Applicazione di tecniche di Machine Learning per problemi di real-time tracking in reti di videosorveglianza

Gottardo Giuseppe, matr. 621855
Lanzini Andrea, matr. 621860
Zanin Claudia, matr. 626618

Abstract—Il progetto mira ad applicare le tecniche di Machine Learning ad un problema di visual-tracking, cioè ad effettuare l'inseguimento di un oggetto (foreground) in movimento rispetto all'ambiente circostante (background). Il problema è stato ampiamente studiato da un punto di vista statico che, lavorando offline, applica gli algoritmi a una serie di dati precedentemente acquisiti. Lo scopo di questo lavoro è lo sviluppo di un algoritmo che possa essere utilizzato in real-time, cercando di limitare il più possibile il carico computazionale. Le tecniche utilizzate prevedono l'uso della teoria dei Support Vector Machines (SVMs) e sono state implementate in MatLab. L'obiettivo fondamentale del progetto consiste nella creazione di un algoritmo che renda possibile l'utilizzo della tecnica SVM in un contesto dinamico e unsupervised, classificando un'immagine in real-time senza passare per una fase di training ogni qualvolta l'immagine tenda a cambiare leggermente, aumentando così la probabilità di non perdere il target che si sta inseguendo.

1 Introduzione

Negli ultimi anni si sta sviluppando molto l'uso di algoritmi di machine learning applicati alla videosorveglianza per individuare ed inseguire oggetti o movimenti anomali in ambienti di vario tipo. Il lavoro qui descritto nasce dall'esigenza di compiere visual tracking intelligente di oggetti che possono cambiare le loro caratteristiche (colore o forma) mentre sono osservati. Questa ricerca è motivata dal fatto che in molte situazioni il target da individuare e seguire all'interno di un ambiente può cambiare conformazione o più semplicemente essere soggetto a variazioni di esposizione di luce e colori che la maggior parte delle volte rendono il tracking difficoltoso, in quanto rischiano di cambiare notevolmente le caratteristiche inizialmente scelte per individuare l'oggetto in ogni frame e di far confondere l'oggetto da inseguire con il background. Per questo motivo si considera solitamente il problema del visual tracking come un problema di classificazione binaria dove le due classi sono rispettivamente l'oggetto da inseguire (foreground) e l'ambiente circostante (background).

I problemi che si riscontrano solitamente in questo genere di applicazioni in ordine di importanza sono:

- **Carico computazionale**
  Avendo a che fare con immagini, molte volte anche a colori, si ha l'esigenza di elaborare frames con quantità d'informazione molto numerose (basti pensare che un'immagine 128 per 96 in scala RGB può essere rappresentata in una matrice di 36.864 elementi). Inoltre il più delle volte è necessario che questi frames vengano elaborati in modo sequenziale e on-line quindi il tempo di elaborazione deve necessariamente essere breve.

- **Metodi di selezione delle features e pattern**
  Quando si costruisce un algoritmo di classificazione per prima cosa si devono stabilire i criteri con cui osservare l'immagine che viene analizzata. Le features sono appunto le caratteristiche dell'immagine (scala RGB, media RGB, scala di grigi, pattern e sequenze) che si prendono come riferimento per l'analisi e la classificazione. La scelta di queste features infatti dipende da molti fattori (variabilità

colore, forma oggetto, movimento, ecc...).

- **Adattabilità classificatore**
  Il classificatore che si costruisce va necessariamente aggiornato affinchè sia sensibile anche alle possibili variazioni del target da inseguire. Qui si pone un problema non banale in quanto, soprattutto nell'unsupervised learning, è necessario avere un controllo adeguato sull'iperpiano che cambiando conformazione rischia di perdere il modello iniziale dell'oggetto che si desiderava inseguire.

Gli algoritmi SVM usati attualmente per la classificazione e il riconoscimento sfruttano la fase iniziale di training per creare l'iperpiano decisionale attraverso un database dato. Questo iperpiano una volta utilizzato resta invariato per la classificazione delle immagini destinate al test [**?**]. Ovviamente l'iperpiano una volta creato nel training resta invariato per tutte le classificazioni successive.
Altre applicazioni di SVM per il tracking hanno sfruttato il filtro di Kalman usando come features le scale di colori e la posizione dell'oggetto nell'immagine visualizzata. Il filtro di Kalman viene usato per tenere traccia della dinamica dell'oggetto riconosciuto e quindi predire la nuova posizione di questo nel frame successivo e aggiornare i nuovi support vectors [**?**].

Il lavoro qui presentato si concentra soprattutto sulla costruzione di un algoritmo che, senza l'uso di training ulteriori, aggiorna l'iperpiano decisionale attraverso un'approssimazione quadratica della stessa funzione di costo utilizzata per il training iniziale. Questa funzione viene utilizzata per trovare ad ogni frame l'iperpiano ottimo, rispetto ai punti del nuovo frame, dopo una precedente classificazione di questi mediante l'iperpiano calcolato al passo precedente. La dinamica dell'iperpiano viene rappresentata attraverso un modello gaussiano che influisce nella ricerca del nuovo piano ad ogni iterazione.
Questo approccio si rivela molto conveniente perchè risolve il problema dell'onere computazionale di ricerca dell'iperpiano ottimo, infatti la minimizzazione della funzione quadratica approssimata è molto meno dispendiosa di un nuovo training e, cosa interessante, tiene traccia della storia passata del modello perchè si basa sulle classificazioni precedenti. Inoltre nella minimizzazione interviene la dinamica del prior gaussiano.
L'algoritmo è stato costruito inizialmente sfruttando un modello fatto ad-hoc di punti generati in modo casuale così da poter osservare le dinamiche e capire eventuali errori. Tuttavia l'algoritmo creato è in grado di lavorare su immagini anche di dimensioni molto elevate a differenza degli algoritmi già presenti in Matlab che hanno sempre un limite di memoria e che risultano inutilizzabili su immagini vere.

2 TEORIA DI BASE

2.1 Machine Learning

Il Machine Learning (noto anche come Apprendimento Automatico), sotto-area fondamentale dell'Intelligenza Artificiale, è una disciplina scientifica che si occupa dell'implementazione di algoritmi che permettono ai calcolatori (learners) di sviluppare delle decisioni intelligenti (behaviours) e di riconoscere in maniera automatica modelli complessi (Pattern Recognition) in base a dati empirici.

Più in dettaglio, i dati disponibili hanno il compito di illustrare le relazioni tra le variabili d'interesse, mentre ciò che viene richiesto al learner è sfruttare l'informazione fornita da dati stessi, al fine di rilevare le caratteristiche più importanti della loro distribuzione di probabilità.

La difficoltà dell'approccio risiede nella "generalizzazione delle decisioni da intraprendere a partire dall'esperienza"; l'obiettivo principale è infatti quello di sviluppare delle decisioni intelligenti anche in situazioni mai osservate prima (test data) a partire da una quantità piuttosto ridotta di casi noti a priori (training data).

Esistono in letteratura diversi tipi di algoritmi di Machine Learning, i più popolari sono:

- Apprendimento Supervisionato (Supervised Learning);
- Apprendimento non-Supervisionato (Unsupervised Learning);
- Apprendimento con rinforzo (Reinforcement Learning)

Nel seguito saranno trattati brevemente solo i primi due.

2.2 Classificazione

Nell'ambito del Machine Learning e del Patterns Recognition, con il termine "Classificazione" ci si riferisce ad una procedura algoritmica che assegna ogni nuovo dato in ingresso (istanza) ad una delle categorie possibili (classi).

Se le classi di distinzione sono solo due, si parla di **classificazione binaria**, altrimenti si parla di **classificazione multi-classe**.

Più in dettaglio, ad ogni categoria corrisponde una etichetta diversa (label); l'algoritmo affida ad ogni istanza una label che indica semplicemente a quale classe appartiene il dato.

Una procedura in grado di esplicare tale funzione è denominata comunemente **classificatore**.

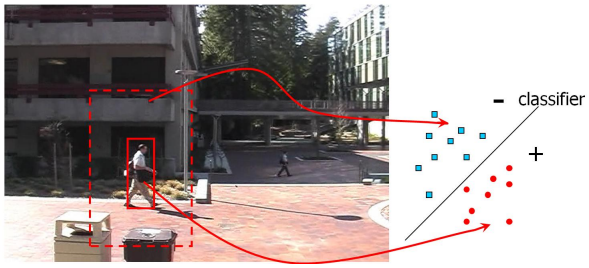

Fig. 1: esempio tipico di classificazione binaria, riconoscimento del foreground dal background

L'istanza è formalmente descritta attraverso un vettore di caratteristiche (features) che insieme costituiscono una descrizione globale, ma al contempo riassuntiva, del dato da classificare.

Esistono numerosi tipi di features utilizzabili, nella pratica quelle più comunemente adottate sono: valori interi, valori reali, features nominali o categoriche, ordinali.

