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.

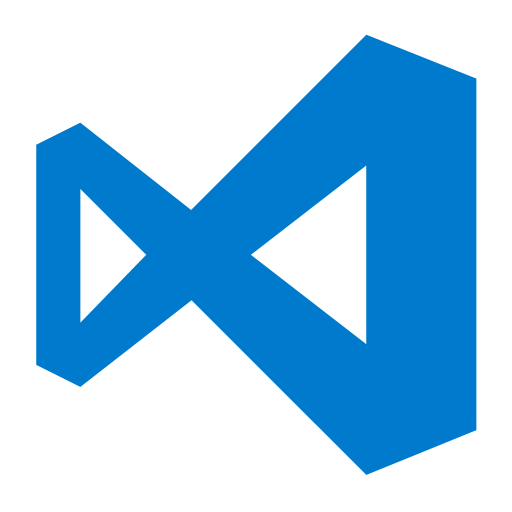
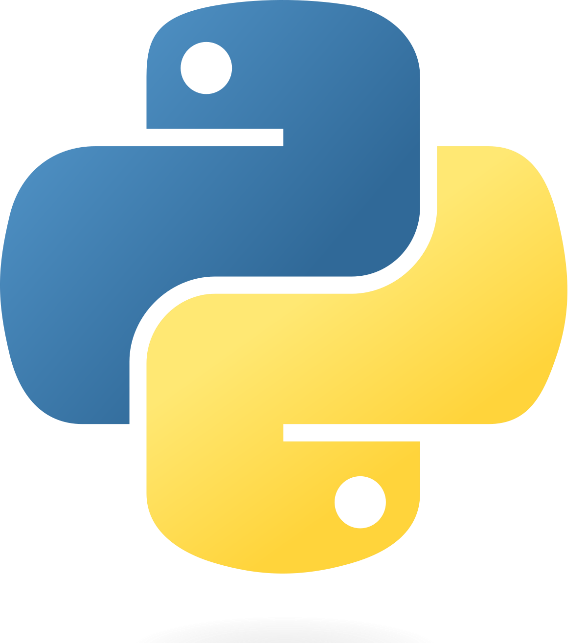
L'apprendimento per rinforzo prevede che un programma interagisca con un ambiente dinamico in cui deve raggiungere un obiettivo specifico, come ad esempio guidare un veicolo o vincere un gioco contro un avversario. Durante l'interazione, il programma riceve un feedback sotto forma di premio e cerca di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



Tqdm

[da https://tqdm.github.io/]

La libreria Python tqdm è uno strumento molto utile per la visualizzazione di barre di avanzamento durante i cicli di elaborazione nel codice.

Il nome "tqdm" deriva dall'unione della parola araba "taqaddum" che significa "progresso" ed è l'abbreviazione di "te quiero demasiado" (ti amo troppo) in spagnolo.

Per utilizzare la libreria, basta inserire qualsiasi iterabile (liste, dizionari, tuple e set) in questo modo metodo \mintinline[bgcolor=bg]{python}{tqdm(iterable)}.

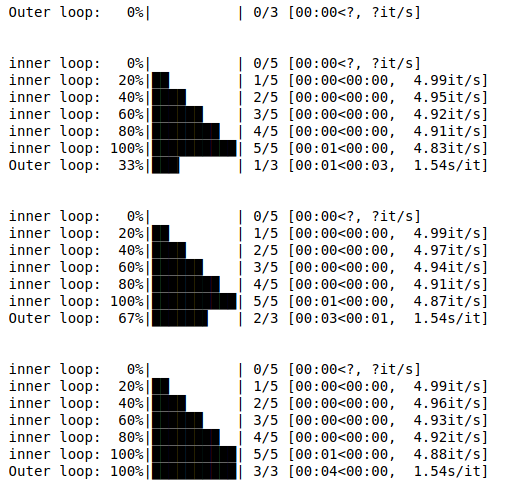
La libreria funziona su qualsiasi piattaforma ed è completamente indipendente dalle dipendenze.

Durante l'estrazione delle Action Units, la libreria tqdm ha fornito una stima precisa del tempo necessario per completare l'elaborazione, consentendo di monitorare l'avanzamento del processo in tempo reale.

Nonostante l'utilizzo della tecnologia CUDA per accelerare le analisi, la grande quantità di immagini da elaborare ha richiesto molto tempo.

