Metodi e strumenti utilizzati

Machine Learning

Il Machine Learning, o apprendimento automatico, è un campo di studio che si occupa di sviluppare algoritmi per i calcolatori che sono in grado di migliorare automaticamente grazie all'esperienza acquisita tramite l'utilizzo dei dati. Gli algoritmi di Machine Learning creano modelli matematici a partire da dati di esempio, chiamati "dati di training", in modo da poter fare predizioni o prendere decisioni senza essere esplicitamente programmati per farlo. Ci sono tre categorie di approcci di Machine Learning: l'apprendimento supervisionato, l'apprendimento non supervisionato e l'apprendimento per rinforzo.

L'apprendimento supervisionato consiste nell'insegnare al calcolatore una regola generale che mappi gli input e gli output desiderati. L'algoritmo di apprendimento è fornito di dati di input di esempio e degli output corrispondenti e, attraverso successive iterazioni, costruisce un modello matematico che può essere usato per predire l'output associato a un nuovo input.

L'apprendimento non supervisionato, invece, non fornisce all'algoritmo di apprendimento le etichette desiderate. In questo caso, l'algoritmo di apprendimento deve estrarre le informazioni significative dai dati di input senza conoscere a priori l'output desiderato.

L'apprendimento per rinforzo prevede che un programma interagisca con un ambiente dinamico in cui deve raggiungere un obiettivo specifico, come ad esempio guidare un veicolo o vincere un gioco contro un avversario. Durante l'interazione, il programma riceve un feedback sotto forma di premio e cerca di massimizzarlo, in modo da imparare a raggiungere l'obiettivo prefissato.

Computer Vision

La Computer Vision è un campo interdisciplinare che si occupa della capacità dei computer di acquisire conoscenza da immagini o video, cercando di replicare il funzionamento dell'apparato visivo umano. La Computer Vision utilizza metodi per l'acquisizione e l'analisi di immagini digitali, in modo da estrarre dati multidimensionali dal mondo reale e produrre informazioni numeriche o simboliche, come decisioni. La Computer Vision si avvale di conoscenze di geometria, fisica, statistica e teoria dell'apprendimento per descrivere il mondo in modo sensato, producendo pensieri che possono portare alla corretta linea d'azione.

Visual Studio Code

Visual Studio Code è un editor di codice sorgente sviluppato da Microsoft per Windows, Linux e macOS, che supporta il debugging, il controllo Git integrato, la Syntax Highlighting, l'IntelliSense, lo Snippet e il refactoring del codice. Visual Studio Code supporta molteplici linguaggi e funzionalità aggiuntive grazie alla possibilità di installare dei plugin disponibili attraverso un repository centrale. Nel testo, si fa riferimento all'utilizzo di Visual Studio Code con il supporto a Python per il lavoro descritto.

Pandas

La libreria software open source Pandas è stata sviluppata per il linguaggio di programmazione Python ed è utilizzata per la manipolazione e l'analisi dei dati. Con Pandas è possibile effettuare operazioni su tabelle numeriche e serie temporali grazie alle sue strutture dati. Il nome "Pandas" deriva dal termine econometrico "Panel Data", che indica un insieme di dati contenenti osservazioni sugli stessi individui durante più periodi di tempo.

Cuda

[da https://developer.nvidia.com/cuda-zone]

CUDA® è una piattaforma di calcolo parallelo e un modello di programmazione sviluppato da NVIDIA per il calcolo generale su unità di elaborazione grafica (GPU). Con CUDA, gli sviluppatori possono aumentare significativamente la velocità delle applicazioni di calcolo sfruttando la potenza delle GPU.

Nelle applicazioni con accelerazione GPU, la parte sequenziale del carico di lavoro viene eseguita sulla CPU - ottimizzata per le prestazioni single-threaded - mentre la parte computazionalmente intensiva dell'applicazione viene eseguita su migliaia di core GPU in parallelo. Quando si utilizza CUDA, gli sviluppatori programmano in linguaggi popolari come C, C++, Fortran, Python e MATLAB ed esprimono la parallelismo attraverso estensioni sotto forma di poche parole chiave di base.