2.3 Supervised - Unsupervised Learning

La classificazione normalmente si riferisce al riconoscimento supervisionato (Supervised Learning), ovvero ad una procedura che impara a classificare nuove istanze basandosi sul cosiddetto **Training Set**.

Quest'ultimo non è altro che un insieme di esempi (examples) scelti a priori; ogni example è una coppia formata da un'istanza e dalla corrispondente label, correttamente assegnata dall'operatore.

La fase di istruzione dell'algoritmo per l'ottenimento del classificatore desiderato, in base al training set, è denominata fase di Training.

I passi fondamentali da seguire per risolvere un problema di riconoscimento supervisionato sono:

- decisione del tipo di training examples al fine di rappresentare al meglio il contesto reale e applicativo su cui si compirà la classificazione;
- rappresentazione della funzione di decisione nello spazio delle features;
- scelta della struttura del classificatore e del corrispondente algoritmo implementativo (SVM, Decision Trees, AdaBoost ecc);
- prova dell'algoritmo ottenuto sul training set e taratura di eventuali parametri di controllo tramite validazione;
- valutazione dell'accuratezza e delle prestazioni del classificatore mediante applicazione ad un insieme di nuove istanze (**Test set**).

L'Unsupervised Learning viene invece utilizzato nei casi in cui non si hanno a disposizione le labels per la fase di training.

In questo caso non si parla più di Classificazione ma di Clustering, ovvero suddivisione degli examples in sottoinsiemi disgiunti tali che gli esempi di uno stesso gruppo risultino molto simili tra loro, secondo i criteri posti dall'operatore.

Questo tipo di apprendimento presenta risultati peggiori rispetto al caso supervised, d'altra parte è l'unico a poter essere utilizzato in real time e a non richiedere necessariamente l'intervento dell'operatore umano, sia nella fase di training sia in occasione di classificazioni errate da parte dell'algoritmo, che andrebbe quindi corretto tramite un nuovo allenamento.

Il tipo di apprendimento utilizzato nella parte implementativa di questo progetto si presenta come un ibrido tra i due appena presentati, dato che riunisce i vantaggi del lavoro in real time, caratterizzante l'Unsupervised Learning, al periodo di training tipico del Supervised Learning, per il conseguimento di performance più elevate.

2.4 Support Vector Machine

Tra le motivazioni che hanno portato alla scelta della tecnica di classificazione SVM, le principali sono:

- solida teoria di base;
- a differenza dei metodi di pattern recognition focalizzati sulla minimizzazione del rischio empirico (e quindi basati sulla minimizzazione degli errori di classificazione nel solo training set) l'SVM si propone di minimizzare il rischio strutturale, ossia la probabilità di classificare in maniera errata anche i nuovi dati in ingresso, oltre a quelli di allenamento;

- l'SVM condensa tutta l'informazione contenuta nel training set in pochi punti essenziali per il learning, alleggerendo così il carico computazionale dell'algoritmo implementativo.

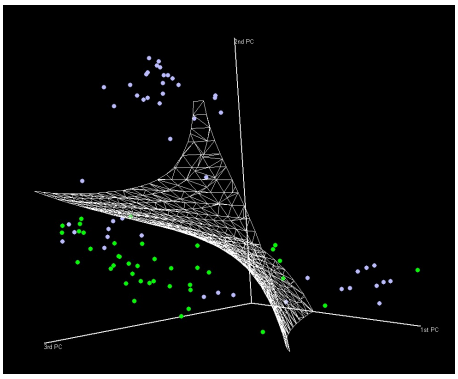

Fig. 2: esempio di classificazione binaria SVM in $\mathbb{R}^3$

Senza perdita di generalità si considererà d'ora in avanti solo il problema di classificazione binaria, in un contesto di tipo supervisionato.

Saranno trattati inoltre separatamente il caso di dati linearmente separabili e la generalizzazione al caso non linearmente separabile, entrambi risolti mediante tecnica SVM lineare.

2.4.1 *Linear SVM: Caso Linearmente Separabile*

Sia dato il Training Set T, composto dai punti $\mathbf{x_i} \in \mathbb{R}^{n_f}$, spazio delle features, $\forall i = 1, 2, \ldots, N_p$.

Ogni punto $\mathbf{x_i}$ è etichettato mediante una label $y_i \in \{-1, 1\}$, a seconda dell'appartenenza ad una delle due classi.

L'obiettivo della classificazione è ricavare l'equazione dell'iperpiano (funzione decisionale lineare) che divida l'insieme T, lasciando tutti i punti afferenti alla stessa classe dalla stessa parte dello spazio delle features e che contemporaneamente massimizzi la distanza tra le classi e l'iperpiano stesso (iperpiano ottimo di separazione).

Il training set T si dice Linearmente Separabile se

$$\exists\ \boldsymbol{\beta} \in \mathbb{R}^{n_f} \quad e \quad \beta_0 \in \mathbb{R} \quad \text{t.c.}$$

$$y_i(\boldsymbol{\beta} \cdot \mathbf{x_i} + \beta_0) \geq 1, \quad \forall i = 1, \ldots, N_p. \tag{1}$$

La coppia $(\boldsymbol{\beta},\ \beta_0)$ definisce un iperpiano di equazione

$$f(\mathbf{x}) = \boldsymbol{\beta} \cdot \mathbf{x} + \beta_0 = 0 \tag{2}$$

denominato iperpiano di separazione.

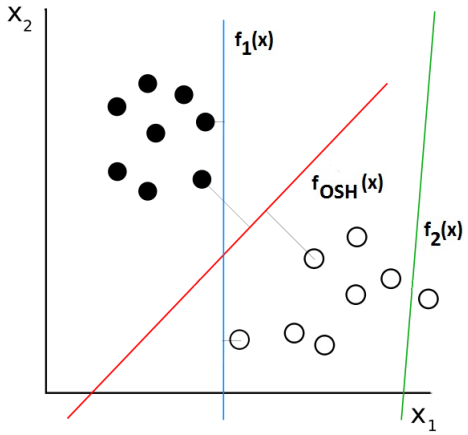

Fig. 3: spazio delle features ($\mathbb{R}^2$) con vari iperpiani di separazione, la decisione ottima è in rosso.

Ponendo $\beta = \| \boldsymbol{\beta} \|$, la distanza con segno d_i del punto $\mathbf{x_i}$ dall'iperpiano (**??**) è data da

$$d_i = \frac{\boldsymbol{\beta} \cdot \mathbf{x_i} + \beta_0}{\beta} \tag{3}$$

Dalle relazioni (**??**) e (**??**) si ottiene

$$y_i d_i \geq \frac{1}{\beta}, \quad \forall\ i = 1, \ldots, N_p \tag{4}$$

Si nota dunque che $1/\beta$ è il limite inferiore per la distanza tra i punti $\mathbf{x_i}$ e l'iperpiano di separazione.

È importante a questo punto introdurre la nozione di rappresentazione parametrica dell'iperpiano di separazione.

Dato un iperpiano di separazione identificato dalla coppia $(\boldsymbol{\beta},\ \beta_0)$ per il training set T linearmente separabile, la rappresentazione canonica dell'iperpiano si ottiene riscalando la coppia $(\boldsymbol{\beta},\ \beta_0)$ nella coppia $(\boldsymbol{\beta}',\ \beta_0')$, in modo che la distanza dei punti $\mathbf{x_j}$ più vicini all'iperpiano sia esattamente $1/\beta'$, ovvero in modo che

$$\min_{\mathbf{x_j} \in T} \{y_i(\boldsymbol{\beta}' \cdot \mathbf{x_j} + \beta_0')\} = 1. \tag{5}$$

Per semplicità, d'ora in poi assumeremo sempre l'iperpiano di separazione già in forma canonica, ovvero $\boldsymbol{\beta} = \boldsymbol{\beta}'$ e $\beta_0 = \beta_0'$.

Dato un T linearmente separabile, l'obiettivo di questa trattazione teorica è quindi ricavare l'equazione dell'OSH (Optimal Separating Hyperplane), ovvero l'iperpiano di separazione che massimizza la distanza dei punti di T a lui più vicini (si vedano la figure (**??**) e (**??**)).

In altre parole l'iperpiano ottimo, identificato dalla coppia $(\bar{\boldsymbol{\beta}},\ \bar{\beta}_0)$, è la soluzione del problema vincolato seguente:

$$\begin{aligned} &\text{Minimizzare} && \tfrac{1}{2} \parallel \boldsymbol{\beta} \parallel^2 \\ &\text{soggetto a} && y_i(\boldsymbol{\beta} \cdot \mathbf{x_i} + \beta_0) \geq 1, \quad \forall i \end{aligned}$$

detto Problema Primale.

