Human Activity Recognition per la classificazione di balli caraibici tramite Pose Estimation

Autore: Alberto Tontoni

Matricola: 521076

Corso: Machine Learning

Link al repository: https://github.com/tontonialberto/dance-classifier

Questa relazione riassume il lavoro svolto per il progetto del corso di Machine Learning presso l'Università Roma Tre, anno accademico 2023-2024. Scopo del progetto è la realizzazione di un classificatore binario per la Human Activity Recognition (HAR) nell'ambito delle danze caraibiche: dato un breve video contenente la performance di uno o più ballerini, il classificatore deve essere in grado di distinguere tra due tipi di ballo: **salsa** e **bachata**.

Cenni sul dominio d'interesse

Anzitutto occorre fare una premessa, dovuta al fatto che con i termini "salsa" e "bachata" ci si può riferire sia a generi musicali sia a stili di ballo, concetti sì correlati ma differenti: nel resto di questa trattazione il focus è incentrato sullo stile di ballo.

La salsa e la bachata sono balli di coppia di origine caraibica, essendo Cuba il luogo d'origine della salsa e la Repubblica Dominicana quello della bachata. Inoltre, la diffusione di queste due discipline a livello globale ha portato alla contaminazione e alla creazione di nuovi sottostili. La bachata è un ballo più emozionale e romantico che si balla principalmente in "coppia chiusa" - ossia una sorta di abbraccio tra i due partner - ed è inoltre caratterizzata da movimenti sincopati. La salsa è invece un ballo energico nel quale è più comune l'alternanza di due fasi: il ballo in coppia chiusa e dei piccoli assoli da parte di entrambi i ballerini.

Talvolta i passi caratteristici di una disciplina vengono utilizzati nell'altra. Cio' rende impossibile, persino per un esperto del dominio, identificare la disciplina di appartenenza partendo da una singola immagine di una performance. Avendo invece a disposizione dei brevi video è in genere semplice, per un occhio allenato, riconoscere una sequenza di salsa da una sequenza di bachata.

E' infine opportuno descrivere il setting in cui la maggior parte dei video viene registrata. Facendo ricerche online relative a video di balli di salsa e bachata si possono trovare le seguenti tipologie di video:

- Video "solo": per l'intera durata del video sono presenti unicamente ballerini;
- Video "social": registrati solitamente all'interno di club e discoteche, includono numerose coppie di ballerini, oltre a spettatori interpretabili come rumore di fondo;
- Video "demo": registrati al termine di una lezione, includono solitamente un'unica coppia di ballerini (gli insegnanti) e un elevato numero di spettatori.

Pose Estimation

Il dataset "raw" per il task d'interesse è costituito da una serie di video etichettati con le classi di nostro interesse (ie. salsa, bachata). Tuttavia può essere complicato per un modello di ML classico lavorare direttamente su sequenze di frames. L'approccio individuato fa invece utilizzo di una rete neurale preaddestrata per le attività di *object detection, pose estimation and tracking*:

- **Object Detection**: riconoscimento di oggetti all'interno di un'immagine. Ad ogni oggetto viene solitamente associato un bounding box e un valore di confidenza tra 0 e 1;
- **Pose Estimation**: stima della posizione dei giunti del corpo di ciascuna persona identificata all'interno di un'immagine. Ad ogni persona viene solitamente associato un vettore di punti (x, y, confidenza) per ciascun giunto del corpo;
- **Tracking**: tecnica che associa un identificatore univoco ad ogni persona che appare in più frames consecutivi di un video.

Nel concreto è stato utilizzato il modello preaddestrato yolov8-pose-p6 poichè in grado di offrire tutte le funzionalità sopra elencate.

Nello specifico, YOLOv8 accetta in input una sequenza di frames e, per ciascun frame, restituisce una lista di persone individuate. Per ciascuna persona effettua la stima di:

- coordinate del bounding box (x, y) e valore di confidenza tra 0 e 1;
- 17 punti associati ai giunti principali del corpo (es: spalle, ginocchia, gomiti, ecc.) e relativi valori di confidenza tra 0 e 1;
- un identificatore univoco per consentire il tracciamento di una stessa persona in più frames.

