Relazione Progetto DCML

Davide Zhang

20 dicembre 2024

1 Introduzione

Il progetto consiste nell'implementazione di un rilevatore di auto-clicker attraverso modelli machine-learning. Lo svolgimento del progetto è stato effettuato in più fasi:

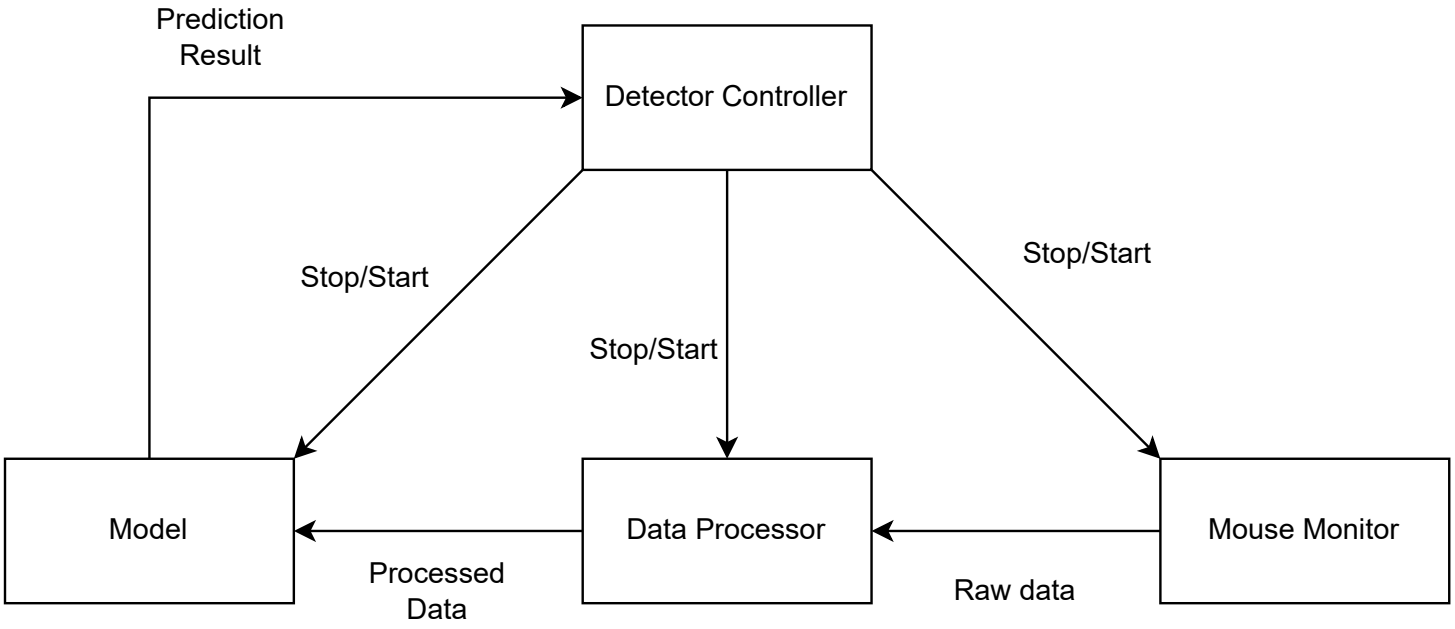
1. Ricerca monitor per registrazione degli eventi mouse e raccolta dati iniziali per il training del modello.
2. Analisi degli dati per la scelta di attributi rilevanti e la loro elaborazione in batch etichettati.
3. esplorazione di modelli adeguati per il problema, allenamento e quindi confronto delle prestazioni.
4. Implementazione finale del detector e test con le varie modalità d'esecuzione di un auto clicker rudimentale implementato da me.

2 Struttura del rilevatore

Il rilevatore è composto da quattro moduli che vengono eseguiti su thread paralleli:

- Detector Controller
  Implementa l'interfaccia utente dell'applicazione e coordina le attività degli altri moduli.
- Mouse Monitor
  Monitora le attività del mouse registrando gli eventi "mouse press" e "mouse release" e successivamente inviandoli al processore degli dati.
- Processore Dati
  Riceve e elabora i dati grezzi dal monitor per costruire degli batch di dati (contenenti dati estratti da un numero arbitrario di eventi clic) da inviare al modello per previsione dell'etichetta.
- Modello ML
  Effettua la previsione con un modello machine-learning precedentemente allenato e comunica il risultato al Controller.

Pagina successiva mostra una mappa riassuntiva della struttura dell'applicazione.


