altri fattori; argomento non affrontato in questa sede.

Pro e Contro Si noti innanzitutto come il metodo *one-vs-one* non necessiti — almeno nella versione appena presentata — del calcolo di valori aggiuntivi (e.g. le distanze δ_i). Ciò si traduce in una maggiore facilità di implementazione rispetto al metodo *one-vs-all*.

La complessità computazionale del *training* è $O(k^2 \cdot SVM)$; si consideri tuttavia che in questo caso ogni classificatore è addestrato solo su una parte del *training set*. La risoluzione del problema di ottimizzazione per il singolo classificatore binario è quindi più veloce rispetto al caso *one-vs-all* (a parità di *training set* e kernel usato). Analogamente, il test ha complessità quadratica in k.

Durante l'implementazione degli esperimenti presentati nel **Capitolo 4** è stato osservato sperimentalmente che, sebbene il primo metodo abbia minore complessità teorica, per *dataset* di dimensioni contenute il metodo *one-vs-one* performa meglio di *one-vs-all*. Considerazioni analoghe valgono per il test, in concordanza a quanto detto nella **Sezione 3.1** sul numero di vettori di supporto [11]. In pratica, non è banale stabilire quale dei due metodi sia più veloce in termini computazionali: tutto dipende dal *dataset* in esame e dalle funzioni kernel scelte. Una comparazione fra i due metodi può trovarsi in [11].

Infine è bene notare che, secondo la formulazione sopra data, quando due (o più) classi hanno pari numero di voti il comportamento del classificatore non è definito. Un criterio per la risoluzione di tale ambiguità può essere la scelta della classe con maggiore cardinalità, fra esse. Tale circostanza sembra comunque presentarsi alquanto raramente.

3.3 Un criterio di unificazione

Studi precedenti in letteratura tendono a mostrare, in genere, che lo schema *one-vs-one* consente di raggiungere performance (di generalizzazione) maggiori di *one-vs-all* [13]. Altri studi sono volti invece in senso contrario [12]. Si cercherà invece di formulare in questa sede un criterio per l'applicazione integrata dei due metodi.

Si addestrino simultaneamente (a partire da un *training set* T) sia gli R classificatori binari secondo lo schema *one-vs-all*, sia quelli secondo *one-vs-one* (k). Sia x l'osservazione in input e y la relativa etichetta da predire.

Si applichi il metodo *one-vs-all* a x, considerando dunque le k distanze $\delta_1, \delta_2, \ ..., \ \delta_k$. Lo schema *one-vs-all* assegnerà a x l'etichetta relativa al classificatore binario con iperpiano a massima distanza da x. L'intuizione dietro al criterio è semplice: si considerino le due distanze δ_a, δ_b dal valore più grande. Se esse risultano approssimativamente uguali (secondo una tolleranza prestabilita) significa che il classificatore *one-vs-all* non riesce a discriminare sufficientemente le due classi a, b. Siano le distanze normalizzate nell'intervallo $[0, 1]$ con:

$$\frac{\delta_j - max\ \delta_i}{max\ \delta_i - min\ \delta_i} \tag{3.13}$$

e siano ordinate in modo decrescente, così da avere una permutazione $\delta_{(p)}, \delta_{(p-1)}, \ ..., \ \delta_1$. Gli indici delle due distanze dal valore più grande sono quindi (p) e $(p-1)$. Si consideri adesso la condizione

$$|\delta_{(p)} - \delta_{(p-1)}| < \epsilon, \qquad \epsilon \in [0, 1] \tag{3.14}$$

fissata una tolleranza ϵ. Il verificarsi di questa condizione è da interpretare come una incapacità da parte del classificatore di distinguere "bene" le due classi, durante la predizione dell'etichetta y. Si utilizzano quindi i classificatori binari del tipo *one-vs-one* in modo che

$$y = M_{(p),(p-1)} \tag{3.15}$$

ovvero l'etichetta per x sarà quella predetta dal classificatore binario addestrato sulla coppia di classi (p), $(p-1)$. Se invece la (1.14) non si verifica, si utilizza semplicemente l'etichetta predetta con il metodo *one-vs-all*.