Allo scopo di estrarre tali informazioni dai video, il notebook `estimate-poses.ipynb` salva i dati delle pose in formato JSON. E' poi necessario costruire una time series per ciascun individuo tracciato, suddividere ciascuna serie temporale con un approccio a sliding window, etichettare ciascuna finestra con la classe del video di appartenenza e infine fornire i dati così processati in input al classificatore binario. Queste operazioni di preprocessing vengono discusse nelle seguenti sezioni.

Raccolta Dati

Per svolgere il lavoro qui trattato, i dataset disponibili in rete sono risultati inadeguati: non esistono banche di dati etichettati con le classi di nostro interesse. Si è dunque proceduto a individuare ed etichettare manualmente video rappresentativi delle due classi a partire da risorse pubblicamente disponibili (eg. video di YouTube). Al termine di tale attività sono stati raccolti 66 video di cui 33 appartenenti alla classe "salsa" e 33 alla classe "bachata".

A questo punto è necessario utilizzare la pose estimation and tracking sopra presentata per estrarre le pose dei ballerini e costruire le serie temporali. Tuttavia le caratteristiche stimate da YOLOv8 sono tutt'altro che perfette:

- Si hanno problemi di tracciamento se la persona non è visibile o lo è solo parzialmente per numerosi frames: Questo fenomeno accade nella maggior parte dei video raccolti a causa dei frequenti scambi di posizione dei ballerini durante la performance. Ciò si traduce con diversi

ID associati ad una stessa persona e quindi con delle serie temporali di durata potenzialmente molto limitata;
- Può accadere che vi sia occlusione di alcune parti del corpo delle persone in un video. In tal caso YOLOv8 associa un valore pari a 0 sia alle coordinate (x,y) sia al valore di confidenza del punto in questione;
- Alcuni video includono moltissimo rumore di fondo, come passanti o spettatori. Questo fenomeno si manifesta soprattutto nei video di tipologia "demo" e, in misura minore, nei video "social". Tale rumore si traduce in svariate finestre temporali che, se non eliminate, vengono etichettate con una delle due classi e possono mandare "in confusione" un modello di ML addestrato su tali dati.

Per risolvere il primo problema sarebbe necessario implementare un algoritmo di *reidentification* poichè non implementato da YOLOv8: per semplicità si è deciso di trascurare questa problematica. Sono invece state applicate delle tecniche di trattamento dei missing values e di filtraggio per ridurre l'impatto del secondo e del terzo problema.

> Verrebbe da chiedersi la motivazione dell'inclusione volontaria di video altamente rumorosi: in fondo, se si avessero a disposizione numerosi video di tipologia "solo" il terzo problema non si porrebbe. Tuttavia in rete è disponibile solo un numero limitato di video di tale tipologia, mentre negli ultimi anni è aumentato considerevolmente il volume di video "demo" e "social" registrati durante festival e congressi internazionali.

Data Cleaning

Come evidenziato in precedenza, viene costruita una time series per ogni sequenza di pose associate a uno stesso ID di tracciamento. Ciascuna time series viene innanzitutto processata nel modo seguente:

- Se nella time series esiste un frame nel quale non è stato individuato nemmeno un punto chiave, si fa uno split in due time series e si genera automaticamente un ID per la nuova porzione. Viene fatto ciò poichè può accadere che YOLOv8 inverta gli ID di tracciamento associati a due ballerini durante uno scambio di posizione: quello che accade durante tale scambio tra il ballerino X e il ballerino Y è che per pochi frames uno dei due ballerini non è più tracciato, e quando torna visibile il modello scambia X per Y e viceversa;
- Si eliminano dal dataset le time series che hanno solo missing values per un'intera coordinata di un qualsiasi punto;
- I missing values vengono interpolati *linearmente*: sono stati tentati anche approcci di interpolazione quadratica e MICE (Multiple Imputation of Chained Equations), tuttavia il loro utilizzo conduceva a movimenti poco realistici;
- Si riduce lo "sfarfallio" nei movimenti stimati da YOLOv8 tramite un filtro moving average sugli ultimi 5 frames. Utilizzare una finestra più lunga per effettuare la media causava la perdita di movimenti rilevanti, ad esempio dei passi dei ballerini;
- I punti chiave vengono normalizzati rispetto al bounding box in modo da risultare nell'intervallo [0,1].

Per rimuovere le serie temporali non rilevanti (es: spettatori), l'approccio è costituito da due fasi da eseguire una dopo l'altra:

1. Eliminazione di sequenze temporali troppo corte: si assume che coloro che appaiono in un video per pochissimi secondi siano rumore di fondo;
2. Eliminazione delle sequenze con una "variazione media di posizione" inferiore ad una soglia: persone che appaiono in un video per molto tempo ma rimangono sostanzialmente ferme sono da considerarsi rumore di fondo.

Dunque si eliminano tutte le sequenze di lunghezza inferiore a 60 frames (circa un secondo per i video collezionati). A seguire, si calcola la *variazione totale media di posizione* delle sequenze rimanenti, usando la seguente formula:

$$\Delta\text{Pos}_{\text{tot}} = \frac{1}{N} \sum_{i=1}^{N-1} \sum_{j=1}^{17} \frac{c_{i,j} + c_{i+1,j}}{2} \sqrt{(x_{i,j} - x_{i+1,j})^2 + (y_{i,j} - y_{i+1,j})^2}$$

dove:

- N è il numero di frames in cui compare la persona considerata;
- $c_{i,j} \in [0, 1]$ è il valore di confidenza associato al punto $(x_{i,j}, y_{i,j})$
- $x_{i,j} \in [0, 1]$ è il j-esimo punto chiave dell'i-esimo frame.

E' poi necessario scalare le variazioni calcolate in modo che possano essere confrontabili fra loro: a tal fine viene impiegato il `MinMaxScaler` fornito dalla libreria `scikit-learn` per il mapping nell'intervallo [0,1]. Infine vengono eliminate tutte le time series tali che:

$$\Delta\text{Pos}_{\text{tot}} < 0.2$$

I valori soglia per il filtraggio sono stati fissati tramite esperimenti trial-and-error su un sottoinsieme dei video disponibili, che tuttavia hanno condotto a risultati soddisfacenti:

- Nei video "demo" e "social" si riescono ad eliminare praticamente *tutti* gli spettatori;
- Viene comunque preservata la maggior parte dei movimenti dei ballerini.

L'analisi di questi risultati è stata possibile grazie a degli script realizzati ad hoc che visualizzano un vero e proprio video a partire dai dati delle pose. A seguire la comparazione tra i dati raw di un frame rispetto a quelli filtrati:

A destra le pose raw di un video "demo": oltre alla coppia di ballerini (al centro) sono presenti numerosi spettatori. A sinistra sono riportate le pose filtrate con il procedimento sopraindicato: in questo caso specifico è stato rimosso *ogni* spettatore. Per visualizzare comparative di questo genere è stato utilizzato lo script `visualizer.py`.

> Nel repository del progetto sono presenti dei video comparativi realizzati tramite lo script sopraindicato. Sono disponibili nella cartella `resources/media`.