La presenza di tqdm è stata quindi fondamentale in quanto ha permesso di gestire efficacemente l'elaborazione dei dati, evitando eventuali problemi tecnici, garantendo l'ottenimento di risultati accurati ed affidabili e facendomi rivalutare scelte algoritmiche, non efficientissime, prese.

In sintesi, la libreria Python tqdm è uno strumento prezioso per semplificare l'elaborazione di grandi quantità di dati, fornendo una stima del tempo rimanente e consentendo di pianificare il lavoro in modo efficiente.



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.



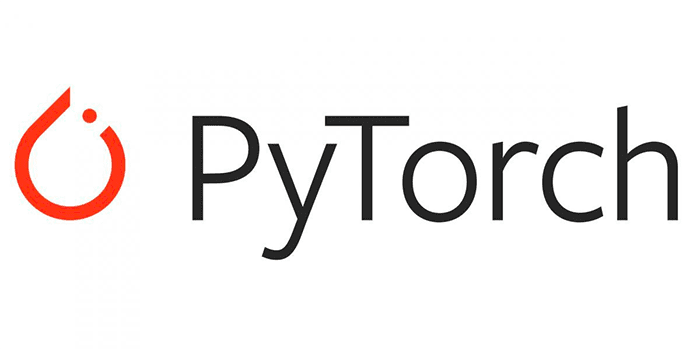
Torch

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.

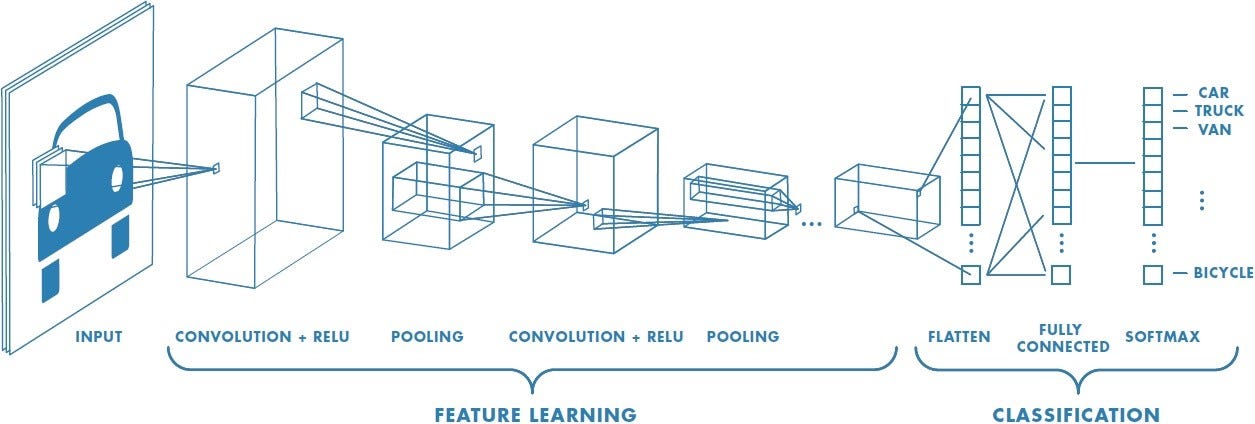


CNN (Convolutional Neural Networks)

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

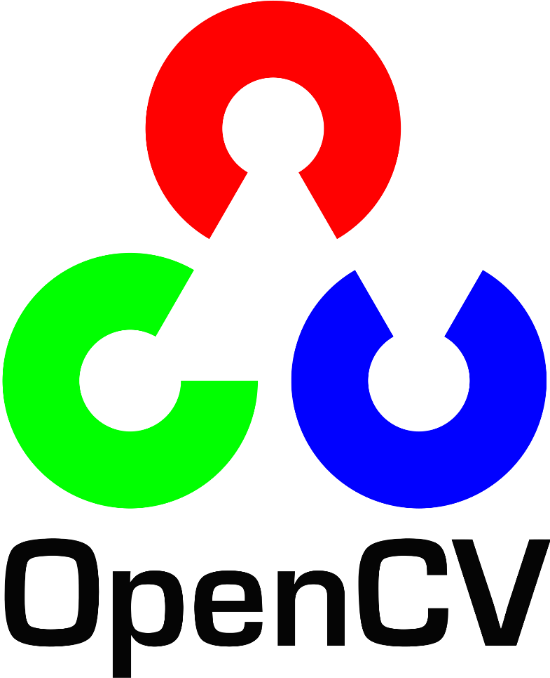
[Da https://opencv.org/about/]

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.