Il toolkit CUDA di NVIDIA fornisce tutto il necessario per sviluppare applicazioni con accelerazione GPU. Il toolkit CUDA include librerie accelerate su GPU, un compilatore, strumenti di sviluppo e il runtime CUDA.

Torch

PyTorch è un popolare framework open-source di deep learning che consente agli sviluppatori di creare modelli di intelligenza artificiale in modo rapido ed efficiente. È stato sviluppato originariamente da Facebook AI Research e ha guadagnato una grande popolarità grazie alla sua facilità d'uso, alla sua flessibilità e alla sua scalabilità.

Una delle principali caratteristiche di PyTorch è la sua architettura a flusso di dati (data flow), che rende il framework particolarmente adatto per le applicazioni di deep learning. Inoltre, PyTorch è dotato di un'ampia gamma di librerie e strumenti, come ad esempio PyTorch Lightning, che semplificano lo sviluppo di modelli di intelligenza artificiale.

PyTorch è anche conosciuto per la sua flessibilità e scalabilità, in quanto permette di creare modelli di deep learning sia per computer singoli che per cluster di computer. Inoltre, PyTorch supporta una vasta gamma di piattaforme hardware, come CPU, GPU e TPU, il che lo rende adatto per le applicazioni in ambiti come il machine learning, la visione artificiale e il linguaggio naturale.

In sintesi, PyTorch è uno dei framework più popolari e potenti per la creazione di modelli di intelligenza artificiale, grazie alla sua flessibilità, alla sua scalabilità e alla sua architettura a flusso di dati.

CNN (Convolutional Neural Networks)

Una rete convoluzionale, o CNN, è un tipo di rete neurale usata per l'analisi delle immagini. Rispetto ad altri algoritmi, le CNN richiedono meno lavoro di preprocessing perché la rete è in grado di imparare a ottimizzare i propri filtri automaticamente. La struttura della CNN è ispirata alla biologia, in particolare all'organizzazione della corteccia visiva animale. I neuroni in una CNN sono organizzati in modo tale che ogni neurone risponde a stimoli in una piccola area del campo visivo, noto come "campo ricettivo". I campi ricettivi dei diversi neuroni si sovrappongono parzialmente per coprire l'intero campo visivo.

Una CNN consiste di più strati che trasformano l'input in un output come punteggi di appartenenza a delle classi. Alcuni dei tipi di strati più comuni includono lo strato convoluzionale, che utilizza filtri per rilevare la presenza di una feature in un qualsiasi punto dell'input, lo strato di pooling, che riduce il numero di parametri e controlla l'overfitting, e gli strati di neuroni completamente connessi. La rete impara i filtri che vengono attivati quando viene rilevata la presenza di una feature, senza la necessità di effettuare un preprocessing manuale.

In sintesi, le CNN rappresentano uno strumento importante per l'analisi delle immagini grazie alla loro capacità di apprendere automaticamente i filtri, alla loro organizzazione biologicamente ispirata e alla loro capacità di ridurre il lavoro di preprocessing richiesto.

OpenCV

[Da https://opencv.org/about/]

OpenCV (Open Source Computer Vision Library) è una libreria software open source per la computer vision e il machine learning. È stata creata per fornire un'infrastruttura comune per le applicazioni di computer vision e per accelerare l'uso della percezione automatica nei prodotti commerciali. In quanto prodotto con licenza Apache 2, OpenCV facilita l'utilizzo e la modifica del codice da parte delle aziende.