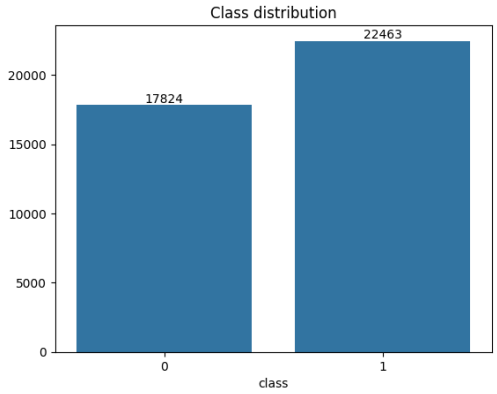
Feature Engineering e Creazione del Dataset

Creazione del Dataset

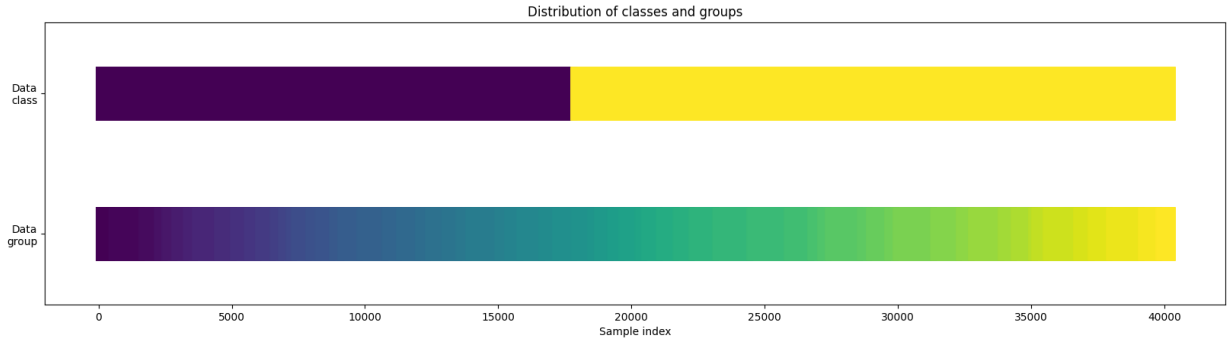
E' ora necessario trasformare i dati filtrati in un formato accettabile da un classificatore binario. A tale scopo si utilizza il comune approccio "a sliding window" relativo al Machine Learning applicato alle Time Series. Le finestre sono parametrizzate come segue:

- Window size di 30 frames;
- Sovrapposizione del 50% tra finestre consecutive;
- Step size pari a 2: in pratica, ogni finestra è relativa a 2 secondi di video, campionato a 15 frames per secondo.

Utilizzando tale tecnica si ottiene un dataset in cui ciascuna osservazione è caratterizzata da circa 1600 features. Con i dati a disposizione si sono ottenute 40287 osservazioni. A seguire una rappresentazione del bilanciamento del dataset:



Come è possibile notare, il dataset è leggermente sbilanciato in favore della classe 1 ("salsa"). Vi è anche un altro tipo di sbilanciamento, ossia tra i diversi "gruppi" costituenti il dataset: in pratica, vi sono video che contengono più osservazioni (ie. finestre) di altri. Ciò viene evidenziato dalla rappresentazione di seguito riportata:



La barra in alto è equivalente al grafico precedente. La barra in basso rappresenta invece un video diverso per ogni tonalità di colore: si può notare come la distribuzione delle osservazioni tra i video non sia perfettamente uniforme. E' importante tenere a mente questo doppio sbilanciamento in fase di partizionamento tra dati di training/validation/test.

Feature Engineering

Sulla base di articoli scientifici in merito alla classificazione di dati caratterizzati da pose, si è ritenuto opportuno applicare delle tecniche di feature engineering per calcolare delle caratteristiche "ad alto livello" a partire dalle coordinate (x,y) a disposizione:

- Velocità ed accelerazioni tra frames consecutivi;
- Distanze mutue tra i punti chiave di uno stesso frame;
- Angoli tra punti chiave di uno stesso frame.

Otteniamo così ulteriori 2538 features che, sommate alle precedenti, conducono ad un totale di circa 4100 features.

Si è infine ritenuto utile calcolare statistiche relative a ciascuna quantità di ciascuna finestra: media, deviazione standard, percentili, valore massimo e minimo.