Scikit-learn

[da <https://en.wikipedia.org/wiki/Scikit-learn>]

Scikit-learn (precedentemente conosciuto come scikits.learn e anche noto come sklearn) è una libreria di machine learning gratuita per il linguaggio di programmazione Python. La libreria include vari algoritmi di classificazione, regressione e clustering, tra cui support-vector machine, random forest, gradient boosting, k-means e DBSCAN, ed è progettata per funzionare in combinazione con le librerie numeriche e scientifiche di Python, come NumPy e SciPy. Scikit-learn è un progetto finanziato da NumFOCUS.

Il progetto scikit-learn è nato come un progetto Google Summer of Code dal data scientist francese David Cournapeau, originariamente chiamato scikits.learn. Il nome del progetto deriva dal concetto di "SciKit" (SciPy Toolkit), un'estensione di terze parti separata e distribuita per SciPy. Il codice originale è stato successivamente riscritto da altri sviluppatori. Nel 2010, i contribuenti Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort e Vincent Michel, dall'Istituto francese per la ricerca in informatica e automazione a Saclay, Francia, hanno preso il comando del progetto e hanno rilasciato la prima versione pubblica della libreria il 1 febbraio 2010. Nel novembre 2012, scikit-learn e scikit-image sono stati descritti come due delle "scikits library" ben mantenute e popolari. Nel 2019, si è notato che scikit-learn è una delle librerie di machine learning più popolari su GitHub.

Scikit-learn è principalmente scritto in Python e utilizza ampiamente NumPy per l'algebra lineare ad alta prestazione e le operazioni sugli array. Inoltre, alcuni algoritmi core sono scritti in Cython per migliorare le prestazioni. Support vector machine è implementato da un wrapper Cython intorno a LIBSVM; la regressione logistica e le macchine a vettori di supporto lineari da un wrapper simile intorno a LIBLINEAR. In tali casi, estendere questi metodi con Python potrebbe non essere possibile.

Scikit-learn si integra bene con molte altre librerie di Python, come Matplotlib e Plotly per la visualizzazione, NumPy per la vettorizzazione degli array, Pandas dataframes, SciPy e molte altre.



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 di immagini utilizzato per il mio caso di studio è il risultato dell’unione dei due 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.

Generazione descrizione in linguaggio naturale

sotto richiesta del professore ho aggiunto una descrizione in linguaggio naturale di ogni immagine utilizzando il seguente codice:

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

if value and value >= 0.5:

return outAU.get ("FACS Name") + ", using the muscles: " + outAU.get ("Muscles") + ", with a value of " + str (value) + "; "

else:

return ""

\end{minted}

Questo algoritmo verifica inizialmente che il valore passato in input al metodo (non riportato interamente in quanto prevede azioni preliminari che è inutile spiegare qui) sia presente e successivamente, se ha anche un valore maggiore o uguale a 0.5 (il range di valori è fra 0 e 1) restituisce la stringa che descrivo qui sotto, altrimenti restituisce una stringa vuota.

La frase restituita dall’algoritmo se il valore è maggiore o uguale a cinque è composta dal nome FACS della relativa Action Unit, viene poi aggiunto il muscolo analizzato da questa Action Unit e il valore che è stato prelevato.

La frase presente nel dataset per ognuno dei samples è il concatenamento delle frasi generate per ogni immagine e separate da un “;”, ad esempio:  
Upper Lip Raiser, using the muscles: Levator Labii Superioris, with a value of 0.6412415504; Dimpler, using the muscles: Buccinator, with a value of 0.6336596608; Chin Raiser, using the muscles: Mentalis, with a value of 0.6474888921; Lip Pressor, using the muscles: Orbicularis Oris, with a value of 0.582298696;