La libreria contiene più di 2500 algoritmi ottimizzati, che includono un insieme completo di algoritmi di computer vision e machine learning sia classici che all'avanguardia. Questi algoritmi possono essere utilizzati per rilevare e riconoscere volti, identificare oggetti, classificare azioni umane nei video, tracciare il movimento della telecamera, tracciare oggetti in movimento, estrarre modelli 3D di oggetti, produrre nuvole di punti 3D da telecamere stereo, unire immagini per produrre un'immagine ad alta risoluzione di un'intera scena, trovare immagini simili da un database di immagini, rimuovere gli occhi rossi dalle immagini scattate con il flash, seguire i movimenti degli occhi, riconoscere paesaggi e creare marker per sovrapporli alla realtà aumentata, ecc. OpenCV ha più di 47mila utenti nella sua comunità e un numero stimato di download superiore a 18 milioni. La libreria viene utilizzata ampiamente in aziende, gruppi di ricerca e da organismi governativi.

Oltre alle aziende ben consolidate come Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda e Toyota, che utilizzano la libreria, ci sono molte startup come Applied Minds, VideoSurf e Zeitera che ne fanno un uso intensivo. OpenCV è utilizzato per molteplici applicazioni, tra cui unire immagini di Street View, rilevare intrusioni in video di sorveglianza in Israele, monitorare l'equipaggiamento minerario in Cina, aiutare i robot a navigare e raccogliere oggetti presso Willow Garage, rilevare gli incidenti di annegamento in piscina in Europa, eseguire arte interattiva in Spagna e New York, controllare le piste di atterraggio per rilevare detriti in Turchia, ispezionare le etichette sui prodotti nelle fabbriche di tutto il mondo e per la rapida rilevazione dei volti in Giappone.

OpenCV ha interfacce per C++, Python, Java e MATLAB e supporta Windows, Linux, Android e Mac OS. OpenCV si concentra principalmente sulle applicazioni di visione in tempo reale e sfrutta le istruzioni MMX e SSE quando disponibili. Sono attualmente in sviluppo interfacce complete per CUDA e OpenCL. Ci sono oltre 500 algoritmi e circa 10 volte tante funzioni che compongono o supportano questi algoritmi. OpenCV è scritto nativamente in C++ e ha un'interfaccia templatica che funziona perfettamente con i contenitori STL.

Random forest

Da [https://www.ibm.com/topics/random-forest#:~:text=Random%20forest%20is%20a%20commonly,both%20classification%20and%20regression%20problems.]



La random forest è un algoritmo di apprendimento automatico comunemente usato, marchiato da Leo Breiman e Adele Cutler, che combina l'output di più alberi decisionali per raggiungere un singolo risultato. La sua facilità d'uso e la flessibilità hanno alimentato la sua adozione, in quanto gestisce sia problemi di classificazione che di regressione.

Alberi decisionali

Dato che il modello di random forest è composto da più alberi decisionali, sarebbe utile iniziare descrivendo brevemente l'algoritmo dell'albero decisionale. Gli alberi decisionali partono da una domanda di base, come ad esempio "Dovrei fare surf?". Da lì, è possibile porre una serie di domande per determinare una risposta, come "C'è un'onda di lungo periodo?" o "Il vento soffia a riva?". Queste domande costituiscono i nodi decisionali dell'albero, agendo come mezzo per suddividere i dati. Ogni domanda aiuta un individuo a giungere a una decisione finale, che sarebbe indicata dal nodo foglia. Le osservazioni che soddisfano i criteri seguiranno il ramo "Sì" e quelle che non li soddisfano seguiranno il percorso alternativo. Gli alberi decisionali cercano di trovare la miglior suddivisione per i dati e vengono tipicamente addestrati attraverso l'algoritmo Classification and Regression Tree (CART). Metriche come l'impurità di Gini, il guadagno di informazione o l'errore quadratico medio (MSE) possono essere utilizzati per valutare la qualità della suddivisione.

Questo albero decisionale è un esempio di un problema di classificazione, dove le etichette di classe sono "fare surf" e "non fare surf".