La motivazione che ha portato alla scelta del suddetto criterio è la seguente: per *dataset* composti da un elevato numero di classi k, il test di x sul metodo *one-vs-one* richiede tempo quadratico in k. Di contro, il test con *one-vs-all* è più efficiente — richiede tempo lineare in k — ma in genere fornisce risultati meno precisi.

L'idea è quindi di sfruttare *one-vs-all* quando esso garantisce una buona capacità di discriminazione (ovvero nella maggior parte dei casi) e di ricorrere a un classificatore binario *ad-hoc* quando vi è un "dubbio" sull'assegnazione di x a una delle due classi. Notare come *non* sia necessario completare l'intera matrice M testando x per tutte le R possibili coppie di classi. Tale metodo è una estensione di *one-vs-all* che utilizza una "approssimazione" di *one-vs-one*, mantenendo bassa (lineare) la complessità del test, in k.

Si noti tuttavia che (a) la fase di *training* è più dispendiosa (b) la quantità di memoria occupata è ovviamente più grande.

Nel **Capitolo 4** verrà mostrato come questo criterio risulti in un buon compromesso fra accuratezza e performance di calcolo; risultando in certi casi leggermente più preciso di *one-vs-all* e talvolta persino di *one-vs-one*.

Capitolo 4

Applicazione

4.1 Validazione

Prima di presentare gli esperimenti effettuati — e relative comparazioni fra i risultati ottenuti — è necessario dare una breve descrizione dei procedimenti adottati per la validazione delle *performance* dei vari classificatori.

Riconoscimento medio Fissato un qualsiasi metodo di classificazione multiclasse — e un *dataset* di dati pre-etichettati, da suddividere in *training* e *test set* — si applichi il metodo scelto ad ogni elemento di test. Ogni elemento x avrà dunque una etichetta "reale" e una predetta; qualora esse non corrispondano si dice che x è stato misclassificato.

Si associa al *test set* una **matrice di confusione** C di dimensioni $k \cdot k$, dove $C(i, j)$ contiene il numero di osservazioni di test di classe i, etichettate dal classificatore con j. C fornisce quindi informazioni (qualitative e quantitative) sulle performance del metodo. I valori sulla diagonale $C(i, i)$ rappresentano il numero di osservazioni di classe i classificate correttamente. Una misura iniziale della bontà del classificatore è data quindi dalla cosiddetta *accuracy*:

$$Accuracy = \frac{\sum_{i=1}^{k} C(i, i)}{|T|} \tag{4.1}$$

ovvero il numero totale di osservazioni di test classificate correttamente.

La *accuracy* non è l'unico indice di valutazione per un classificatore: da C possono essere infatti ricavate varie misure. Per quanto riguarda gli esperimenti discussi nelle Sezioni successive, l'indice utilizzato è il *riconoscimento medio*:

$$\rho = \frac{\sum_{i=1}^{k} C_{\%}(i,i)}{k} \tag{4.2}$$

dove $C_{\%}$ è la matrice di confusione C preventivamente riportata in percentuale (dividendo ogni elemento per la somma degli elementi della sua riga). ρ è quindi la media delle percentuali di riconoscimento (simile ad *accuracy*) calcolate sulle singole classi.

Quando si lavora con *dataset* "sbilanciati" — ovvero in cui vi è una classe predominante in numero — la *accuracy* tende a sovrastimare le reali capacità di discriminazione del classificatore: osservazioni appartenenti a classi di minore cardinalità tendono ad essere misclassificate, ciononostante la *accuracy* si mantiene alta a causa dell'alto riconoscimento per la classe predominante. La scelta del riconoscimento medio come indice è un tentativo di ovviare al problema.

Cross-Validazione Un altro espediente usato al fine di ottenere misure di performance realistiche è la cosiddetta **k-fold cross-validation** (con $k = 10$).

L'idea dietro la *cross-validation* è la seguente: il *dataset* viene suddiviso in k parti disgiunte e il procedimento consta di k iterazioni; durante la i-esima iterazione, la i-esima parte ($i = 1, \ldots, k$) viene usata come *test set* e il resto del *dataset* come *training set*. Si hanno quindi k classificatori addestrati su sottoinsiemi del *dataset* distinti. Ogni classificatore ha una matrice di confusione ed un valore di riconoscimento medio ρ_i. Il valore finale di ρ assunto come indice della bontà del classificatore è la media dei ρ_i, accompagnato dalla relativa deviazione standard dei ρ_i.