Valutazione delle prestazioni

Sono stati utilizzati due differenti approcci per valutare le prestazioni di un classificatore binario:

- Valutazione di più algoritmi di ML utilizzando l'80% dei video per il training set e il restante 20% per il test set. Non è stata applicata la Cross Validation;
- Valutazione della curva di apprendimento dell'algoritmo di ML con il miglior bilanciamento tra prestazioni e tempo di training, utilizzando una 5-fold Cross Validation sul training set costituito dall'80% dei dati disponibili.

Valutazione di più classificatori

Questo è stato il primo approccio utilizzato. La suddivisione tra dati di training e dati di test è fatta utilizzando un bilanciamento "naive" tra le classi: l'80% dei video di ciascuna classe va a formare il training set. Si ottengono dunque 58 video di training (29 per classe) e 8 video di test (4 per classe).

	Training Set	**Test Set**
# Class 0	15879	1945
# Class 1	19490	2973
# Total	35369	4918

Per mantenere il bilanciamento tra le classi, viene utilizzato il random undersampling offerto dalla libreria `imblearn`.

Nella scelta dei modelli addestrare si è tenuto conto delle principali famiglie di algoritmi, identificando dunque i seguenti candidati:

- Stochastic Gradient Descent (SGD)
- Random Forest
- Histogram Gradient Boosting
- Support Vector Machine con kernel RBF

Ciascun modello è stato addestrato sullo stesso numero di osservazioni, ma facendo 3 esperimenti rispetto al numero di features:

- raw: vengono utilizzate solo le features estratte dal modello di pose estimation;
- hlf: si utilizzano anche le features "ad alto livello" illustrate in precedenza;
- stats: si utilizzano anche le statistiche per singola finestra illustrate in precedenza.

La metrica utilizzata è f1-score: nella tabella che segue, "F1 avg" rappresenta la *macro* f1-score, mentre "F1 0/1" è la f1-score rispetto a una delle due classi.

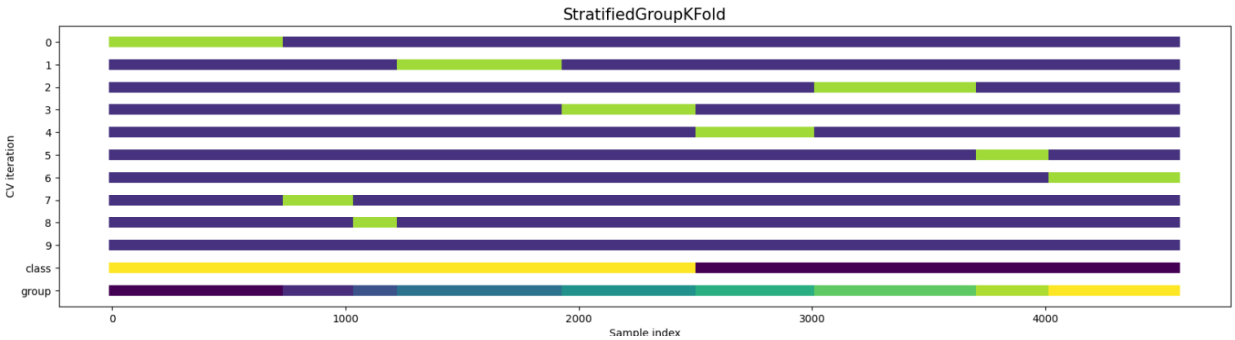
A seguire un riepilogo dei risultati ottenuti:

Modello	**F1 0 (raw)**	**F1 1 (raw)**	**F1 avg (raw)**	**F1 0 (hlf)**	**F1 1 (hlf)**	**F1 avg (hlf)**	**F1 0 (stats)**	**F1 1 (stats)**	**F1 avg (stats)**
SGD	0.6	0.66	**0.63**	0.66	0.72	**0.69**	0.57	0.67	**0.62**
Random Forest	0.65	0.76	**0.705**	0.67	0.76	**0.715**	0.65	0.72	**0.685**
Histogram Gradient Boosting	0.65	0.75	**0.70**	0.64	0.74	**0.69**	0.64	0.72	**0.68**
SVM RBF	0.69	0.77	**0.73**	0.68	0.75	**0.715**	0.68	0.75	**0.715**

Ne deduciamo che aggiungere features di alto livello può aumentare le prestazioni. Il caso più evidente è SGD che passa dal 63% al 69%. In altri casi le prestazioni peggiorano. E' probabile che i modelli siano andati in overfitting.