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}

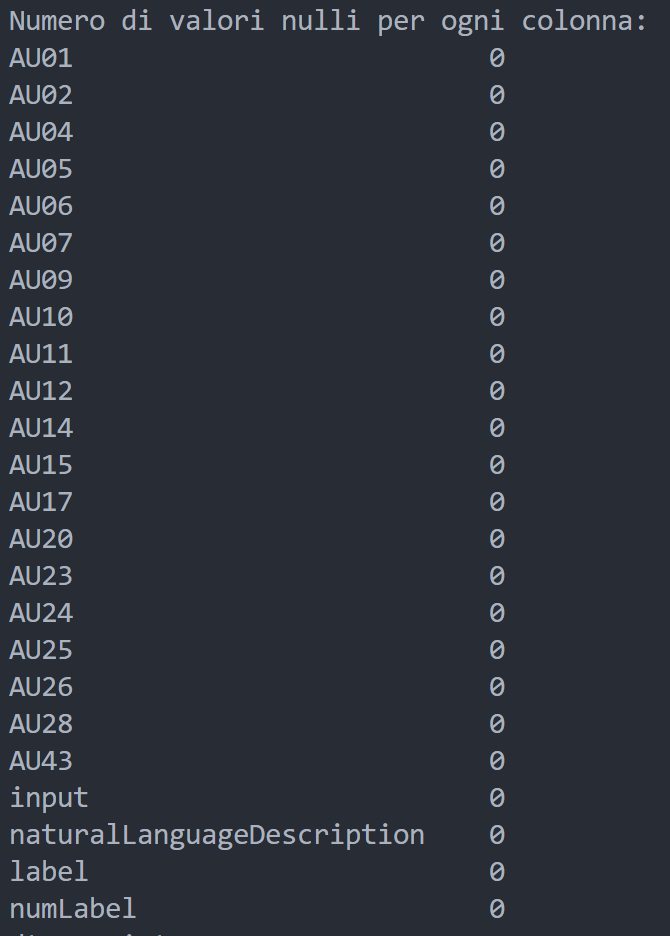
nullVals = df.isnull()

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 delle colonne che non riguardano le Action Units
* aggiunta colonne al dataset
  + label:
    - le analisi inizialmente estrapolate non presentavano già le relative label e le ho quindi dovute aggiungere
  + numLabel
    - ho associato ad ognuna delle label presenti un numero da 0 a 5:
      * 0 🡪 confused
      * 1 🡪 engaged
      * 2 🡪 frustrated
      * 3 🡪 bored
      * 4 🡪 drowsy
      * 5 🡪 looking away
  + descrizione in linguaggio naturale descritta precedentemente

Creazione del random forest classifier

Per effettuare delle predizioni sul dataset ho realizzato un classificatore random forest sul quale effettuare poi delle query fornendogli i dati riguardanti le Action Units, da nuove immagini, sempre attraverso la libreria py-feat.

Per la creazione del classificatore ho per prima letto il file csv contenente il dataset pre-elaborato (modifiche delle quali ho precedentemente parlato sulle quali aggiungero successivamente, in quanto è risultato applicarne ulteriori per migliorare la precisione del predittore creato), rimosso le colonne non necessarie, ho poi diviso il dataset in set di addestramento e di test usando la funzione train\_test\_split della libreria sklearn.model\_selection, con l’output di questo metodo ho ricavato i pandas’s dataframes Xtrain, Xtest, yTrain, yTest.

Questo è il metodo relativo:

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

def getXtrainYTrain():

pd.read\_csv(join(dirname(abspath(\_\_file\_\_)), "../final analysis/DAiSEE and student engagement dataset clean sampled.csv"))

y = df['label']

X = df.drop(["input","naturalLanguageDescription","label","numLabel"], axis=1)

Xtrain, Xtest, yTrain, yTest = train\_test\_split(X, y, test\_size=0.2, random\_state=69)

return Xtrain, yTrain

\end{minted}

Una volta ottenuti questi dataset ho effettuato la creazione del classificatore utilizzando l’oggetto a disposizione fornito dalla libreria sklearn.ensable.

Il RandomForestClassifier viene generato con 100 alberi di decisione e viene addestrto con l’utilizzo dei due dataframe Xtrain e yTrain restituiti dalla funzione getXtrainYTrain().