La quantità $2/\beta$ è detta margine, essendo il limite inferiore della minima distanza tra i punti di classi diverse (si veda la figura).

Questa quantità rappresenta approssimativamente il grado di difficoltà del problema; più il margine è stretto e più il problema è difficile da risolvere.

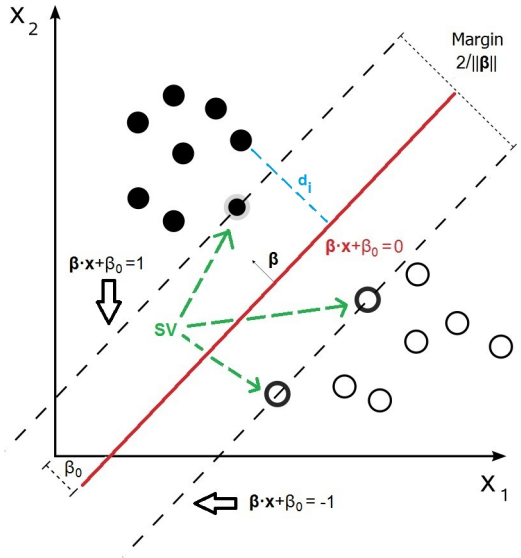

Fig. 4: Iperpiano ottimo e margine relativo in $\mathbb{R}^2$, caso linearmente separabile

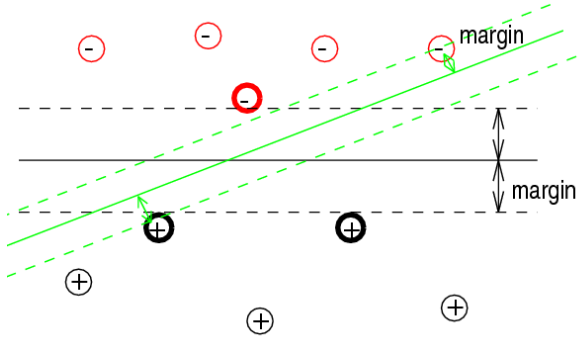

Fig. 5: l'iperpiano ottimo di decisione è associato al margine più ampio

Il problema primale si risolve solitamente con il Metodo dei Moltiplicatori di Lagrange.

Denotando con $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_{N_p})$ il vettore degli N moltiplicatori di Lagrange associati ai vincoli dati da (**??**), risolvere il problema primale equivale a trovare il punto di sella della funzione lagrangiana

$$L(\boldsymbol{\beta},\ \beta_0,\ \boldsymbol{\alpha}) = \frac{1}{2} \parallel \boldsymbol{\beta} \parallel^2 - \sum_{i=1}^{N_p} \alpha_i \{y_i(\boldsymbol{\beta} \cdot \mathbf{x_i} + \beta_0) - 1\} \tag{6}$$

Nel punto di sella, L presenta un minimo in $\boldsymbol{\beta} = \bar{\boldsymbol{\beta}}$ e $\beta_0 = \bar{\beta}_0$ ed un massimo in $\boldsymbol{\alpha} = \bar{\boldsymbol{\alpha}}$. Dunque:

$$\frac{\partial L(\boldsymbol{\beta},\ \beta_0,\ \boldsymbol{\alpha})}{\partial \beta_0} = \sum_{i=1}^{N_p} y_i \alpha_i = 0 \tag{7}$$

$$\frac{\partial L(\boldsymbol{\beta},\ \beta_0,\ \boldsymbol{\alpha})}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta} - \sum_{i=1}^{N_p} y_i \alpha_i \boldsymbol{x_i} = 0 \tag{8}$$

Una volta sostituite le equazioni (**??**) e (**??**) nella (**??**), si nota più chiaramente che risolvere il problema primale equivale a risolvere il seguente problema vincolato, detto Problema Duale:

$$\begin{aligned} &\text{Minimizzare} && -\tfrac{1}{2}\boldsymbol{\alpha}^T D \boldsymbol{\alpha} + \textstyle\sum_{i=1}^{N_p} \alpha_i \\ &\text{soggetto a} && \textstyle\sum_{i=1}^{N_p} y_i \alpha_i = 0 \\ & && \alpha_i \geq 0 \quad \forall i \end{aligned}$$

dove $D \in \mathbb{R}^{(N_p\ x\ N_p)}$ e t.c. $[D]_{ij} = y_i y_j \mathbf{x_i} \cdot \mathbf{x_j}$.

Dalla (**??**) si vede immediatamente che

$$\bar{\boldsymbol{\beta}} = \sum_{i=1}^{N_p} \bar{\alpha}_i y_i \mathbf{x_i}, \tag{9}$$

mentre $\bar{\beta}_0$ si può determinare da $\bar{\boldsymbol{\alpha}}$, soluzione del problema duale, e dalle condizioni di Karush-Kühn-Tucker (KKT)

$$\bar{\alpha}_i \{y_i(\bar{\boldsymbol{\beta}} \cdot \mathbf{x_i} + \bar{\beta}_0) - 1\} = 0, \qquad i = 1, \ldots, N_p. \tag{10}$$

È molto importante osservare che gli unici $\bar{\alpha}_i$ non-nulli nella (**??**) sono quelli per cui i vincoli (**??**) sono soddisfatti con l'eguaglianza. Conseguenza di ciò è che il vettore ottimo $\bar{\boldsymbol{\beta}}$ è una combinazione lineare di un numero relativamente basso dei punti $\mathbf{x_i}$. Questi punti sono chiamati Support Vector (SV), essendo i punti più vicini all'iperpiano ottimo di separazione ed anche gli unici punti del training set T indispensabili a determinare l'OSH stesso (si rimanda alla figura (**??**)). Si può quindi dire che essi condensano al loro interno tutta l'informazione contenuta in T.

Dati due qualunque SV, $\mathbf{x_r}$ e $\mathbf{x_s}$, tali che $y_r = 1,\ y_s = -1$ e $\bar{\alpha}_s,\ \bar{\alpha}_r \geq 0$, il parametro $\bar{\beta}_0$ si può ottenere nel modo seguente:

$$\bar{\beta}_0 = -\frac{1}{2}\bar{\boldsymbol{\beta}} \cdot (\mathbf{x_r} + \mathbf{x_s}) \tag{11}$$

Dunque il problema di classificazione di un nuovo dato, $\mathbf{x}$, si riduce all'osservazione del segno di

$$f_{OSH}(\mathbf{x}) = \bar{\boldsymbol{\beta}} \cdot \mathbf{x} + \bar{\beta}_0$$

2.4.2 *Linear SVM: Caso Non-linearmente Separabile*

Il caso di classificazione con training set non linearmente separabile si può affrontare generalizzando quanto visto fin'ora.

Si introducono, infatti, N_p varibili non-negative $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_{N_p})$ in modo che

$$y_i(\boldsymbol{\beta} \cdot \mathbf{x_i} + \beta_0) \geq 1 - \xi_i, \qquad i = 1, 2, \ldots, N_p \tag{12}$$

L'inserimento di queste variabili di slack permette di considerare anche possibili classificazioni errate.

L'iperpiano ottimo di separazione è allora la soluzione del Problema Primale generalizzato:

$$\begin{aligned} &\text{Minimizzare} && -\tfrac{1}{2} \parallel \boldsymbol{\beta} \parallel^2 + \gamma \textstyle\sum_{i=1}^{N_p} \xi_i \\ &\text{soggetto a} && y_i(\boldsymbol{\beta} \cdot \mathbf{x_i} + \beta_0) \geq 1 - \xi_i,\ \forall i \\ & && \xi_i \geq 0, \quad \forall i \end{aligned}$$

Il termine aggiuntivo $\gamma \sum_{i=1}^{N_p} \xi_i$ rende l'OSH meno sensibile all'eventuale presenza di outliers nel training set, mentre il parametro γ si può interpretare come un parametro di regolarizzazione. L'iperpiano ottimo, infatti, tende a massimizzare il margine per bassi valori di γ e a minimizzare il numero di classificazioni errate per valori di γ elevati.

Anche in questo caso si può trasformare il problema primale nel corrispondente Problema Duale generalizzato:

$$\begin{aligned} &\text{Minimizzare} && -\tfrac{1}{2}\boldsymbol{\alpha}^T D \boldsymbol{\alpha} + \textstyle\sum_{i=1}^{N_p} \alpha_i \\ &\text{soggetto a} && \textstyle\sum_{i=1}^{N_p} y_i \alpha_i = 0 \\ & && 0 \leq \alpha_i \leq \gamma, \quad \forall i \end{aligned}$$

con D analoga a quella del caso linearmente separabile.

