Head and Shoulder Profile Detector

Ilario Pierbattista

11 luglio 2015

1 Panoramica generale

Si vuole sviluppare un software in grado di riconoscere in modo efficiente e quanto più accurato il profilo umano ripreso dall'alto dal sensore di profondità di un dispositivo Kinect v2.

1.1 Hardware e setup

Il sensore di profondità del Kinect ha una risoluzione di 16bit ed una precisione intorno ai 3mm (che diminuisce con l'aumentare della distanza). È montato al soffitto di una stanza a circa tre metri da terra, ha un campo visivo[1] di $70° \times 60°$, i quale, in relazione alla distanza dal pavimento, permette di catturare un'area di circa $4m \times 5m$.

Il Kinect cattura 30 frame ogni secondo. La dimensione di ogni frame è pari a 512×424 pixel.

1.2 Struttura software

Il software è diviso in due moduli: il modulo per il riconoscimento e il modulo di allenamento.

Nella prima fase, quella di allenamento, il relativo modulo software, a partire da un insieme di campioni classificati, genererà un insieme di regole per classificare campioni con le medesime caratteristiche. Implementa l'algoritmo Adaboost.

La seconda fase, quella di riconoscimento, lavora su dati reali: ad ogni frame acquisito tramite il sensore di profondità del Kinect si applicano le regole di classificazione generate nella fase precedente, stabilendo se nel frame è stata ripresa una persona e - in caso affermativo - la posizione di quest'ultima.

2 Adaboost

Adaboost è un *meta algoritmo* di *machine learning* che viene utilizzato per aumentare le prestazioni di altri algoritmi di apprendimento. Ciò avviene estraendo un *classificatore*

[1] FOV: `https://en.wikipedia.org/wiki/Field_of_view`

forte a partire da un insieme di *classificatori deboli*, dove per classificatore si intende una funzione $f(x)$ che identifichi la *classe* di appartenenza dell'oggetto x in input.

La realtà d'interesse del problema prevede l'esistenza di due classi di oggetti: *umano* e *non umano*.

I dati in input per la fase di allenamento sono costituiti dal *dataset di allenamento*, ovvero una raccolta di oggetti preclassificati attraverso i quali Adaboost riuscirà a selezionare una combinazione di classificatori deboli che meglio approssimano la classificazione degli elementi del dataset.

2.1 Il dataset dall'allenamento

La finestra di visualizzazione, come accennato in precedenza è di 512×424 pixel. In una registrazione che riprende dall'alto una persona che attraversa la stanza camminando, la figura della persona occupa una porzione della finestra di circa 160×100 pixel.

Un dataset di allenamento non è altro che un insieme di porzioni di frame di una certa dimensione (la stessa per tutti) che vegono classificati manualmente dal creatore[2] del dataset.

La creazione del dataset è un'attività delicata. I criteri di scelta vengono trattati nella sezione 3.

2.2 Preprocessing

Gli elementi del dataset verranno preprocessati prima di poter essere dei dati di input adeguati per l'algoritmo.

Da ricordare che stiamo trattando di matrici 160×100 (porzione del frame principale) i cui valori corrispondo al risultato della misurazione del sensore di profondità del Kinect ed è proporzionale alla distanza rilevata.

2.2.1 Conversione delle distanze

La prima operazione di preprocessing consiste nel trasformare tali distanze (sensore-superficie) in distanze dal pavimento della stanza. Come è facilmente intuibile

$$d' = d_{pavimento} - d$$

dove la distanza del pavimento dal sensore ($d_{pavimento}$). La distanza del pavimento può essere estratta dall'immagine stessa

Questa prima operazione di preprocessing ha complessità computazionale $\Theta(n \cdot m)$ con n ed m dimensioni della porzione del frame.

[2] Il sistema di apprendimento che si sta trattando ricade nella categoria dei sistemi di *apprendimento supervisionato*.

2.2.2 Immagine integrale

Costruire una funzione riesca a classificare un'immagine (la matrice dei valori del sensore di profondità è a tutti gli effetti considerabile come un'immagine) prevede l'utilizzo delle *Haar-like features*.

L'obbiettivo di una feature di Haar è quello di evidenziare le *differenze d'intensità* tra due regioni rettangolari adiacenti di un'immagine. Il valore di una semplice feature a due rettangoli corrisponde alla differenza tra la somma delle intensità di tutti i pixel della prima area e la somma delle intensità di tutti i pixel della seconda. Per intensità del pixel si intende il valore che il pixel assume. Inoltre si possono considerare aree partizionate in più di due rettangoli con una semplice generalizzazione del calcolo.

$$feature = Aree_{White} - Aree_{Black}$$

Saranno utilizzate solamente features a due e a tre rettangoli (bandate).