La *cross-validation* tende ad "attenuare" il valore dell'indice di riferimento, mediandolo rispetto a più scelte del *training set*. Un alto valore di deviazione standard σ associato a ρ indica che le performance del metodo (fissato *dataset*, kernel ed eventuali parametri) sono molto sensibili alla scelta del *training set*; viceversa un valore basso di σ indica che il metodo è abbastanza robusto in tal senso.

Considerazioni sui parametri I tre metodi discussi nel **Capitolo 3** (*one-vs-all*, *one-vs-one*, *ibrido*) sono stati testati su vari *dataset* di *benchmark* e confrontati tramite ρ, σ otteunti con *cross-validation*.

Per quanto riguarda il metodo *ibrido* (di unificazione) presentato, si ricordi che la sua applicazione dipende da un parametro di tolleranza ϵ fissato a priori. Si pone quindi il problema di scegliere opportunamente tale valore, affinché il metodo abbia buone performance.

Il procedimento adottato consiste nella ricerca esaustiva di un valore ϵ nel range $[0, 0.25]$ ottimale, cioè tale da massimizzare il valore di ρ. Ciò è stato realizzato facendo variare ϵ (in tale intervallo) con passo discreto (e.g. 0.01); fissato un valore di $\epsilon = \epsilon_i$ il metodo è testato con *cross-validation* (ha ρ_i), il valore di ϵ scelto è infine l'ϵ_i per cui si ottiene ρ_i massimo.

ϵ non è inoltre l'unico parametro incontrato nel corso di questo elaborato: si ricordi e.g. il parametro C nella formulazione *SVM soft-margin*, e σ relativo al kernel *RBF*. Anche essi possono essere individuati mediante *cross-validation*.

Si osservi tuttavia che l'introduzione di ulteriori parametri aumenta "lo spazio della ricerca" dei loro valori (e.g la coppia (C, ϵ) per cui ρ è massimo). Visto che la *cross-validation* è di per sé computazionalmente dispendiosa, si è preferito trascurare tali aspetti in questa sede, posto che lo scopo della presente tesi non consiste nella classificazione ottimale dei *dataset* di seguito presentati, quanto nel confrontare uniformemente i tre metodi a parità di parametri. Si è supposto quindi $C = 1$ e $\sigma_{rbf} = 1$. La scelta di un valore per C è eventualmente lasciata all'implementazione *SVM* sottostante, e.g. tramite criteri euristici.

Una considerazione analoga vale per le funzioni kernel scelte: e.g. fissato il metodo *one-vs-one* con kernel quadratico, si è preferito addestrare *ogni* classificatore binario (per le singole coppie di classi) con kernel quadratico. Ovviamente al fine di massimizzare le performance è possibile selezionare tali kernel *ad-hoc* (e.g. per ogni coppia di classi), in modo fortemente dipendente dal *dataset* in esame. Ancora una volta, ciò che interessa non è trovare la migliore configurazione possibile per massimizzare le performance sui *dataset* di *benchmark*, quanto confrontare i tre metodi a parità di condizioni.

Segue che i risultati di riconoscimento di seguito presentanti sono da interpretare come *migliorabili* e puramente *indicativi* (del fatto che, in determinati casi, essi performano sufficientemente bene nonostante le scelte di parametri e kernel non siano ottimali).

4.2 Esperimenti e Risultati

I *dataset* scelti per l'applicazione pratica dei metodi discussi nel **Capitolo 3** sono: Ecoli[1], Yeast[2], Breast Tissue[3], ottenibili liberamente da *UCI Machine Learning Repository*[4] [8]. Per la classificazione binaria con *SVM* sono state utilizzate le funzionalità *built-in* presenti in MATLAB. A partire da esse, sono stati implementati i metodi *one-vs-one*, *one-vs-all*, *ibrido*; più il codice necessario alla *cross-validation*.