Questo è il metodo relatio:

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

randomForestClassifier = RandomForestClassifier(n\_estimators=100, verbose=True , random\_state=42)

Xtrain, yTrain = getXtrainYTrain ()

randomForestClassifier.fit(Xtrain, yTrain)

return randomForestClassifier

\end{minted}

Oltre alla creazione del classificatore viene anche generato un grafico per la visualizzazione dell’influenza di ognuna delle label sulla predizione che riporto qui:

Immagine che contiene grafico, grafico a torta

Descrizione generata automaticamente

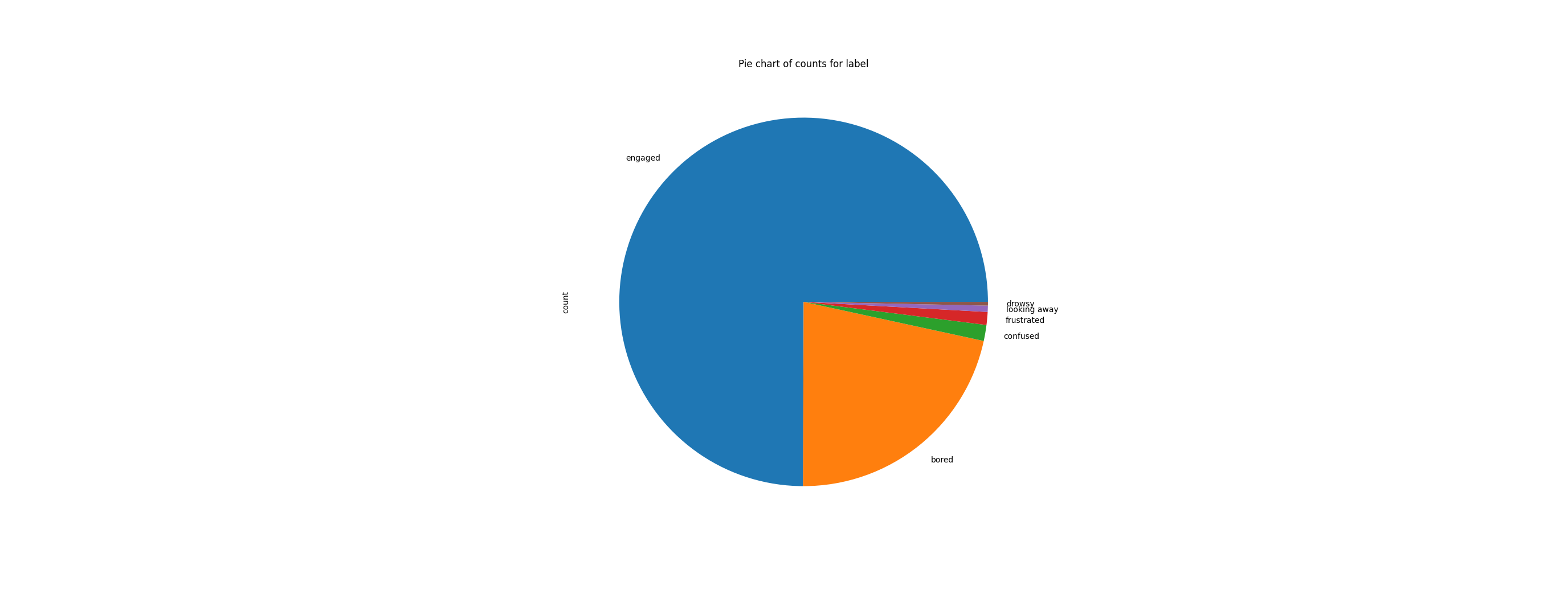
Come è possibile notare quasi tutte le label influenzano la predizione in modo simile (in un range fra il 7,2% e l’11,9%) tranne per la AU43 (Eyes Closed) che, ovviamente, influenza molto la predizione effettuata dal modello.