Il calcolo di una feature è un'operazione computazionalmente costosa. Si introduce quindi il concetto di *immagine integrale* (altrimenti nota come *summed area table*): ogni pixel dell'immagine integrale corrisponde alla somma dei valori di tutti i pixel della regione in alto a sinistra dell'immagine. Rigorosamente, sia $II(x, y)$ l'intensità del pixel alla posizione (x, y) dell'immagine integrale:

$$II(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} I(i, j)$$

Calcolare la somma delle intensità di tutti i pixel in un'area rettangolare attraverso l'immagine integrale diventa un'operazione molto veloce. Infatti:

$$\sum_{i=x_1}^{x_2} \sum_{j=y_1}^{y_2} I(i, j) = II(x_2, y_2) + II(x_1, y_1) - II(x_1, y_2) - II(x_2, y_1)$$

Grazie all'immagine integrale, una feature di qualsiasi dimensione può essere calcolata in un tempo costante.

2.3 Estrazione del classificatore forte

Sia $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ un insieme di n coppie costituite da un'immagine (x_i) e la relativa classificazione $(y_i \in \{0, 1\})$. Se $y_i = 1$, allora x_i appartiene alla classe *umano* (la coppia (x_i, y_i) prende il nome di *esempio positivo*), altrimenti appartiene alla classe *non umano* (la coppia (x_i, y_i) prende il nome di *esempio negativo*).

L'insieme D può essere partizionato come segue:

$$P = \{(x, y) \in D | y = 1\} \text{ e } N = \{(x, y) \in D | y = 0\}$$

Si introduce anche il concetto di *classificatore debole*. Si tratta di una funzione che per una data immagine x in ingresso, assume il valore che simboleggia la presunta classe

di appartenenza di quest'ultima. Nel dettaglio:

$$h(x) = \begin{cases} 1 & \text{se } pf(x) < p\theta \\ 0 & \text{altrimenti} \end{cases} \tag{1}$$

dove $f(x)$ è il valore di una feature di Haar applicata all'immagine x, $p \in \{-1, 1\}$ è detta *polarità* e θ è la *soglia* (*threshold*). Tutti i classificatori deboli sono costruiti usando un'unica feature.

L'obbiettivo di Adaboost è quello di formare un *classificatore forte* come combinazione lineare dei migliori classificatori deboli estraibili dal set di allenamento, dove il fattore moltiplicativo di ogni classificatore nella combinazione è inversamente proporzionale agli errori di classificazione compiuti da quest'ultimo in fase di allenamento.

1. Si associa ad ogni elemento (x_i, y_i) un peso w_i tale che $w_i = \frac{1}{2l}$ se $(x_i, y_i) \in P$ oppure $w_i = \frac{1}{2m}$ se $(x_i, y_i) \in N$, dove $l = \#(P)$ ed $m = \#(N)$ (numero degli esempi positivi e numero degli esempi negativi).

2. *For* $t = [1 : T]$

   (a) Si normalizzano i pesi, in modo che la loro somma sia pari ad 1:

   $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$

   (b) Si estrae il miglior classificatore debole. La procedura viene esposta nel dettaglio nella sezione 2.4, ma si tenga presente che il miglior classificatore è quello il cui *errore pesato* è minimo per la corrente iterazione.

   $$\epsilon_t = \min_{f,p,\theta} \{ \sum_{i=1}^{n} w_{t,i} \cdot |h(x_i, f, p, \theta) - y_i| \}$$

   Siano inoltre f_t, p_t, θ_t i parametri del classificatore debole che ne minimizzano l'errore pesato:

   $$h_t(x) := h(x, f_t, p_t, \theta_t)$$

   (c) $\beta_t \leftarrow \frac{\epsilon_t}{1-\epsilon_t}$

   (d) Si aggiornano i pesi

   $$w_{t+1,i} \leftarrow w_{t,i} \cdot \beta_t^{e_i}$$

   dove $e_i = 1$ se $(x_i, h_t(x_i)) \in D$ (ovvero se x_i è classificata correttamente), $e_i = 0$ altrimenti.