Sebbene gli alberi decisionali siano comuni algoritmi di apprendimento supervisionato, possono essere soggetti a problemi come il bias e l'overfitting. Tuttavia, quando più alberi decisionali formano un insieme nell'algoritmo di random forest, predicono risultati più accurati, in particolare quando i singoli alberi non sono correlati tra loro.

L'algoritmo della random forest

L'algoritmo della random forest è un'estensione del metodo di bagging in quanto utilizza sia il bagging che la casualità delle caratteristiche per creare una foresta di alberi decisionali non correlati. La casualità delle caratteristiche, anche nota come bagging delle caratteristiche o "il metodo del sottospazio casuale", genera un sottoinsieme casuale di caratteristiche che assicura una bassa correlazione tra gli alberi decisionali. Questa è una differenza chiave tra gli alberi decisionali e le foreste casuali. Mentre gli alberi decisionali considerano tutte le possibili suddivisioni delle caratteristiche, le foreste casuali selezionano solo un sottoinsieme di quelle caratteristiche.

Se torniamo all'esempio "dovrei fare surf?", le domande che potrei porre per determinare la previsione potrebbero non essere così esaustive come il set di domande di qualcun altro. Tenendo conto di tutta la potenziale variabilità dei dati, possiamo ridurre il rischio di sovradattamento, di bias e di varianza complessiva, ottenendo previsioni più precise.

Come funziona

Gli algoritmi delle foreste casuali hanno tre iperparametri principali che devono essere impostati prima dell'allenamento. Questi includono la dimensione del nodo, il numero di alberi e il numero di caratteristiche campionate. Da lì, il classificatore della foresta casuale può essere utilizzato per risolvere problemi di regressione o di classificazione.

L'algoritmo della foresta casuale è composto da una collezione di alberi decisionali, e ogni albero nell'insieme è costituito da un campione di dati tratto da un set di allenamento con sostituzione, chiamato campione di bootstrap. Di quel campione di allenamento, un terzo viene messo da parte come dati di test, noti come campione fuori dalla borsa (oob), a cui torneremo in seguito. Un'altra istanza di casualità viene quindi iniettata attraverso il bagging delle caratteristiche, aggiungendo maggiore diversità al dataset e riducendo la correlazione tra gli alberi decisionali. A seconda del tipo di problema, la determinazione della previsione varierà. Per un compito di regressione, gli alberi decisionali individuali verranno mediati, mentre per un compito di classificazione, una maggioranza di voti - ossia la variabile categorica più frequente - darà come risultato la classe prevista. Infine, il campione oob viene utilizzato per la convalida incrociata, finalizzando quella previsione.

Benefici e sfide del random forest

Ci sono diversi vantaggi e sfide chiave che l'algoritmo random forest presenta quando utilizzato per problemi di classificazione o regressione. Alcuni di essi includono:

Principali vantaggi

Riduzione del rischio di overfitting: Le alberi di decisione corrono il rischio di overfitting poiché tendono ad adattarsi strettamente a tutti i campioni all'interno dei dati di formazione. Tuttavia, quando ci sono un robusto numero di alberi di decisione in un random forest, il classificatore non sovrastimerà il modello poiché la media di alberi scorrelati riduce la varianza complessiva e l'errore di previsione.

Fornisce flessibilità: poiché il random forest può gestire sia compiti di regressione che di classificazione con un elevato grado di precisione, è un metodo popolare tra i data scientist. Inoltre, la feature bagging rende il classificatore random forest uno strumento efficace per stimare i valori mancanti poiché mantiene l'accuratezza quando una parte dei dati è mancante.

Facile determinazione dell'importanza delle feature: il random forest rende facile valutare l'importanza delle variabili, o il contributo, al modello. Ci sono alcuni modi per valutare l'importanza della feature. L'importanza di Gini e la diminuzione media dell'impurità (MDI) vengono solitamente utilizzati per misurare quanto diminuisce l'accuratezza del modello quando una determinata variabile viene esclusa. Tuttavia, l'importanza permutazione, nota anche come diminuzione media dell'accuratezza (MDA), è un'altra misura di importanza. MDA identifica la diminuzione media dell'accuratezza permutando casualmente i valori delle feature nei campioni out-of-bag.