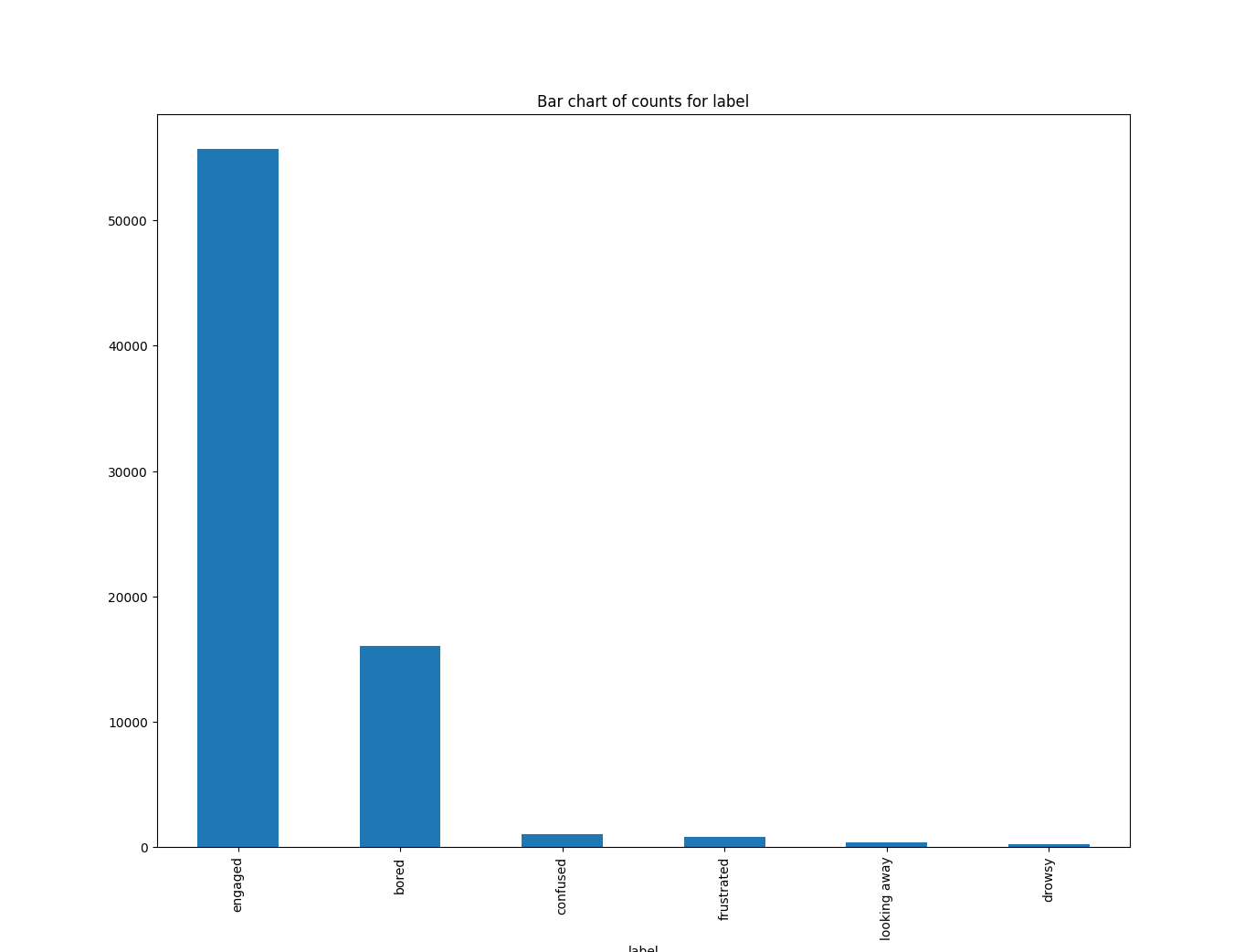
Resampling del dataset per migliorare la precisione delle predizioni

Il dataset risultato che ho creato presenta un grande problema se dovessi decidere di effettuare delle predizioni attraverso un classificatore random forest generato direttamente dal dataset as-his, ovvero il fatto che il numero di campioni (samples) per ognuno dei valori della colonna labels sono sbilanciati.

Per sbilanciati intendo il fatto che sono presenti molti valori per alcune delle classi e troppi pochi, a confronto, per altri.

In questi grafici è possibile vedere la differenza fra il numero di elementi per ogni valore unico nella colonna label:





Il resampling, ovvero la tecnica di modificare la distribuzione dei dati di un dataset mediante la rimozione o l’aggiunta, in modo casuale, di valori simili a quelli delle sue istanze, è una tecnica comune per bilanciare dataset sbilanciati o per migliorare le prestazioni di modelli di machine learning. Ci sono due tipi principali di resampling: undersampling e oversampling.

L'undersampling, come suggerisce il nome, consiste nel rimuovere alcune delle istanze della classe maggioritaria (ovvero quella con un maggior numero di campioni) in modo da bilanciare la distribuzione delle classi nel dataset. Questo può essere fatto in modo casuale, ma è anche possibile utilizzare tecniche più sofisticate come l'eliminazione degli esempi più vicini (nearest neighbor deletion) o la selezione degli esempi più rappresentativi (prototype selection).