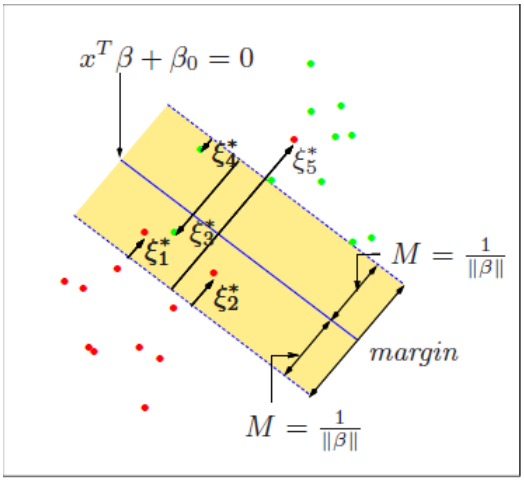

Fig. 6: iperpiano ottimo di decisione per training set non linearmente separabile

Dalla soluzione ottima $\bar{\boldsymbol{\alpha}}$ del problema duale generalizzato, si ricava facilmente

$$\bar{\boldsymbol{\beta}} = \sum_{i=1}^{N_p} \bar{\alpha}_i y_i \mathbf{x_i}, \tag{13}$$

mentre $\bar{\beta}_0$ si può determinare dalle condizioni di KKT:

$$\bar{\alpha}_i\{y_i(\bar{\boldsymbol{\beta}} \cdot \mathbf{x_i} + \beta_0) - 1 + \bar{\xi}_i\} = 0\ \forall i \tag{14}$$

$$(\gamma - \bar{\alpha}_i)\bar{\xi}_i = 0\ \forall i \tag{15}$$

dove $\bar{\xi}$ è il valore di ξ calcolato nel punto di sella.

Come in precedenza, i punti $\mathbf{x_i}$ a cui corrispondono $\bar{\alpha}_i \geq 0$ sono chiamati Support Vector.

In questo contesto bisogna fare una distinzione ulteriore tra i SV corrispondenti ad $\bar{\alpha}_i < \gamma$ e quelli per cui $\bar{\alpha}_i = \gamma$.

Nel primo caso, dalla condizione (**??**) segue che $\bar{\xi}_i = 0$, e dunque, dalla (**??**), che i support vectors giacciono esattamente sul margine (Margin Vectors).

Invece i SV relativi ad $\bar{\alpha}_i = \gamma$ (chiamati generalmente errori) sono punti:

- classificati non correttamente, se $\xi_i > 1$;
- classificati correttamente ma all'interno del margine dell'OSH
  se $0 < \xi_i < 1$;
- Margin Vectors, se $\xi_i = 0$.

Tutti i punti che non sono SV sono classificati correttamente e giacciono al di fuori del margine.

2.5 Ulteriore sviluppo e approssimazione

Il problema di classificazione fin'ora affrontato si può equivalentemente riformulare utilizzando una

funzione di penalità (detta Loss Function):

$$\sum_{i=1}^{N_p} [1 - y_i f(\mathbf{x_i})]_+ + \lambda \parallel \boldsymbol{\beta} \parallel^2 \tag{16}$$

dove $L(y_i, f(x_i)) := [1 - y_i f(\mathbf{x_i})]_+$ è la Hinge Loss Function, che pesa l'errore compiuto sulla classificazione di ogni punto x_i, mentre il termine quadratico finale è un termine di regolarizzazione (penalità quadratica) e $\lambda = \frac{1}{\gamma}$.

Il contributo apportato da questo progetto allo stato dell'arte consiste nell'implementazione di un algoritmo SVM basato sul Sequential Quadratic Programming (SQP). L'algoritmo SQP qui utilizzato può essere considerato come una versione semplificata del filtro di Kalman iterato (IKF), dato che non necessita di un modello di misura probabilistico $p(y|x)$, di difficile realizzazione in questo contesto.

Per quanto appena riportato, è necessario svolgere una linearizzazione del primo addendo della funzione (**??**) fino al secondo ordine, dato che quest'ultimo non è una quadratica in $\boldsymbol{\beta}$ e β_0. Occorre però, per prima cosa, rendere questo addendo differenziabile; per questo si è deciso di approssimare la Hinge Loss function con la Binomial Deviance:

$$L(y_i, f(\mathbf{x_i})) = log[1 + e^{-y_i f(\mathbf{x_i})}] \tag{17}$$

e quindi linearizzare il nuovo funzionale di costo:

$$\sum_{i=1}^{N_p} log[1 + e^{-y_i f(\mathbf{x_i})}] + \lambda \parallel \boldsymbol{\beta} \parallel^2 \tag{18}$$

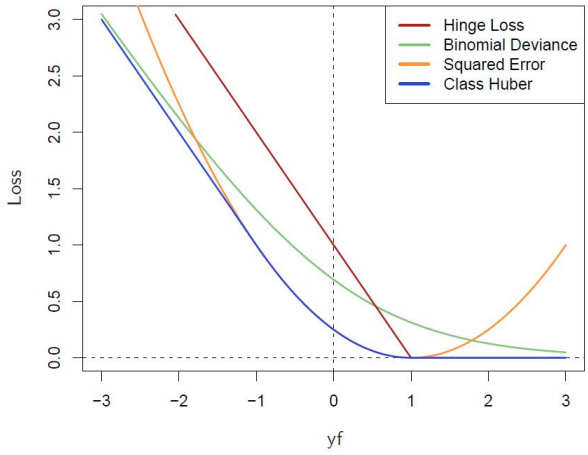

Fig. 7: funzione di penalità Hinge Loss e sua approssimazione differenziabile Binomial Deviance

Ciò che rimane da inserire, a questo punto, è la dinamica dell'iperpiano ottimo di separazione, in modo da trasformare il problema di ottimizzazione da statico in dinamico, ovvero da risolvere ad ogni istante di tempo t:

$$\min_{\boldsymbol{\beta}_t,\ \beta_{0_t}} \sum_{i=1}^{N_p} log[1 + e^{-y_i f(\mathbf{x_i})}] + \lambda \parallel \boldsymbol{\beta_t} \parallel^2 \tag{19}$$

Il modello non-lineare utilizzato per rappresentare la dinamica su $\boldsymbol{\beta}$ è il modello random walk seguente:

$$\begin{bmatrix} \boldsymbol{\beta}_{t+1} \\ \beta_{0_{t+1}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\beta}_t \\ \beta_{0_t} \end{bmatrix} + \begin{bmatrix} w_t \\ w_{0_t} \end{bmatrix} \tag{20}$$

dove $w_t \in \mathbb{R}^{n_f}$ e $w_{0_t} \in \mathbb{R}$ sono rumori gaussiani bianchi tra loro scorrelati e

$$Q = \begin{bmatrix} Var[w_t] & \mathbf{0} \\ \mathbf{0} & var[w_{0_t}] \end{bmatrix}$$

A questo punto si può procedere alla linearizzazione del primo addendo di (**??**) $\forall t$:

$$J(\boldsymbol{\beta}_t,\ \beta_{0_t},\ \mathbf{x},\ \mathbf{y}) := \sum_{i=1}^{N_p} log[1 + e^{-y_i f(\mathbf{x_i})}]$$

con $\mathbf{x} = \begin{bmatrix} \mathbf{x_1}^T \\ \mathbf{x_2}^T \\ \vdots \\ \mathbf{x_{N_p}^T} \end{bmatrix} \in \mathbb{R}^{(N_p\ x\ n_f)}$, $\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N_p} \end{bmatrix} \in \mathbb{R}_p^N$, e

$f(\mathbf{x_i}) = \boldsymbol{\beta}^T \mathbf{x_i} + \beta_0$.

Si ha dunque che:

$$J(\boldsymbol{\beta}_t,\ \beta_{0_t},\ \mathbf{x},\ \mathbf{y}) \simeq J(\hat{\boldsymbol{\beta}}_{t-1},\ \hat{\beta}_{0_{t-1}}, \mathbf{x},\ \mathbf{y}) +$$

$$+ D|_{\begin{bmatrix} \hat{\boldsymbol{\beta}}_{t-1} \\ \hat{\beta}_{0_{t-1}} \end{bmatrix}} \mathbf{z}_t + \frac{1}{2} \mathbf{z}_t^T H|_{\begin{bmatrix} \hat{\boldsymbol{\beta}}_{\boldsymbol{t-1}} \\ \hat{\beta}_{0_{t-1}} \end{bmatrix}} \mathbf{z}_t + o(\parallel \mathbf{z}_t \parallel^2) \tag{21}$$

dove:

- $\mathbf{z}_t := \begin{bmatrix} \boldsymbol{\beta}_t \\ \beta_{0_t} \end{bmatrix} - \begin{bmatrix} \hat{\boldsymbol{\beta}}_{t-1} \\ \hat{\beta}_{0_{t-1}} \end{bmatrix}$;
- $\begin{bmatrix} \hat{\boldsymbol{\beta}}_{t-1} \\ \hat{\beta}_{0_{t-1}} \end{bmatrix} \in \mathbb{R}^{(n_f+1)}$ è il punto attorno al quale si effettua la linearizzazione ad ogni istante di tempo;
- $D = \sum_{i=1}^{N_p} \frac{(-y_i)e^{-y_i f(\mathbf{x_i})}}{[1+e^{-y_i f(\mathbf{x_i})}]} \begin{bmatrix} 1 & \mathbf{x_i}^T \end{bmatrix} \in \mathbb{R}^{1x(n_f+1)}$ è il

gradiente;

- $H = \sum_{i=1}^{N_p} \begin{bmatrix} A_i & B_i \\ B_i{}^T & C_i \end{bmatrix}$ è l'Hessiano, con:
  - $C_i = \frac{e^{-y_i f(\mathbf{x_i})}}{[1+e^{-y_i f(\mathbf{x_i})}]^2} \in \mathbb{R}$;
  - $[A_i]_{(j,k)} = (\mathbf{x_i}_j \ \mathbf{x_i}_k)\ C_i \in \mathbb{R}^{(n_f \ x \ n_f)}$;
  - $[B_i]_k = \mathbf{x_i}_k \ C_i \in \mathbb{R}^{n_f}$.