Prediction
Result
Detector Controller
Stop/Start
Stop/Start
Stop/Start
Model
Data Processor
Mouse Monitor
Processed
Data
Raw data

3 Scelta del Monitor e attributi

Usando la libreria Python pynput possiamo monitorare facilmente i vari eventi di input da mouse o tastiera, in particolare quelli riguardanti i clic dei tasti del mouse: gli altri eventi, pur interessanti, non sono molto rilevanti per l'attività di un auto-clicker che consiste principalmente nella simulazione di segnali da tasto sinistro del mouse. Andando ad ignorare gli eventi di rotazione della rotella e del movimento del mouse, ad ogni clic del mouse vengono associati due eventi: digitazione e rilascio del tasto. Per ognuna di questi due tipi di eventi, pynput chiama il metodo callback "on_click" passata all'istanziazione del listener. Il metodo ha quattro argomenti: le coordinate in pixel del clic, il tasto premuto e un valore booleano che indica se l'evento si tratta di una digitazione o rilascio. Una prima elaborazione che possiamo fare è quella di combinare le coppie di eventi digitazione-rilascio per creare una tupla rappresentante l'intero clic. Prima di effettuare l'elaborazione c'è sempre la necessità di controllare la consistenza dei dati:

- i tipi devono corrispondere: un evento digitazione seguito da rilascio
- il tempo del primo evento deve essere minore di quello del secondo.

Otteniamo quindi una lista di possibili attributi di un clic:

- timestamp del clic (tempo al momento della digitazione)
- durata del clic (tempo tra digitazione e rilascio)
- posizione x partenza
- posizione y partenza
- modulo r vettore spostamento durante il clic
- angolo θ del vettore spostamento in radianti rispetto all'asse x (nel sistema di riferimento dello schermo gli angoli avranno un orientamento diverso rispetto a quello normale in quanto y diminuisce per pixel più alti)
- tasto mouse usato

Tra questi attributi ho deciso poi di tralasciare l'angolo θ in quanto penso che la direzione di movimento per clicchi di mouse trascinati sia poco influente sulla classificazione e anche perché gli attributi stanno diventando un po' troppi. Nel passo di elaborazione successivo ho deciso di raggruppare i dati in batch, in modo da evidenziare attributi emergenti come media del tempo al clic successivo, e diminuire lo sforzo computazionale della previsione. Questo perché la frequenza dei clicchi potrebbe essere molto elevata per la presenza di un auto-clicker.

Il risultato è la seguente lista degli attributi:

- media tempo al clic successivo (TTNC), deviazione standard TTNC
- media durata clicchi, deviazione standard durata clicchi
- coordinate posizione media clicchi
- vettore spostamento medio tra clicchi in coordinate polari
- deviazione standard modulo e angolo vettore spostamento tra clicchi
- media e deviazione standard distanza trascinamento dei clicchi
- proporzione dei clicchi con tasto sinistro del mouse

Siccome la semplice divisione dei dati raccolti in batch di dimensione uguale porta ad una riduzione significativa del numero di campioni disponibili per il processo di allenamento, ho deciso di applicare una tecnica di raggruppamento diversa, inserendo eventi clic in più batch e assicurando che ogni batch contenga solo clic immediatamente successivi l'un l'altro.

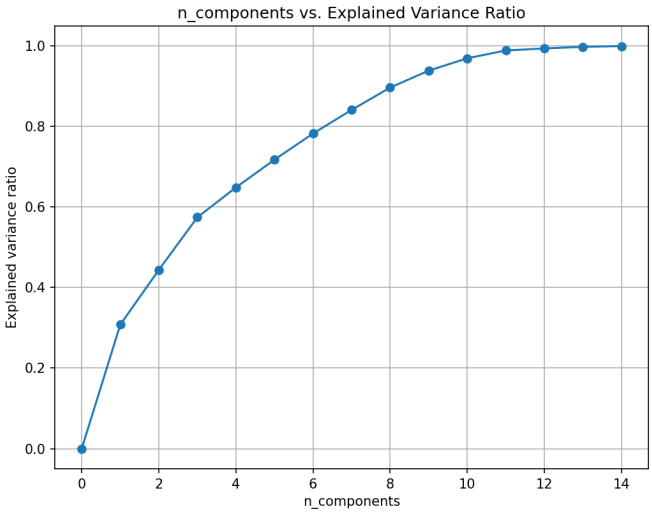
4 Raccolta dati e allenamento modello

La raccolta dei dati è stata fatta separando le sessioni di normale attività del mouse da quelle con presenza di un auto-clicker ed etichettandole in maniera corrispondente. I dati raccolti sono stati poi elaborati prima di essere usati nell'allenamento dei modelli. Sono stati allenati e testati 6 diversi algoritmi machine-learning:

- Standard Scaler + Principal Component Analysis + K-Means
- Standard Scaler + Random Forest
- Standard Scaler + ADA Boost
- Standard Scaler + Multi Layer Perceptron
- Standard Scaler + Logistic Regression
- Standard Scaler + C-Support Vector Classification

L'algoritmo che ha dato i risultati peggiori è stato l'algoritmo di apprendimento non supervisionato K-Means senza molte sorprese. Non ci sono infatti garanzie che i due raggruppamenti che l'algoritmo realizza possano coincidere semanticamente con quelli richiesti dalla classificazione. Una strategia di verifica sarebbe quella di controllare le etichette generate da K-Means con quelle vere per il dataset di allenamento e verificarne la sovrapposizione,

scegliendo il raggruppamento con sovrapposizione maggiore. Per migliorare le prestazioni dell'algoritmo ho cercato di effettuare un'analisi PCA per diminuire la dimensionalità del dataset mantenendo per più possibile la varianza: dall'analisi ho osservato che riducendo a 11 componenti riuscivo comunque a mantenere la maggior parte della varianza di partenza.



Successivamente riporto le prestazioni dell'algoritmo misurate in diverse metriche senza e con l'applicazione di PCA.

	accuratezza in train-set	MCC in train-set	accuratezza in test-set	MCC in test-set
senza PCA	0.64493	-0.03383	0.65149	-0.03112
con PCA	0.69608	0.50390	0.68306	0.48933

Possiamo osservare un lieve miglioramento dell'accuratezza della previsione e un consistente miglioramento per quanto riguarda il MCC anche se le prestazioni dell'algoritmo rimangono molto più basse rispetto agli algoritmi supervisionati.

Una procedura d'analisi simile è stata applicata agli algoritmi supervisionati, misurando le metriche di MCC e di accuratezza da convalida incrociata e da test degli algoritmi con o senza standardizzazione precedente dei dati in input da parte di uno standard scaler.

	MCC	accuratezza da convalida incrociata	accuratezza da test
Con standardizzazione antecedente			
Random Forest	0.99866	1.0	0.99939
ADA Boost	0.99866	1.0	0.99939
MultiLayer Perceptron	0.99866	0.99880	0.99939
Logistic Regression	0.99466	0.99551	0.99757
Support Vector Machine	0.998661	0.99940	0.99939
Senza standardizzazione antecedente			
Random Forest	0.99866	1.0	0.99939
ADA Boost	0.99866	1.0	0.99939
MultiLayer Perceptron	0.93768	0.96411	0.97146
Logistic Regression	0.92075	0.96291	0.96296
Support Vector Machine	-	-	-

Senza la rimozione della media e il ridimensionamento a varianza unitaria effettuata dallo standard scaler possiamo notare una diminuzione di prestazioni che è generale per i algoritmi non basati su alberi di decisione: per C-Support Vector Classification, in particolare, il tempo di allenamento è salito in maniera iperbolica. Questo si verifica perché in questi algoritmi molti elementi usati dalla funzione obiettivo assumono che gli attributi fossero centrati in zero e che la varianza sia nello stesso ordine di grandezza. Per quanto riguarda il tuning dei parametri per i vari algoritmi ho ottenuto i seguenti risultati:

- Il numero degli estimatori non ha grossa influenza sulle prestazioni dei metodi ensemble. Le prestazioni di Random Forest iniziano a scendere un pochino solo per n_estimator=3, mentre per ADA Boost l'accuratezza e MCC non scendono affatto al diminuire del numero degli estimatori. Questo forse è dovuto al fatto di poter arrivare a previsione molto buone anche solo guardando la durata media dei clic: se questo è molto vicino a zero il batch dei clic sarà probabilmente generata da un auto-clicker.

- Il fattore C non ha molte influenze sulle ottime prestazioni del SVM ma in generale fa decrescere l'accuratezza e il MCC al diminuire di C per il Logistic Regression. Inoltre il solver preferibile per il linear regression è newton-cholesky dato che n_samples $\gg$ n_features * n_classes nel nostro caso. Questo è confermato dalle metriche migliori.

- Per l'algoritmo del MultiLayer Perceptron, in generale, fornito un numero sufficiente di neuroni nel primo strato, le prestazioni non variano molto anche se si aggiungono molti neuroni negli strati successivi.

Per quanto riguarda il modello usato nel rilevatore, è possibile usare tutti i modelli prodotti dagli algoritmi supervisionati anche se il modello di default è il MLP. Per cambiare modello da eseguire nel rilevatore basta cambiare il percorso file passato al costruttore dell'oggetto MachileLearningModel a quello desiderato.