Tutti i metodi sono stati confrontati inoltre con un classificatore *k-Nearest Neighbor* basato anch'esso su distanza euclidea. Un classificatore di questo tipo identifica i k vicini all'osservazione x — secondo la distanza scelta — ed assegna a x l'etichetta più frequente in tali punti. Il parametro k ottimale è stato ricercato con *10-fold cross-validation* (così come ϵ). Le tabelle mostrano i risultati dell'applicazione dei metodi, riportando ρ, σ e parametri. Si denotano con K_1, K_2, K_3 rispettivamente kernel lineare, quadratico e cubico ($p = 3$).

4.2.1 Dataset Ecoli

Il *dataset* Ecoli contiene 336 osservazioni di 8 attributi reali che rappresentano caratteristiche biologiche di batteri della specie Escherichia Coli — legati ai processi digestivi. L'insieme prevede un numero originario di 8 classi, tuttavia tre di esse (e i relativi esempi) sono state rimosse a causa della loro bassa cardinalità.

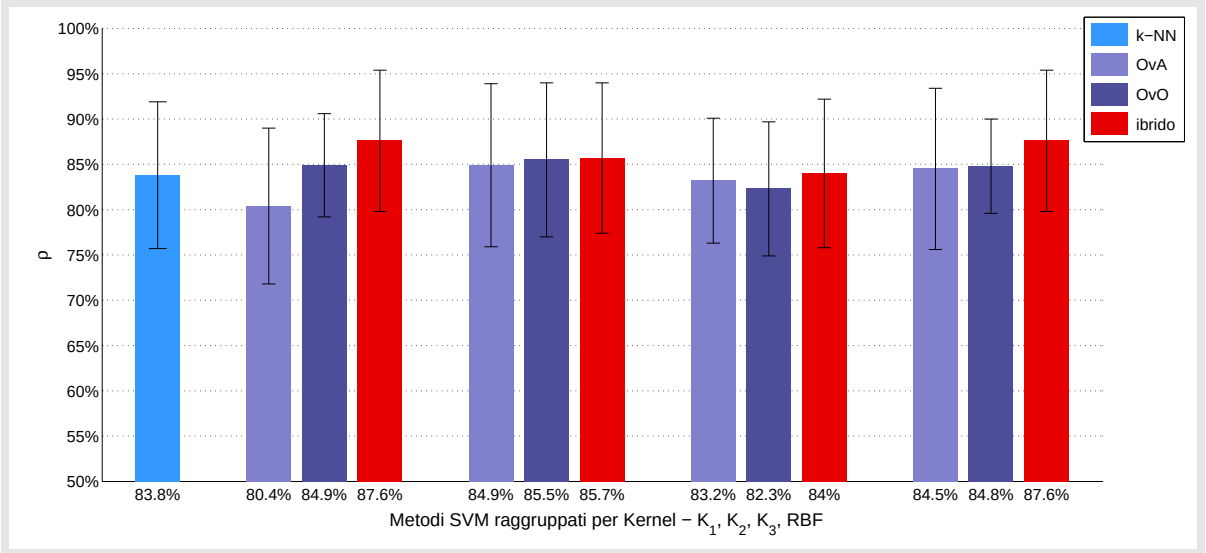

Figura A: *Grafico a barre per il confronto dei metodi su Ecoli*

[1] https://archive.ics.uci.edu/ml/datasets/Ecoli

[2] https://archive.ics.uci.edu/ml/datasets/Yeast

[3] https://archive.ics.uci.edu/ml/datasets/Breast+Tissue

[4] https://archive.ics.uci.edu/ml/

Metodo	Kernel	ρ	σ	Parametro
k-NN	·	83.8 %	8.1 %	$k = 10$
one-vs-all	K_1	80.4 %	8.6 %	·
	K_2	84.9 %	9.0 %	·
	K_3	83.2 %	6.7 %	·
	RBF	84.5 %	8.9 %	·
one-vs-one	K_1	84.9 %	5.7 %	·
	K_2	85.5 %	8.6 %	·
	K_3	82.3 %	7.4 %	·
	RBF	84.8 %	5.2 %	·
ibrido	K_1	87.6 %	7.8 %	$\epsilon = 0.11$
	K_2	85.6 %	8.3 %	$\epsilon = 0.09$
	K_3	84.0 %	8.2 %	$\epsilon = 0.09$
	RBF	87.6 %	7.8 %	$\epsilon = 0.11$

Si può notare come, in questo caso: (a) i metodi *one-vs-all* e *one-vs-one* abbiano prestazioni simili, tranne nel caso lineare dove il secondo risulta più preciso (b) non solo il metodo ibrido migliora le performance di *one-vs-all* (sotto qualsiasi kernel), ma risulta essere leggermente più preciso anche di *one-vs-one*. Notare inoltre come *one-vs-one* nei casi lineare e RBF sia il metodo più robusto rispetto alla scelta del *training set*, visto che ha minore σ.