Cross Validation

In questo caso l'approccio per la suddivisione train/test è più elaborato: al fine di ottenere un buon compromesso tra bilanciamento tra gruppi e quello tra le classi, si è utilizzato lo `StratifiedGroupKFold` offerto da `scikit-learn`. Teniamo a mente che il soddisfacimento di un vincolo di bilanciamento puo' andare in conflitto con l'altro. Ad esempio, `StratifiedGroupKFold` per 10 fold e un dataset sufficientemente piccolo (in termini di numero di video) può portare ai seguenti partizionamenti (in verde il test set):



Notiamo però che ciascun test set include osservazioni *di soltanto una delle due classi*. In sintesi è dunque importante stabilire il numero di fold in accordo con la dimensione del dataset.

Utilizzando l'80% (circa) dei dati per il training set, otteniamo la seguente suddivisione:

	Training Set	Test Set
# Class 0	14127	3697
# Class 1	17962	4501
# Total	32089	8198

Il training set viene ulteriormente partizionato per realizzare una 5-fold cross validation di un modello Random Forest. Ad ogni fold viene effettuato random undersampling e, per ridurre la varianza tra le differenti fold e anche il tempo di training, viene applicata PCA con una varianza spiegata del 99.9%. Questo approccio viene ripetuto per dimensioni crescenti del dataset al fine di ottenere la curva di apprendimento. I risultati sono riportati di seguito:

Numero di osservazioni	F1 average (hlf)	F1 standard deviation (hlf)	Numero di componenti PCA (hlf) (in media)
1000	0.5303	0.0183	646
5000	0.5726	0.0308	906
10000	0.5879	0.0317	940
15000	0.5944	0.0309	951
19200	0.5945	0.0285	956

Notiamo che le prestazioni risultano migliorare all'aumentare del numero di osservazioni. Tuttavia la suddivisione train/test, la cross validation e il sottocampionamento hanno drasticamente ridotto le dimensioni del dataset, che nel caso "migliore" contiene solo 19200 osservazioni totali - quasi la metà rispetto al caso senza cross-validation e con train/test split fatto in maniera "naive".

Un ultimo tentativo è stato effettuato utilizzando una tecnica di oversampling, in particolare SMOTE. I risultati non vengono riportati per brevità, tuttavia si è notata una diminuzione delle prestazioni soprattutto nei fold molto sbilanciati - si ipotizza dunque che le stime fatte da SMOTE con i dati a disposizione siano poco affidabili.

Conclusioni e sviluppi futuri

Il lavoro qui presentato ha consentito di valutare l'applicabilità di tecniche di Machine Learning nel contesto del riconoscimento tra stili di ballo che presentano delle mutue somiglianze. Dai risultati ottenuti si evince che, collezionando più dati, un kernel RBF otterrebbe un buon livello di prestazioni. Inoltre i dati necessari sono ampiamente disponibili in rete e le tecniche di filtraggio qui presentate consentono di trattare anche risorse potenzialmente molto rumorose.

Tra gli sviluppi futuri consiglio innanzitutto di raccogliere più video: rivalutare le prestazioni con almeno 50 video per ciascuna classe. Tecniche di data augmentation non sono state incluse per motivi di tempo ma sono senz'altro una strada percorribile. Infine, il modello risultante risulterebbe senza dubbio più utile se si includesse un'ulteriore classe a rappresentazione di tutte le attività che non coincidono con il ballo.