Successivamente si effettua una sostituzione del termine di regolarizzazione in (**??**)

$$\lambda \parallel \boldsymbol{\beta}_t \parallel^2 \implies \mathbf{z}_t^T \Sigma_{\boldsymbol{\beta}}^{-1} \mathbf{z}_t$$

con $\Sigma_{\boldsymbol{\beta}} = H^{-1} + Q$. Questa scelta è giustificata dalla linerizzazione effettuata e dalla scelta del prior su $\begin{bmatrix} \boldsymbol{\beta}_t & \beta_{0_t} \end{bmatrix}^T$.

La forma finale del funzionale che si intende minimizzare è quindi:

$$J_{FIN}(\boldsymbol{\beta}_t,\ \beta_{0_t},\ \mathbf{x},\ \mathbf{y}) := J(\hat{\boldsymbol{\beta}}_{t-1},\ \hat{\beta}_{0_{t-1}}, \mathbf{x},\ \mathbf{y}) +$$

$$+D|_{\begin{bmatrix} \hat{\boldsymbol{\beta}}_{t-1} \\ \hat{\beta}_{0_{t-1}} \end{bmatrix}} \mathbf{z}_t + \frac{1}{2}\mathbf{z}_t^T H|_{\begin{bmatrix} \hat{\boldsymbol{\beta}}_{\mathbf{t-1}} \\ \hat{\beta}_{0_{t-1}} \end{bmatrix}} \mathbf{z}_t + \mathbf{z}_t^T \Sigma_{\boldsymbol{\beta}}^{-1} \mathbf{z}_t \tag{22}$$

Ponendo a zero la derivata di (**??**) rispetto a $\begin{bmatrix} \boldsymbol{\beta}_t & \beta_{0_t} \end{bmatrix}^T$, si ottiene il punto di ottimo:

$$\begin{bmatrix} \hat{\boldsymbol{\beta}}_t \\ \hat{\beta}_{0_t} \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{\beta}}_{t-1} \\ \hat{\beta}_{0_{t-1}} \end{bmatrix} - [(H + \Sigma_{\boldsymbol{\beta}}^{-1})^{-1} D^T] \tag{23}$$

3 IMPLEMENTAZIONE

Per l'implementazione degli algoritmi esposti si è utilizzato Matlab. Questo programma non è necessariamente l'ambiente migliore per studiare la Computer Vision: esistono librerie C, per esempio openCV, più complete e molto veloci studiate appositamente per problemi di visione. Nonostante questo si è scelto di usare Matlab in quanto il fine del progetto non è tanto quello di elaborare immagini o video ma di costruire e verificare l'esattezza dell'algoritmo creato. Questo scopo può essere raggiunto infatti tramite l'uso di modelli semplici che, generalizzando le dinamiche delle immagini, consentono di studiare cosa accade durante le simulazioni e individuare facilmente eventuali errori.

3.1 Costruzione del modello e generazione dei punti

Per una prima simulazione dell'algoritmo si parte da un problema semplice in $\mathbb{R}^2$, nel quale si suppone quindi che lo spazio delle features sia di due dimensioni: Il vettore delle features $\mathbf{x_i} = [x\ y]'$ ($\mathbf{x_i} \in \mathbb{R}^2$) contiene le posizioni dei punti rispetto agli assi. Questa scelta è legata al fatto che la rappresentazione più semplice ed intuitiva dell'iperpiano può essere fatta solo in uno spazio di due dimensioni, in cui questo è rappresentato da una retta che divide, ad ogni passo, le due classi.
Il modello è stato costruito appositamente per studiare il caso linearmente separabile, quindi, come già spiegato precedentemente, si sono generati due gruppi di punti che compiono una traiettoria nello spazio restando divisi.
In dettaglio: si generano casualmente, all'interno di due aree prestabilite, N punti per ognuna delle due classi (nel nostro caso punti rossi e punti blu). Successivamente ad ogni punto viene assegnata una velocità random e la traiettoria di ognuno di questi viene costruita parametrizzando un moto lineare uniforme. Ovviamente punti della stessa classe hanno direzione uguale ma velocità diverse, quindi i due gruppi si muovono restando separati ma al loro interno variano le distanze tra i punti che li costituiscono. L'appartenenza dei campioni creati ad una delle due classi è implementata legando una terza componente ad ognuno di essi, la label, che distingue tra rosso e blu, a seconda che il suo valore sia 1 o -1.
Quindi, ad ogni passo, si genera un frame dove le distanze e le posizioni dei punti variano. È proprio su questi frames che si testano gli algoritmi creati cercando ad ogni istante l'iperpiano che divide le due classi in modo ottimale.

3.2 Algoritmo

Il file *fun_SQP.m* contiene la funzione implementata di cui si riporta più avanti una descrizione riassuntiva. Si consideri che l'algoritmo, nella fase iniziale, per la costruzione del primo iperpiano utilizza il training già implementato in un toolbox per Matlab chiamato *libsvm-mat-3.0.1* [**?**] (Questo pacchetto fin dalle prime simulazioni è risultato più efficiente di quello già presente in Matlab in quanto riesce a gestire quantità più grandi di dati ed è implementato in *C++* con evidente risparmio di onere computazionale).
Ripercorriamo per passi le varie fasi della funzione:

SQP dinamico:

Fase di Inizializzazione (solo per t = 1)

- Analisi di N_P punti per ognuna delle due classi;
- Calcolo dell'iperpiano iniziale $f(\mathbf{x}) = \boldsymbol{\beta} \cdot \mathbf{x} + \beta_0$ attraverso il metodo di Training di cui sopra;
- Impostazione della varianza Q del rumore del modello (**??**).

Fase di Ottimizzazione (istante t > 1)

- Analisi dei nuovi $2N_P$ punti lungo la traiettoria;
- Classificazione di questi attraverso l'iperpiano precedente $f(\mathbf{x}) = \boldsymbol{\beta}_{t-1} \cdot \mathbf{x} + \beta_{0_{t-1}}$;
- Ricerca del nuovo iperpiano ottimo rispetto ai nuovi punti (iterato più volte in base al numero di linearizzazioni N_{LIN} scelte):
  - Linearizzazione fino al 2° ordine della funzione di costo attorno a $\begin{bmatrix}\boldsymbol{\beta}_{t-1} \\ \beta_{0_{t-1}}\end{bmatrix}$ e calcolo delle matrici Hessiano e Gradiente;
  - Calcolo del $\begin{bmatrix}\boldsymbol{\beta}_t \\ \beta_{0_t}\end{bmatrix}$ e costruzione del nuovo iperpiano ottimizzato $f(\mathbf{x}) = \boldsymbol{\beta}_t \cdot \mathbf{x} + \beta_{0_t}$;

Come si vede la parte iniziale fa uso del training di *LibSVM* in modo da basare le fasi di ottimizzazione successive su una prima classificazione con un iperpiano giusto. Questa è una questione su cui conviene soffermarsi: l'algoritmo *fun_SQP*, infatti, ad ogni iterazione compie un'ottimizzazione che si basa sulla classificazione dei punti fatta con l'iperpiano precedente. Il training iniziale, quindi, si rende necessario in quanto, almeno al primo passo, si vuole essere certi che la classificazione sia corretta, onde evitare la ripercussione di errori di classificazione nella creazione degli iperpiani successivi.