4.2.2 Dataset Yeast

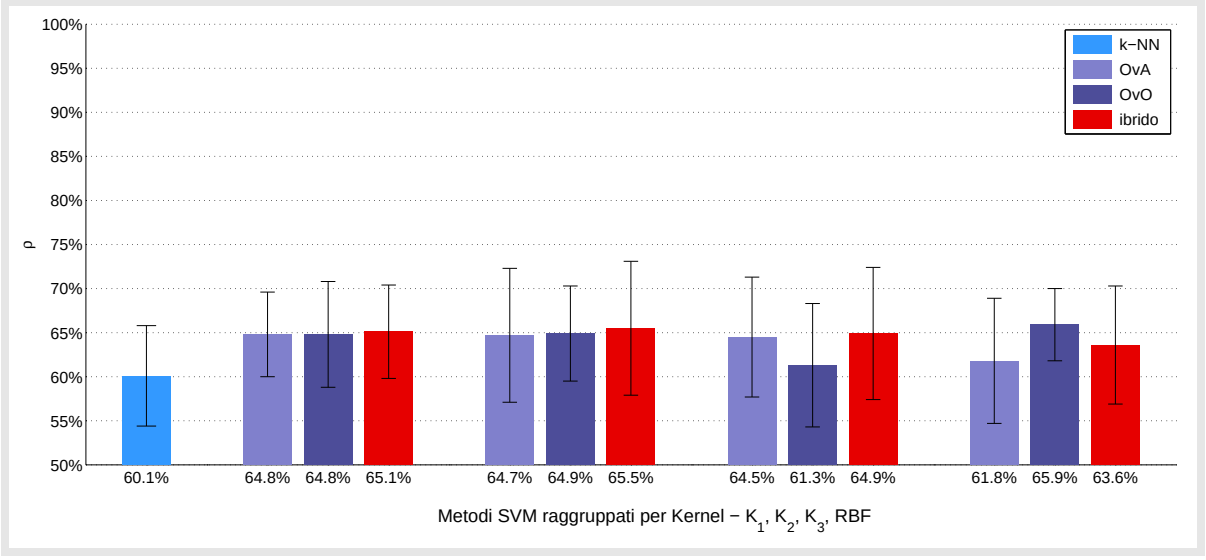

Figura B: *Grafico a barre per il confronto dei metodi su Yeast*

Anche questo *dataset* contiene dati di natura biologica ed in particolare caratteristiche di batteri presenti nel lievito. Di 1484 osservazioni (8 features) suddivise in 10 classi, solo le quattro classi più frequenti sono state mantenute. Nonostante ciò, la classificazione si è rivelata essere un problema abbastanza arduo su tali dati — come è indicato nella tabella successiva (e in figura) dai bassi valori di ρ.

Metodo	Kernel	ρ	σ	Parametro
k-NN	$\cdot$	$60.1\,\%$	$5.7\,\%$	$k = 20$
one-vs-all	K_1	$64.8\,\%$	$4.8\,\%$	$\cdot$
	K_2	$64.7\,\%$	$7.6\,\%$	$\cdot$
	K_3	$64.5\,\%$	$6.8\,\%$	$\cdot$
	RBF	$61.8\,\%$	$7.1\,\%$	$\cdot$
one-vs-one	K_1	$64.8\,\%$	$6.0\,\%$	$\cdot$
	K_2	$64.9\,\%$	$5.4\,\%$	$\cdot$
	K_3	$61.3\,\%$	$7.0\,\%$	$\cdot$
	RBF	$65.9\,\%$	$4.1\,\%$	$\cdot$
ibrido	K_1	$65.1\,\%$	$5.3\,\%$	$\epsilon = 0.18$
	K_2	$65.5\,\%$	$7.6\,\%$	$\epsilon = 0.20$
	K_3	$64.9\,\%$	$7.5\,\%$	$\epsilon = 0.13$
	RBF	$63.6\,\%$	$6.7\,\%$	$\epsilon = 0.18$