Principali sfide

Processo che richiede tempo: poiché gli algoritmi random forest possono gestire grandi set di dati, possono fornire previsioni più accurate, ma possono essere lenti nell'elaborazione dei dati in quanto computano i dati per ogni singolo albero decisionale.

Richiede più risorse: poiché i random forest elaborano set di dati più grandi, richiedono più risorse per archiviare quei dati.

Più complesso: la previsione di un singolo albero decisionale è più facile da interpretare rispetto a una foresta di essi.

Dataset risulato:

Come anticipato nel primo capitolo il dataset utilizzato per il mio caso di studio è il risultato dell’unione dei 2 dataset Student engagement dataset e DAiSEE.

Le immagini al loro interno sono state inizialmente elaborate attraverso la libreria py-feat per ottenere le misure delle Action Units.

Le labels risultanti e il numero di valore per ogniuna di queste sono:

* enagegd con 55707 samples
* bored con 16086 samples
* confused con 1041 samples
* frustated con 495 samples
* looking away con 409 samples
* frustated con 344 samples
* drowsy con 240 samples

con un totale di 74322 immagini (o frame estratti da video) per le quali sono stati generati i dati relativi alle Action Units.

TODO Parla degli altri dati presenti nel dataset, oltre alle AUs

Generazione descrizione in linguaggio naturale

TODO

Estrazione delle Action Units utilizzando la libreria Py-feat

Prima di poter effettuare delle predizioni è necessaria la creazione di un oggetto Detector fornito dalla libreria.

\begin{minted}[bgcolor=bg]{python}

device = "cuda" if torch.cuda.is\_available() else "cpu"

return Detector(

device=device,

face\_model="retinaface",

landmark\_model="mobilefacenet",

au\_model="xgb",

emotion\_model="resmasknet",

facepose\_model="img2pose",

)

\end{minted}

Come è possibile notare è possibile nel codice, durante la creazione dell’oggetto Detector è possibile specificare il parametro \mintinline[bgcolor=bg]{python}{device}, questo permette di eseguire le operazioni utilizzando la tecnologia cuda sulla quale ho già approfondito. Per controllare che sia effettivamente possibile utilizzare questa funzionalità è stata usata la libreria torch per python, tecnologia sulla quale ho precedentemente approfondito.

Il parametro face\_model imposta il modello di rilevamento del viso da utilizzare. Ho deciso di impostato su "retinaface", perche questo è un popolare modello di rilevamento del viso che utilizza una CNN (Convolutional Neural Networks) sulla quale ho già approfondito.

Il parametro landmark\_model imposta il modello di rilevamento dei landmark facciali da utilizzare. Qui, è impostato su "mobilefacenet", che è un Single-stage dense face localisation in the wild, ottenuto dagli autori della libreria da [18]

Il parametro au\_model imposta il modello utilizzato per la rilevazione automatica dell'unità d'azione (AU) facciale. Il modello è un classificatore Extreme Gradient Boosting (XGB) estratto, dagli autori di py-feat da i datasets BP4D, DISFA, CK+, UNBC-McMaster shoulder pain, e AFF-Wild2 e basato sul lavoro di [19].

Il parametro emotion\_model imposta il modello utilizzato per la rilevazione delle emozioni dalle espressioni facciali. Il modello utilizzato si chiama "resmasknet", implementato utilizzando il lavoro di [20].

Il parametro facepose\_model imposta il modello utilizzato per la stima della posa della testa. Il modello utilizzato si chiama "img2pose", implementato utilizzando il lavoro di [21].