Con un'applicazione diversa è stato creato anche l'algoritmo relativo al file *fun_SQP_static.m*. Sostanzialmente quest'ultimo ha il fine di usare l'ottimizzazione SQP per effettuare il training senza utilizzare i metodi già esistenti in letteratura. Più in dettaglio: si sceglie un data set di punti e si cerca l'iperpiano ottimo che li separa. Per fare ciò si estrae dal data set un gruppo di punti iniziali abbastanza consistente, utilizzato per un primo training, e, in seguito si estraggono ulteriori sottogruppi (di dimensioni scelte a piacere) per correggere l'iperpiano iniziale ottenuto. La questione cruciale di questo algoritmo è che in questo contesto l'ottimizzazione quadratica non viene mai utilizzata per fare classificazione: qui si è interessati solo ad effettuare il training in un modo nuovo cercando di diminuire l'onere computazionale derivante dai metodi classici. Il problema in analisi dunque non è dinamico, per questo motivo nel modello (**??**) imposteremo $Q = 0$ (proprio per questo viene chiamato statico).

SQP statico:

Fase di Inizializzazione (solo per t = 1)

- Scelta training set e del gruppo iniziale di punti da esaminare;
- Impostazione del $\begin{bmatrix}\boldsymbol{\beta}_t \\ \beta_{0_t}\end{bmatrix}$ iniziale (arbitrario);
- Calcolo del $\begin{bmatrix}\boldsymbol{\beta}_t \\ \beta_{0_t}\end{bmatrix}$ ottimo nuovo;

Fase di Ottimizzazione (istante t > 1)

- Analisi del nuovo gruppo di punti di numerosità N_G;
- Ricerca iperpiano ottimo rispetto a nuovi punti (iterato più volte in base al numero di linearizzazioni N_{LIN} scelte):
  - Linearizzazione fino al 2° ordine della funzione di costo attorno a $\begin{bmatrix}\boldsymbol{\beta}_{t-1} \\ \beta_{0_{t-1}}\end{bmatrix}$ e calcolo delle matrici Hessiano e Gradiente;
  - Calcolo del $\begin{bmatrix}\boldsymbol{\beta}_t \\ \beta_{0_t}\end{bmatrix}$ e costruzione del nuovo iperpiano ottimizzato $f(\mathbf{x}) = \boldsymbol{\beta}_t \cdot \mathbf{x} + \beta_{0_t}$;

Il fatto interessante è che il training set, dopo il primo passo, può essere suddiviso in gruppi più piccoli con il vantaggio di semplificare il problema di training, riducendo i campioni esaminati e quindi la complessità temporale ad ogni passo.
Come poi verrà dimostrato nelle simulazioni questo algoritmo è stato applicato anche all'intero training set senza dividerlo in sottogruppi più piccoli dando risultati soddisfacenti.
L'ultima funzione è stata testata anche su data set di punti non linearmente separabili e ha dato anche in questo caso dei risultati molto soddisfacenti che si avvicinano al caso ideale.

4 SIMULAZIONE

Fin dalle prime simulazioni si sono riscontrate alcune difficoltà di calcolo riguardanti l'inversa di

Σ_β che sono state superate sostituendo la matrice stessa con la sola diagonale. Questo accorgimento ha migliorato sensibilmente le prestazioni in entrambe le funzioni consentendo un'accuratezza maggiore nel calcolo dei valori di β ottimale.

4.1 Simulazione dell'algoritmo *SQP dinamico*

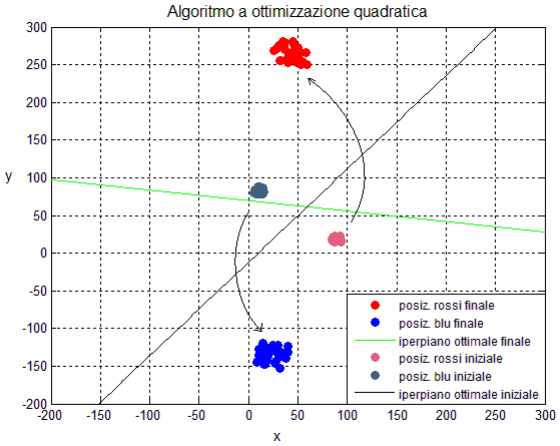

Fig. 8: Dinamica dei punti

In questa fase la dinamica dei punti generata ad ogni passo rispecchia il caso linearmente separabile dato che i due gruppi di punti non si sovrappongono mai durante il movimento. Per le varie prove qui presentate è stata scelta un'unica realizzazione del movimento su 30 frames in modo da rendere il confronto più efficace al variare dei parametri del modello. Gli unici gradi di libertà disponibili sono la varianza Q del rumore di modello presente in (**??**) e il numero di linearizzazioni che vengono eseguite ad ogni passo. Un punto di particolare importanza di questo studio riguarda la costruzione di indici di prestazione che, in questo contesto, non è stata affatto banale. Infatti i dati a disposizione più significativi sono le distanze punto-retta minime ottenute ad ogni passo con l'algoritmo SQP, confrontate con le distanze punto-retta del caso ideale, prese come riferimento (per distanza punto-retta si intende la distanza minima tra retta ottima calcolata e ognuno dei due gruppi, queste distanze vengono chiamate anche margine sinistro e destro).

Teoricamente margine sinistro e destro dovrebbero coincidere come si vede nel caso ideale, invece nell'SQP dinamico si riscontra una leggera differenza tra i due valori ad ogni passo. Questa differenza è giustificabile in quanto l'algoritmo costruito si basa su un'approssimazione quadratica che cerca il punto di minimo in modo

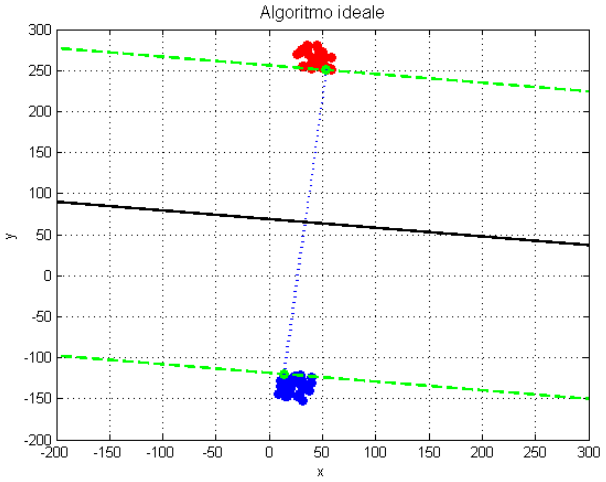

Fig. 9: Margini nel caso ideale

tanto accurato quante più sono le linearizzazioni. Si è quindi scelto di calcolare la media tra i due margini in modo da ricavare un valore unico da confrontare con il valore ideale calcolato con il metodo di training classico implementato nella *LibSVM* (nel caso ideale i valori dei due margini sono identici). Questo confronto consiste semplicemente nel calcolo dell'errore assoluto tra la media descritta e la distanza nel caso ideale. Di seguito si riportano le prove effettuate al variare dei parametri, ognuna con il rispettivo grafico dell'errore assoluto. Fissato un numero di linearizzazioni si testa l'algoritmo al variare di Q secondo i seguenti valori:

$$Q = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 100 \end{bmatrix}$$

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Q = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Q = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 1000 \end{bmatrix}$$

$$Q = \begin{bmatrix} 1000 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}$$

Da queste immagini si nota come aumenti la sensibilità alla variazione di Q nel momento in cui, nell'intervallo tra $10°$ e $20°$ frame, i due gruppi

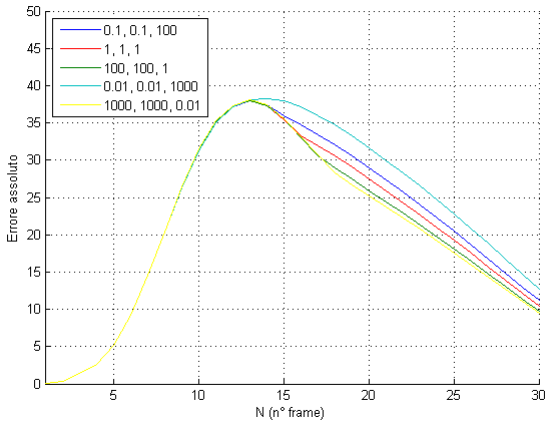

Fig. 10: Errore rispetto caso ideale con 5 linearizzazioni

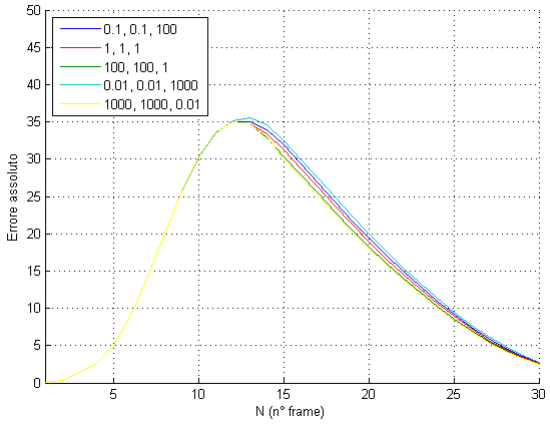

Fig. 11: Errore rispetto caso ideale con 10 linearizzazioni

di punti si avvicinano e la retta dell'iperpiano è costretta a variare in modo più brusco. Questo comportamento è normale in quanto il movimento dei due gruppi costringe la retta a muoversi in modo tanto veloce quanto più i due gruppi sono vicini. Si nota invece che più i due gruppi sono distanti tra loro e più gli errori rispetto al caso ideale si abbassano eguagliandosi. Questo dato è molto importante in quanto dimostra che l'algoritmo creato tende al caso ideale all'aumentare della distanza tra i due gruppi di punti. All'aumentare delle linearizzazioni si osserva come l'errore tenda a concentrarsi in un'unica curva rendendo la differenza di varianze nel modello poco influente. La spiegazione di questo comportamento è da rintracciare nel fatto che all'aumentare delle linearizzazioni l'ottimizzazione "dimentica" sempre di più l'iperpiano precedente costruendo quello (unico) che ottimizza le distanze in modo migliore nel passo corrente.