Si osservi comunque come i metodi basati su *SVM* si comportino meglio di *k-Nearest Neighbors* (con $k = 20$ opportunamente scelto). È degno di nota inoltre come il metodo ibrido non risulti meno preciso di *one-vs-all*, indipendentemente dal kernel scelto.

4.2.3 Dataset Breast Tissue

Il *dataset* Breast Tissue contiene 106 esempi (di 10 attributi) contenenti varie misurazioni effettuate su tessuti organici (e.g. adiposo, connettivo, etc.) al fine di identificare masse tumorali su tali tessuti. Poiché le 6 classi risultano ben bilanciate, nessuna è stata rimossa per il *training*. Di seguito i risultati:

Metodo	Kernel	ρ	σ	Parametro
k-NN	$\cdot$	$78.3\,\%$	$18.9\,\%$	$k = 5$
one-vs-all	K_1	$60.8\,\%$	$11.2\,\%$	$\cdot$
	K_2	$60.8\,\%$	$17.6\,\%$	$\cdot$
	K_3	$68.3\,\%$	$16.1\,\%$	$\cdot$
	RBF	$62.5\,\%$	$16.3\,\%$	$\cdot$
one-vs-one	K_1	$69.2\,\%$	$14.2\,\%$	$\cdot$
	K_2	$65.8\,\%$	$19.0\,\%$	$\cdot$
	K_3	$67.5\,\%$	$20.6\,\%$	$\cdot$
	RBF	$74.2\,\%$	$21.0\,\%$	$\cdot$
ibrido	K_1	$61.2\,\%$	$11.9\,\%$	$\epsilon = 0.07$
	K_2	$72.5\,\%$	$13.6\,\%$	$\epsilon = 0.08$
	K_3	$73.3\,\%$	$21.1\,\%$	$\epsilon = 0.08$
	RBF	$72.5\,\%$	$20.8\,\%$	$\epsilon = 0.20$