L'oversampling, d'altra parte, consiste nell'aumentare il numero di istanze della classe minoritaria (ovvero quella con un minor numero di campioni) in modo da bilanciare la distribuzione delle classi nel dataset. Ci sono molte tecniche per l'oversampling, tra cui la duplicazione casuale degli esempi esistenti, la generazione di nuovi esempi sintetici attraverso tecniche come la Synthetic Minority Over-sampling Technique (SMOTE), e la duplicazione degli esempi esistenti con una variazione minore (data augmentation).

In sintesi, il resampling è una tecnica utile per bilanciare dataset sbilanciati e migliorare le prestazioni dei modelli di machine learning.

Il codice utilizzato per effettuare l’undersampling e l’oversampling delle classi è il seguente:

undersampling

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

def undersampleDataset(df, columnName, valueToDownSample, numberOfSamplesAfter):

if len(df[df[columnName] == valueToDownSample]) > numberOfSamplesAfter:

engagedIndices = df[df[columnName] == valueToDownSample].index

tempDf = df.loc[engagedIndices]

tempDfUndersampled = tempDf.sample(n=numberOfSamplesAfter, random\_state=69)

print ()

return pd.concat([df.drop(engagedIndices), tempDfUndersampled])

return df

\end{minted}

oversampling

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

def oversampleDataset(df, columnName, valueToOversample, numberOfSamplesAfter):

if len(df[df[columnName] == valueToOversample]) < numberOfSamplesAfter:

engagedIndices = df[df[columnName] == valueToOversample].index

tempDf = df.loc[engagedIndices]

tempDfOversampled = resample(tempDf, replace=True, n\_samples=numberOfSamplesAfter, random\_state=69)

return pd.concat([df.drop(engagedIndices), tempDfOversampled])

return df

\end{minted}

Codice che chiama i metodi

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

def resampleDataset ():

df = pd.read\_csv(oldDatasetPath)

visualizeDataFrameChart(df)

numberOfValuesForEachLabel = 2000

labelsList = df["label"].unique()

for label in labelsList:

df = undersampleDataset (df, "label", label, numberOfValuesForEachLabel)

for label in labelsList:

df = oversampleDataset (df, "label", label, numberOfValuesForEachLabel)

visualizeDataFrameChart(df)

df.to\_csv(newDatasetPath, index=False)

\end{minted}

1. sklearn.utils.resample è una funzione fornita dalla libreria scikit-learn che viene utilizzata per generare esempi sintetici per l'oversampling di dataset. In particolare, la funzione resample prende in input un insieme di campioni e genera un nuovo insieme di campioni sintetici.

La funzione resample prende in input i seguenti parametri:

* X: un array o un dataframe che rappresenta le feature dei campioni
* y: un array o un dataframe che rappresenta le label dei campioni
* replace: un valore booleano che indica se l'oversampling deve essere fatto con o senza sostituzione (ovvero se gli esempi sintetici possono essere duplicati)
* n\_samples: il numero di esempi sintetici da generare
* random\_state: un valore intero che rappresenta il seed per la generazione casuale degli esempi sintetici

La funzione resample restituisce due oggetti:

* X\_resampled: un array o un dataframe contenente le feature dei campioni originali e dei nuovi esempi sintetici generati (unico output utilizzato)
* y\_resampled: un array o un dataframe contenente le label dei campioni originali e dei nuovi esempi sintetici generati

In sostanza, la funzione resample genera nuovi esempi sintetici aggiungendo variazioni minime ai campioni esistenti, in modo da produrre una distribuzione bilanciata delle classi nel dataset. Questi nuovi esempi sintetici vengono poi utilizzati insieme ai campioni esistenti per addestrare i modelli di machine learning.

1. pandas.DataFrame.sample è una funzione fornita dalla libreria pandas che viene utilizzata per estrarre casualmente un sottoinsieme di righe da un dataframe. In particolare, la funzione sample prende in input un dataframe e restituisce un nuovo dataframe contenente solo un sottoinsieme delle righe del dataframe originale.