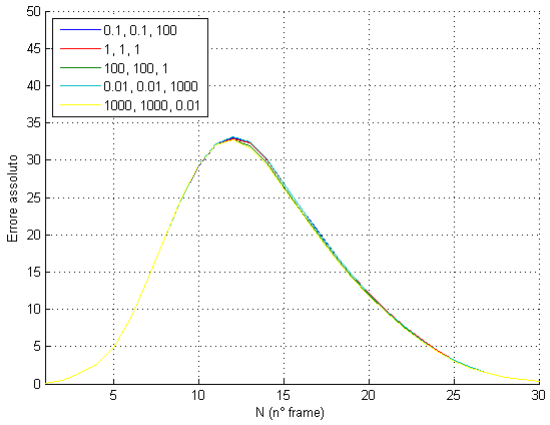

Fig. 12: Errore rispetto caso ideale con 20 linearizzazioni

4.2 Simulazione su data-set dell'algoritmo *SQP statico*

In questa fase si è costruito un data set di 1000 punti (500 per ognuno dei due gruppi) sui quali è stato effettuato il training in tre modalità differenti al fine di ottenere l'iperpiano decisionale ottimo da poter poi applicare ad un'eventuale classificazione. Le tre fasi sono:

- SVM train sulla totalità dei punti con metodo classico;
- SVM train sulla totalità dei punti con algoritmo SQP;
- SVM train con algoritmo SQP diviso in due parti: training iniziale con 100 punti per ogni gruppo e successivi training parziali con gruppi di punti in numero differente di volta in volta.

Lo stesso studio è stato effettuato nel caso di gruppi di punti linearmente separabili e non linearmente separabili.

4.2.1 Caso linearmente separabile

Si riportano di seguito i risultati ottenuti nelle tre fasi. Nel caso del Training parziale si considerano gruppi da 50 punti per i Training successivi al primo. Di questi si analizza solo l'ultimo effettuato in figura (**??**).

Il grafico in figura (**??**) è uno dei risultati fondamentali di questa ricerca. Infatti si vede come l'errore assoluto (calcolato confrontando il β del training classico e quello calcolato con il metodo SQP) diminuisce esponenzialmente all'aumentare

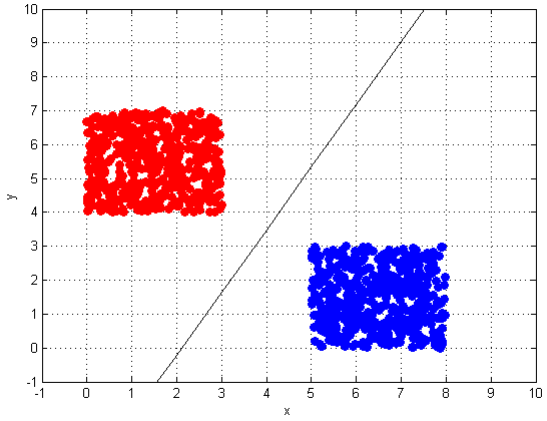

Fig. 13: Training con metodo SVM classico - caso linearmente separabile

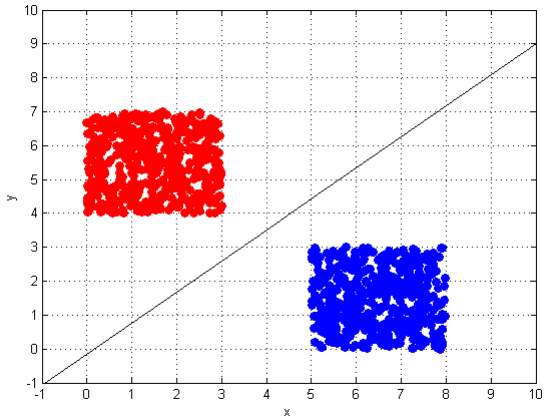

Fig. 14: Training con metodo SQP - caso linearmente separabile (20 linearizzazioni)

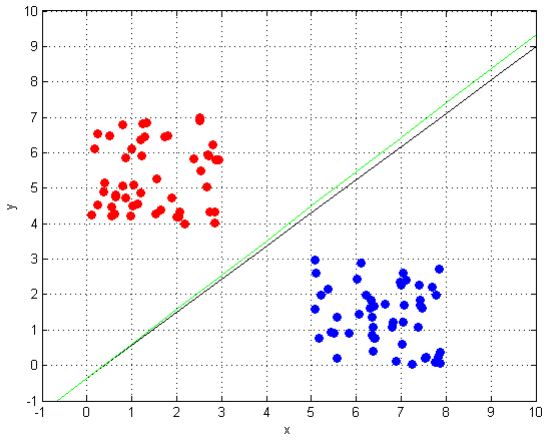

Fig. 15: Training con metodo SQP - caso linearmente separabile (20 linearizzazioni, in verde l'iperpiano aggiornato)

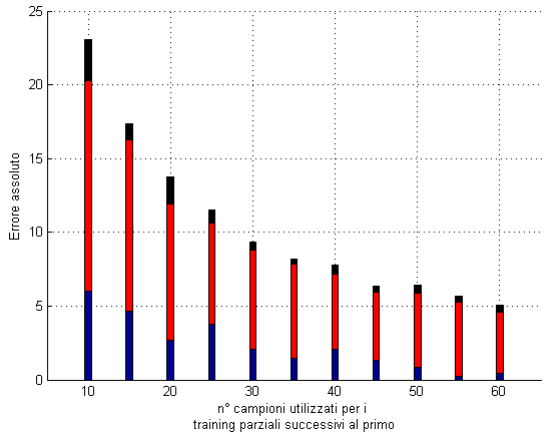

Fig. 16: Valore assoluto dell'errore dell'iperpiano ottimo calcolato al variare della numerosità dei gruppi utilizzati per i Training successivi al primo

del numero di punti in ogni gruppo di training successivo a quello iniziale. Da questo si deduce come il metodo qui descritto può essere usato per semplificare il problema di training su un insieme trovando però un giusto trade-off tra divisione in sottogruppi più piccoli ed efficienza come riportata sul grafico. Infatti l'errore diminuisce se il numero dei punti per ogni sottogruppo è sufficientemente alto. Si può dedurre dall'immagine che sopra i 40 punti si ottengono buoni risultati con un errore che tende a stabilizzarsi intorno al valore assoluto 5.
Riassumendo, ipotizzando di avere un training set di 1000 punti di due classi ed effettuando un primo training su di un test set di 100 punti per gruppo, la scelta ottimale di divisione della restante parte degli 800 elementi è compiuta considerando test set più piccoli di minimo 40 punti per gruppo. In questo modo l'iperpiano ottenuto con l'algoritmo SQP tende a coincidere con l'iperpiano costruito nel caso ideale in figura (**??**).

4.2.2 Caso non linearmente separabile

Si riportano di seguito i risultati ottenuti nelle tre fasi. Nel caso del Training parziale si considerano gruppi da 50 punti per i Training successivi al primo. Di questi si analizza solo l'ultimo effettuato in figura (**??**).

Confrontando la figura (**??**) con il caso ideale in (**??**) si può vedere che si ottengono buoni risultati, anzi, il caso non separabile sembra compiere meno errore dell'analogo separabile. La spiegazione a questo fatto si chiarisce osservando la dinamica dei vari frames: nel caso non linearmente separabile l'iperpiano ad ogni nuovo sottogruppo varia molto poco ed è più vincolato a stare al centro della regione di sovrapposizione dei punti.