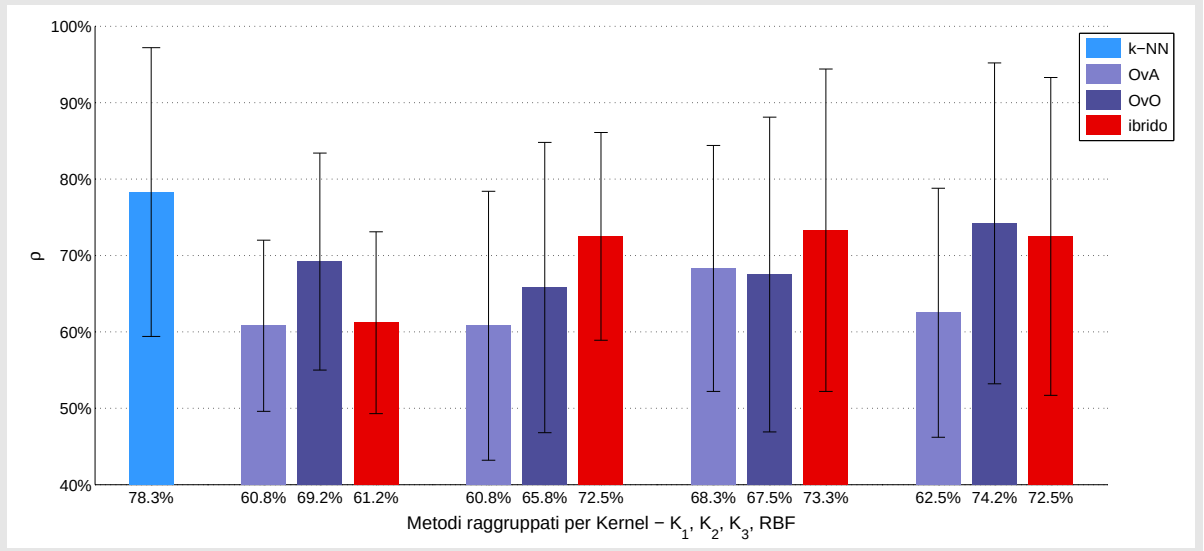

Figura C: *Grafico a barre per il confronto dei metodi su Breast Tissue*

Si noti come tutti i metodi siano in questo caso particolarmente sensibili alla scelta del *training set* (visti gli elevati valori di σ) — *k-NN* incluso — ciò suggerisce che tale problematica sia inerente ai dati piuttosto che al metodo scelto. È evidente come nel caso lineare *one-vs-one* sia significativamente più preciso di *one-vs-all*, e come con kernel quadratico e cubico il metodo ibrido sia più preciso del primo. Con kernel RBF, invece, il metodo ibrido si avvicina comunque a *one-vs-one*, migliorando le prestazioni di *one-vs-all*.

Notare inoltre come, a dispetto della sua semplicità, *k-NN* abbia in questo caso prestazioni migliori di tutti i metodi basati su *SVM*, indipendentemente dal kernel (ciò potrebbe non essere più vero scegliendo adeguatamente i kernel *ad-hoc*, comunque — per quanto detto nella sezione precedente).

4.3 Conclusioni

A partire dagli esperimenti effettuati, sembra lecito dedurre le seguenti conclusioni:

- il metodo *one-vs-one* raggiunge in genere prestazioni maggiori di *one-vs-all*, specialmente a parità di kernel lineare e RBF;
- sotto kernel cubico le prestazioni di *one-vs-one* tendono invece ad abbassarsi;
- in quasi tutti i casi esaminati, il metodo ibrido è in grado di avvicinarsi alle prestazioni di *one-vs-one* e a volte di superarle.

Il criterio nel **Capitolo 3**, pur essendo abbastanza semplice, consente in determinati casi di migliorare la precisione di *one-vs-all* pagando un piccolo prezzo in termini computazionali — ovvero introducendo una ricerca lineare in k; per un totale di $O(2k + 1)$ — ma rimanendo comunque più efficiente di *one-vs-one* durante il test, almeno in linea teorica (non si considera né la complessità di calcolo del kernel scelto né il numero di vettori di supporto).

Ciò suggerisce, per quanto riguarda eventuali sviluppi futuri, che sia possibile lavorare ulteriormente in questa direzione, cercando di trovare un compromesso fra *one-vs-all* e *one-vs-one* formulando criteri più sofisticati e raffinati.

Bibliografia

[1] T. Mitchell, *Machine Learning*. McGraw Hill, 1997.

[2] Bing Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer, 2nd edition, 2001.

[3] Christopher J. C. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition*. 1998.

[4] M.A. Hearst, S.T. Dumais, E. Osman, J. Platt; *Support vector machines*. IEEE Intelligent Systems and their Applications. 1998.

[5] B. E. Boser, I. M. Guyon and V. N. Vapnik; *A Training Algorithm for Optimal Margin Classifiers*. Proc. Fifth Ann. Workshop Computational Learning Theory, ACM Press, New York. 1992.

[6] C. Cortes, V. N. Vapnik; *Support-Vector Networks*. Machine Learning (Volume 20, Issue 3). 1995.

[7] M. A. Aizerman, E. A. Braverman, L. Rozonoer; *Theoretical foundations of the potential function method in pattern recognition learning*. Automation and Remote Control. 1964.

[8] K. Bache, M. Lichman (2013). UCI Machine Learning Repositor [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[9] A. Ben-Hur, J. Weston; *A user's guide to support vector machines*.

[10] J. Shawe-Taylor, N. Cristianini; *Kernel Methods for Pattern Analysis*. Cambridge University Press. 2004.

[11] Chih-Wei Hsu, Chih-Jen Lin; *A comparison of methods for multi-class support vector machines*. IEEE transactions on Neural Networks. 2002.

[12] Ryan Rifkin, Aldebaro Klautau; *In Defense of One-Vs-All Classification*. Journal of Machine Learning Research (5), 2004.

[13] E. L. Allwein, R. E. Schapire, Y. Singer; *Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers*. Journal of Machine Learning Research (1), 2000.