Py-feat permette di estrarre i valori delle Action units attraverso il metodo \mintinline[bgcolor=bg]{python}{detector.detect\_image(imagePath)} che prende in input il percorso di un’immagine e restituisce i valori estratti; i valori estratti da questo metodo non si limitano alle Action Units che sono da me utilizzate vengono calcolati anche altri valori, quali:

* FaceRectX: la coordinata X dell'angolo in alto a sinistra del rettangolo del viso rilevato nell'immagine di input
* FaceRectY: la coordinata Y dell'angolo in alto a sinistra del rettangolo del viso rilevato nell'immagine di input
* FaceRectWidth: la larghezza del rettangolo del viso rilevato
* FaceRectHeight: l'altezza del rettangolo del viso rilevato
* FaceScore: un punteggio che indica il livello di fiducia del modello di rilevamento del viso nella regione del viso rilevata
* x\_0 a x\_67: le coordinate X dei 68 punti landmark facciali rilevati dal modello di landmark
* y\_0 a y\_67: le coordinate Y dei 68 punti landmark facciali rilevati dal modello di landmark
* Pitch: l'angolo di inclinazione del volto (inclinazione su o giù) rilevato dal modello di posizione del volto
* Roll: l'angolo di rollio del volto (inclinazione a sinistra o destra) rilevato dal modello di posizione del volto
* Yaw: l'angolo di imbardata del volto (girare a sinistra o destra) rilevato dal modello di posizione del volto
* anger, disgust, fear, happiness, sadness, surprise, neutral: i punteggi di probabilità delle classi di emozioni rilevate come previsto dal modello di emozione.
* input: il percorso dell'immagine di input
* frame: l'indice del frame elaborato (se si sta elaborando più di un frame)

I risultati ottenuti sono poi stati trasformati in formato json attraverso il metodo \mintinline[bgcolor=bg]{python}{detector.detect\_image(imagePath).to\_json()}, aggregati e salvati su un file, sempre in questo formato, cosi da poterli mostrare in modo più chiaro e, successivamente, questo file è stato trasformato in formato csv per una lettura più veloce da parte della libreria pandas.

Per quanto riguarda i video analizzati dal dataset DAiSEE la libreria offre il metodo \mintinline[bgcolor=bg]{python}{detector.detect\_video(videoPath, skip\_frames)}.

Il parametro \mintinline[bgcolor=bg]{python}{videoPath} si riferisce ovviamente al percorso del video dal quale estrarre i dati; mentre il parametro \mintinline[bgcolor=bg]{python}{skip\_frames} è un intero che determina ogni quanti frame estrapolare l’immagine per calcolarne i relativi valori.

Ho deciso di estrarre un’immagine per ogni secondo di video, ed ho quindi scritto un metodo attraverso il quale estrarre il framerate ognuno dei video:

\begin{minted}[bgcolor=bg]{python}

def getFPS (videoPath):

cap = cv2.VideoCapture(videoPath)

fps = cap.get(cv2.CAP\_PROP\_FPS)

cap.release()

return fps

\end{minted}

Il risultato di questo metodo è stato poi dato in input al metodo per effettuare l’analisi del video.

Le analisi dei video sono organizzati in modo diverso rispetto alle analisi per le immagini in quanto ognuno dei campi citati prima (FaceRectX, FaceRectY, …) contengono al loro interno i campi per i singoli frame, esempio:

\begin{minted}[bgcolor=bg]{json}

"FaceRectX": {

"0.0": 334.3970982143,

"30.0": 325.8671875,

"60.0": 319.8182291667,

"90.0": 314.8222470238,

"120.0": 313.5849330357,

"150.0": 312.7389136905,

"180.0": 312.5695684524,

"210.0": 307.6665178571,

"240.0": 310.235639881,

"270.0": 312.9242931548

},

\end{minted}

È quindi stato necessario effettuare una rielaborazione dei file ottenuti per portare ognuno dei dati estratti nello stesso formato delle immagini:

\begin{minted}[bgcolor=bg]{json}

{

"FaceRectX": 2.4332027435,

"FaceRectY": 1.9402399063,

"FaceRectWidth": 39.422876358,

"FaceRectHeight": 42.0940465927,

"FaceScore": 0.6566667557,

"x\_0": 6.6779442048,

"x\_1": 5.354107498,

"x\_2": 4.4593806637,

…,

\end{minted}

Una volta ottenuti tutti i dati in un singolo file json (e parallelamente nel file csv) i dati sono stati puliti, eseguendo queste operazioni:

* pulizia dei valori nulli:
  + sono state rimosse le righe dei datasets risultanti dalle analisi, attraverso il codice: \mintinline[bgcolor=bg]{python}{df = df.dropna(subset=['AU01'])}

Il codice presentato rimuove ogni riga dove il valore della colonna \mintinline[bgcolor=bg]{python}{AU01} è nullo.

In rari casi, py-feat ha trovato difficoltà nel riconoscere il volto della persona presente nel video, o questa non era presente all’interno dell’immagine, questo ha portato al mancato riconoscimento di tutte le AUs e degli altri dati; filtrando le righe vuote per una sola colonna (la prima delle AUs) ottengo la rimozione di tutte le righe del tutto vuote in modo efficiente.

Per verificare la mancanza di righe vuote ho esguito il seguente codice:

\begin{minted}[bgcolor=bg]{python}

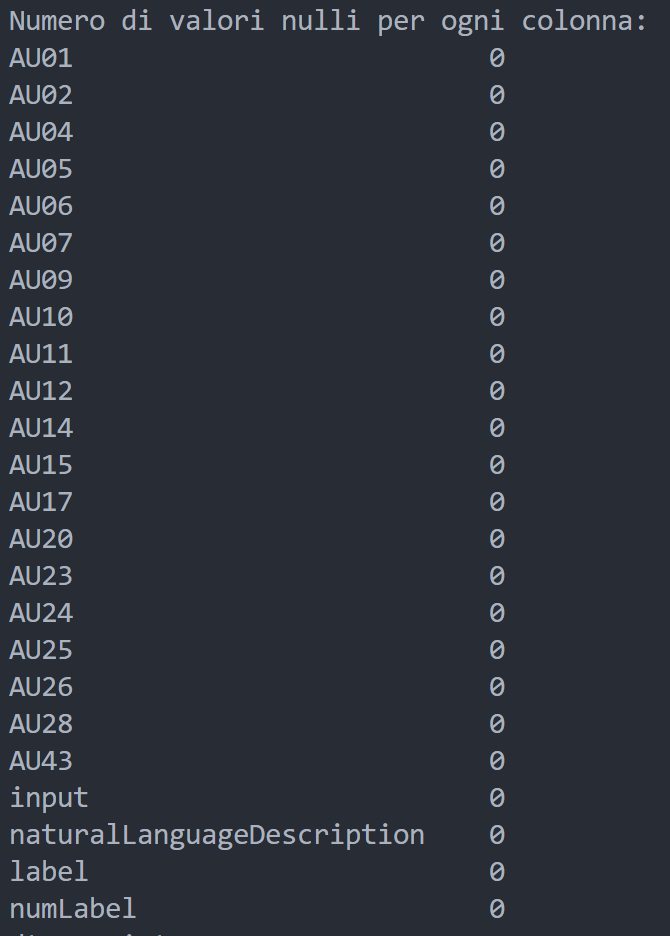
nullVals = df.isnull()

print("Numero di valori nulli per ogni colonna:")

print(nullVals.sum())

\end{minted}

che dà in output:



Come è possibile vedere dall’immagine, il dataset non presenta valori nulli

* aggiunta del valore di frame per ognuna delle analisi dei video:
  + ogni analisi dei singoli frame di un video presentava lo stesso percorso di input (il file video associato), ho quindi aggiunto alla fine del valore della colonna di input il frame dal quale sono state estratte le analisi
* rimozione colonne non aus
* aggiunta colonne
  + label
  + numlabel
  + descrizione in linguaggio naturale

TODO