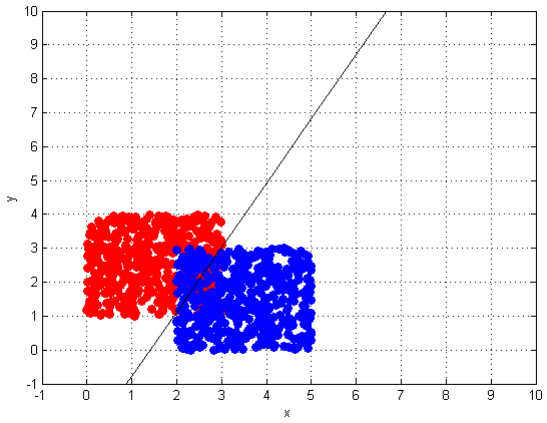

Fig. 17: Training con metodo SVM classico - caso non linearmente separabile

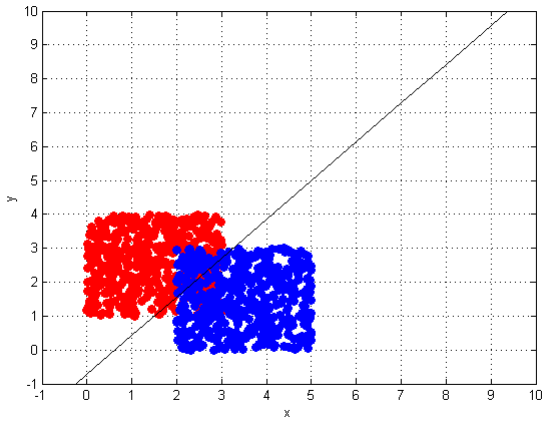

Fig. 18: Training con metodo SQP - caso non linearmente separabile (20 linearizzazioni)

4.3 Applicazione su video

Per dare un'idea delle applicazioni che possono essere sviluppate attraverso l'algoritmo SQP dinamico si presenta in questa sezione un'implementazione che fa uso di un video reale. Quest'ultimo consiste in una ripresa molto semplice nella quale è presente in primo piano una pallina bianca su sfondo praticamente nero. La pallina viene illuminata in modo graduale partendo quasi da un assenza di luce fino ad arrivare ad un'illuminazione molto forte e completa. Questo esempio è esemplificativo di una dinamica dei colori del soggetto da inseguire (la pallina) che varia nel tempo.

In questo esempio le *features* sono tre e consistono nei tre colori dell'immagine RGB (Nelle sezioni precedenti si è trattata la classificazione con sole due features). I frames usati sono di dimensione

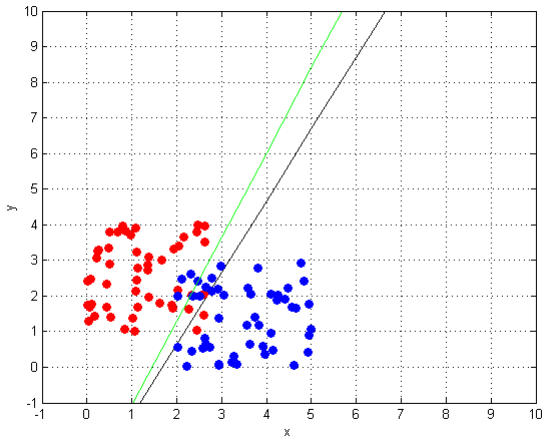

Fig. 19: Training con metodo SQP - caso non linearmente separabile (20 linearizzazioni, in verde l'iperpiano aggiornato)

128 per 96 pixel e ne sono stati selezionati 5 per secondo con un programma (*"Super"*) in modo da dare una certa gradualità agli eventuali cambiamenti di caratteristiche nell'immagine. L'iperpiano iniziale ottimo è stato costruito attraverso il training di *LibSVM* dando delle soglie molto basse per individuare un minimo di bianco nel primo frame. Qui sotto si riportano alcuni frames per dare un'idea della variabilità dell'immagine usata.

Fig. 20: Alcuni frames del video in esame

Di seguito si riportano alcuni frames in cui i punti riconosciuti dall'iperpiano vengono colorati di rosso (Con Matlab risulta difficile trovare un metodo che evidenzi meglio l'oggetto riconosciuto).

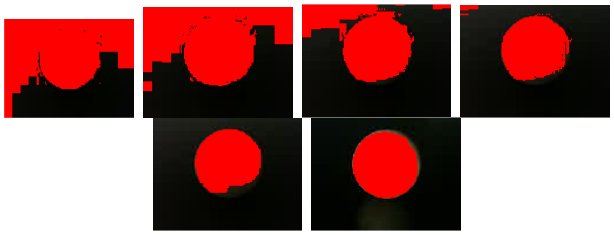

Fig. 21: Riconoscimento della pallina dei frames precedentemente riportati

Questo esempio dimostra come l'algoritmo creato si adatta dinamicamente alle immagini in modo da poter rendere possibile il riconoscimento della pallina anche se cambia di colore (passa da un grigio scuro ad un bianco molto intenso). In questa fase infatti l'iperpiano si sposta e viene ricalcolato ad ogni passo un nuovo β.

5 Conclusioni

Si riportano di seguito alcune considerazioni sul lavoro svolto.

5.1 Considerazioni su SQP dinamico

I risultati ottenuti in questo algoritmo sono da considerarsi soddisfacenti in quanto si è riusciti ad inserire la dinamica di un modello all'interno dell'ottimizzazione quadratica per la ricerca dell'iperpiano ottimo. Inoltre questo iperpiano ha delle buone capacità di **adattamento** alle variazioni dell'immagine ed è adatto ad un uso nei casi applicativi come dimostrato appena sopra nell'esempio della pallina.
Una considerazione importante riguarda il **modello** (**??**) utilizzato per rappresentare la dinamica dell'iperpiano, da considerarsi molto generico. È molto probabile che una scelta più accurata e calibrata del modello possa diventare più determinante nell'ottimizzazione rendendo la dinamica dell'iperpiano più adeguata al problema in esame. Testando l'algoritmo su alcuni video reali si è notato che il classificatore talvolta tende ad adattarsi a dinamiche dell'immagine che lo portano a modificarsi in modo sbagliato tanto da rendere inadeguato l'inseguimento al target inizialmente assegnato. La causa di questo comportamento è da rintracciare in una **assenza di controllo** da parte dell'algoritmo su alcune dinamiche del modello.

5.2 Considerazioni su SQP statico

I risultati ottenuti in questa fase del progetto si ritengono essere molto buoni e anche alternativi rispetto ai metodi tipici utilizzati per compiere il Training con metodo SVM. Infatti notiamo due importanti caratteristiche positive di questo algoritmo:

- La possibilità di **suddividere il training set** in piccoli sottogruppi molto utili se si utilizzano grandi moli di dati (algoritmo distribuito) e un conseguente carico computazionale minore.
- L'**efficienza** di questo algoritmo di Training è piuttosto elevata e si ottengono risultati molto vicini a quelli dell'algoritmo di Training SVM centralizzato (ovvero con l'utilizzo di tutti i campioni in una sola volta).

6 Sviluppi futuri

A partire dalle considerazioni appena fatte si sottolineano eventuali possibilità interessanti di ulteriore sviluppo:

- Sviluppo di un controllo adeguato alla correzione della dinamica dell'iperpiano;
- Possibile implementazione di una funzione di costo più sensibile ai cambiamenti dei punti esaminati
- Studio di un modello di misura adeguato per l'implementazione degli algoritmi creati attraverso la teoria del filtro di Kalman iterato.
- Uso del training implementato con il caso statico al posto training iniziale anche nel caso dinamico.

Ringraziamenti

Si ringraziano i Professori Pillonetto Gianluingi e Schenato Luca per il sostegno ed i suggerimenti indispensabili alla realizzazione di questo lavoro.

References

[1] M. Pontil and A. Verri, *Support Vector Machines for 3D Object Recognition*, vol.20, pp. 637-646, 1998.
[2] A. Garg, I. Cohen, T. S. Huang, *Adaptative Learning Algorithm for SVM Applied to Feature Tracking*, 1999.
[3] Chih-Chung Chang and Chih-Jen Lin, *LIBSVM - A Library for Support Vector Machines*, http://www.csie.ntu.edu.tw/∼cjlin/libsvm/.
[4] Trevor Hastie, Robert Tibshirani, Jerome Friedman, *The elements of statistical Learnings*, Second Edition.
[5] Bradley, M. Bell, Fredrick W. Cathey, *The iterated Kalman Filter Update as a Gauss Newton method*.
[6] Nando de Dreitas, Marta Milo, Philip Clarkson, Mahesan Niranjan, Andrew Gee, *Sequential support vector. machines*
[7] Shai Avidan, *Support Vector Tracking*.
[8] Shai Avidan, *Subset selection for efficient SVM tracking*.
[9] Corinna Cortes, Vladimir Vapnik, *Support Vector Networks*.
[10] Carlotta Domeniconi, Dimitrios Gunopulos, *Incremental Support Vectors Machines constraction*.
[11] Stephan Ruping, *Incremental Learning with support vector machines*.