   (e) $\alpha_t \leftarrow \log(\frac{1}{\beta_t})$

3. Il classificatore forte è dato da:

$$F(x) = \begin{cases} 1 & \text{se } \sum_{t=1}^{T} \alpha_t h_t(x) > \theta \alpha_t \\ 0 & \text{altrimenti} \end{cases} \tag{2}$$

   dove $\theta \in [0, 1]$ è la soglia.

Si noti che, nell'operazione 2b, l'errore pesato non è altro che la somma dei pesi degli esempi non classificati correttamente. Infatti:

$$h(x_i, f, p, \theta) = y_i \Rightarrow |h(x_i, f, p, \theta) - y_i| = 0$$

$$h(x_i, f, p, \theta) \neq y_i \Rightarrow |h(x_i, f, p, \theta) - y_i| = 1$$

Al punto 2c, il valore di β_t non è altro che il rapporto tra l'errore pesato del classificatore debole e la somma dei pesi delle immagini classificate correttamente. Tale valore è chiaramente $0 < \beta_t < 1$.

In fase di aggiornamento dei pesi (punto 2d), i pesi relativi ad esempi classificati correttamente vengono moltiplicati per β_t ($\beta_t^1 = \beta_t < 1$) e quindi decrementati, mentre gli altri vengono lasciati inalterati ($\beta_t^0 = 1$). Fare in modo che gli esempi non classificati correttamente abbiano un peso maggiore di quelli classificati correttamente è il modo per influenzare la scelta del classificatore debole successivo che andrà a colmare le lacune del suo predecessore.

La scelta della soglia per il classificatore forte (punto 3) deve minimizzare il numero di esempi classificati in modo errato.

2.4 Il miglior classificatore debole

Si è detto che un classificatore debole è costruito a partire da una feature di Haar. La scelta del migliore, quindi, mira ad identificare la feature, la polarità e la soglia che minimizzano l'errore pesato di classificazione.

Si ricordi che le feature di Haar sono degli indicatori di quanto le intensità dei pixel variano da una regione della feature ad un'altra. Il classificatore debole, quindi, classificherà l'immagine a seconda che tale indice sia maggiore o minore di una certa soglia. Il compito della polarità è quello di stabilire il verso della diseguaglianza.

Il pool di feature da testare è costituito - teoricamente - da tutte quelle individuabili nell'immagini di allenamento. In [*inseririre riferimento*] Viola-Jones vengono utilizzate immagini di allenamento di 24×24 pixel e 5 tipologie di feature: il numero di possibili features in tale area è maggiore di 160000. In questa applicazione si utilizzano immagini di 160×100 pixel e 4 tipologie di feature: il pool è costituito da un numero di elementi maggiore di almeno 4 ordini di grandezza.

È da tener presente che moltissime di queste feature sono poco significative in questa situazione: non ha senso calcolare la variazione di intensità di due aree molto piccole con delle immagini che hanno una risoluzione tanto alta. Effettuando una prima scrematura, si cercherà di avere un pool di feature selezionabili la cui dimensione non superi quella del pool di Viola-Jones per più di un ordine di grandezza.

Sia $\{f_1, ..., f_k\}$ l'insieme di tutte le feature selezionabili, $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ l'insieme degli esempi di allenamento e $\{w_1, ..., w_n\}$ l'insieme dei relativi pesi. La scelta del classificatore debole avviene come descritto dal seguente algoritmo in pseudocodice:

1. Si calcolano T^+ e T^-, rispettivamente, somma dei pesi degli esempi negativi e di quelli negativi:
$$T^+ \leftarrow \sum_{i=1}^{n}(w_i y_i) \ , \ T^- \leftarrow \sum_{i=1}^{n}[w_i(1-y_i)]$$
2. *For* $f = [f_1, ..., f_k]$
   (a) Si inizializza una lista di n elementi per memorizzare i valori della feature i-esima applicata ad ogni immagine di allenamento:
   $$values[i] \leftarrow f(x_i) \ \forall x_i \in D$$
   (b) Si ordinano gli elementi della lista in ordine crescente. Si tenga in conto che, dopo tale operazione, all'i-esima posizione della lista non corrisponderà più il valore della feature applicata all'i-esima immagine di allenamento.
   (c) Si inizializzano S^+ ed S^-, con le quali, scorrendo gli elementi della lista con un cursore, indicheremo rispettivamente la somma dei pesi degli esempi positivi e di quelli negativi: $S^+ \leftarrow 0, S^- \leftarrow 0$
   (d) *For* $i = [1 : n]$
      i. A causa del rilocamento degli indici, alla posizione i-esima della lista corrisponderà il valore della feature dell'elemento x_j con classificazione y_j e peso w_j tale che $(x_j, y_j) \in D$:
      $$x_j, y_j, w_j \Leftarrow values[i]$$
      ii. *If* $y_i = 1$ *Then*
      A. $S^+ \leftarrow S^+ + w_j$
      iii. *Else*
      A. $S^+ \leftarrow S^- + w_j$
      iv. Si calcola calcola l'errore pesato di classificazione:
      $$e_i = \min\{S^+ + (T^- - S^-), S^- + (T^+ - S^+)\}$$
   (e) Si determinano la polarità (p_f) e la soglia (θ_f) per cui l'errore pesato (ϵ_f) di classificazione per un classificatore che utilizza la feature f è minimo:
   $$p_f, \theta_f | \epsilon_f = \min\{e_1, ..., e_n\}$$
3. Si scelgono polarità (p) e soglia (θ) finali, ovvero quelle del classificatore debole con errore pesato ϵ minore tra tutti i classificatori possibili:
   $$p, \theta | \epsilon = \min\{\epsilon_1, ..., \epsilon_k\}$$
   Il miglior classificatore debole è quindi:
   $$h(x) := \begin{cases} 1 & \text{se } pf(x) < p\theta \\ 0 & \text{altrimenti} \end{cases} \tag{3}$$

Il metodo di identificazione della polarità e della soglia non viene riportato nell'algoritmo, essendo un passaggio che merita una trattazione a parte. Selezionare un valore di soglia vuol dire trovare il *punto che partiziona al meglio la lista dei valori della feature calcolata sulle immagini di allenamento, in modo tale da minimizzare gli errori di classificazione*. La miglior soglia di una buona feature fa in modo che la maggior parte delle immagini appartenenti alla stessa classe assumano un valore minore (o maggiore). Si può notare, come diretta conseguenza, che con una feature pessima per la classificazione non sarà possibile trovare un valore di soglia che soddisfi tale criterio.

Al punto 2(d)iv viene calcolato l'errore pesato di classificazione per una feature f con soglia $values[i]$[3]. Per essere più espliciti, bisogna effettuare una serie di osservazioni:

1. T^+ (T^-) corrisponde alla somma dei pesi degli esempi positivi (negativi);
2. S^+ (S^-) corrisponde alla somma dei pesi degli esempi positivi (negativi) dalla prima posizione fino all'i-esima della lista (quella su cui è posizionato il cursore);
3. $T^+ - S^+$ ($T^- - S^-$) corrisponde alla somma dei pesi degli esempi positivi (negativi) dalla posizione $i+1$ della lista fino alla fine;
4. Per un classificatore della forma (3) con $p = 1$, S^+ corrisponde alla *somma dei pesi degli esempi classificati correttamente*, mentre $S^- + (T^+ - S^+)$ corrisponde alla somma dei pesi degli esempi classificati in modo scorretto;
5. Per l'osservazione 4 un classificatore (3) con $p = 1$, la quantità $S^- + (T^+ - S^+)$ è *l'errore pesato*;
6. Analogamente alle osservazioni 4 e 5, un classificatore (3) con $p = -1$ commetterà un errore pesato pari alla quantità $S^+ + (T^- - S^-)$.

Grazie alle osservazioni 5 e 6, dal semplice calcolo dell'errore di classificazione pesato, si ottiene anche il valore della polarità:

$$p = \begin{cases} 1 & \text{se } S^- + (T^+ - S^+) < S^+ + (T^- - S^-) \\ -1 & \text{altrimenti} \end{cases} \tag{4}$$

In definitiva, con uno scorrimento della lista ordinata si ottengo i parametri per la costruzione del classificatore debole. La complessità di tale operazione è fortemente legata all'algoritmo di ordinamento della lista, la quale ha $\Theta(n \log n)$ come limite teorico inferiore. Nell'implementazione è stato scelto proprio un algoritmo che avesse complessità $\Theta(n \log n)$ nel caso peggiore. Ripetendo queste operazioni per ognuna delle feature selezionabili, si ottiene che l'algoritmo di selezione del miglior classificatore debole ha complessità $\Theta(kn \log n)$.

3 Considerazioni sulla costruzione dei dataset

[3] I possibili valori delle soglie corrispondono esattamente ai valori della feature calcolata sugli esempi di allenamento