La funzione sample prende in input i seguenti parametri:

* n: il numero di righe da estrarre casualmente
* frac: (non valorizzato) la frazione di righe da estrarre casualmente (ad esempio, 0.5 per estrarre il 50% delle righe)
* replace: (non valorizzato) un valore booleano che indica se le righe estratte devono essere selezionate con o senza sostituzione (ovvero se una stessa riga può essere selezionata più volte)
* weights: (non valorizzato) un array di pesi per ogni riga, utilizzato per selezionare le righe in modo ponderato
* random\_state: un valore intero che rappresenta il seed per la generazione casuale degli indici delle righe da selezionare

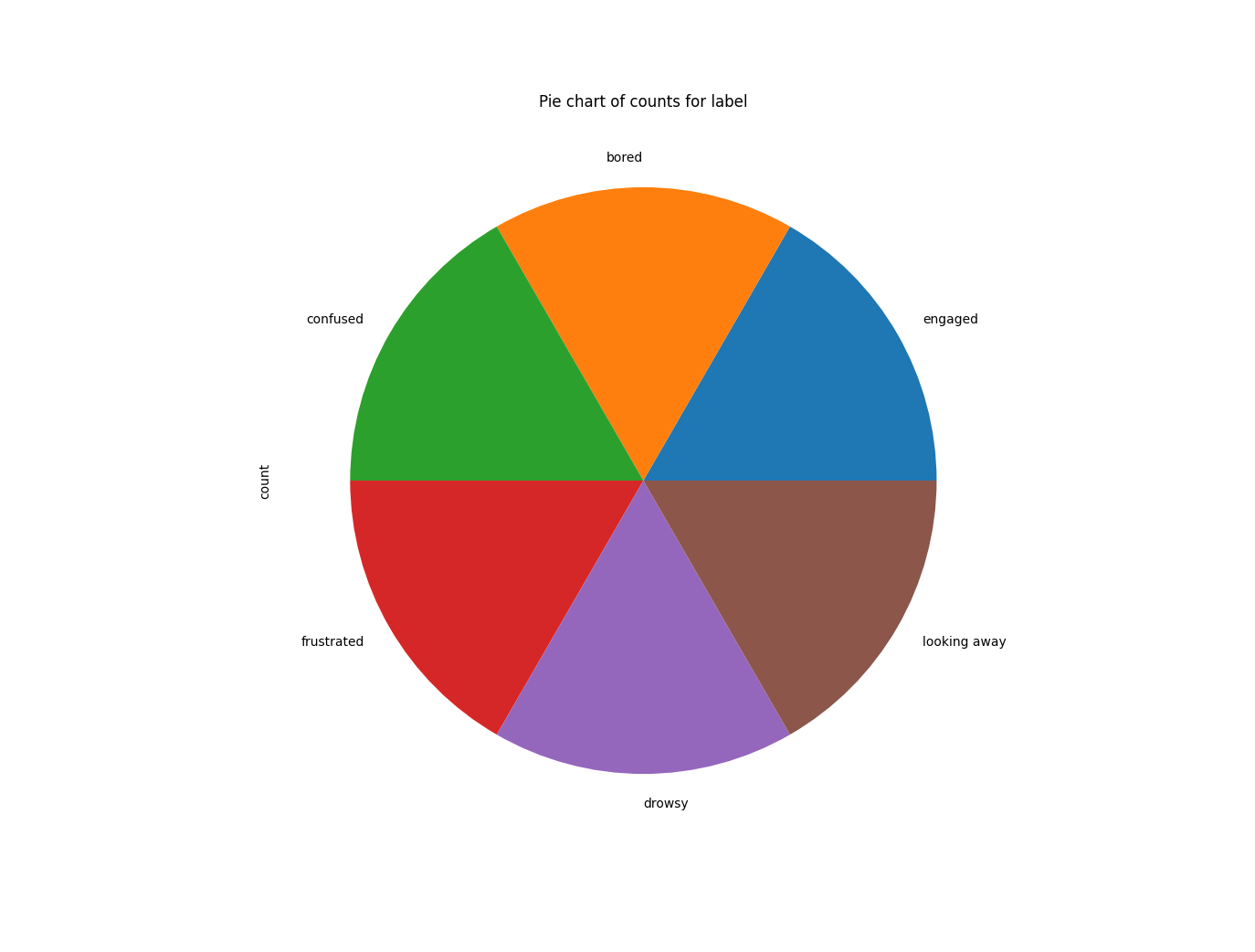
La funzione sample restituisce un nuovo dataframe contenente solo il sottoinsieme delle righe selezionate casualmente dal dataframe originale.

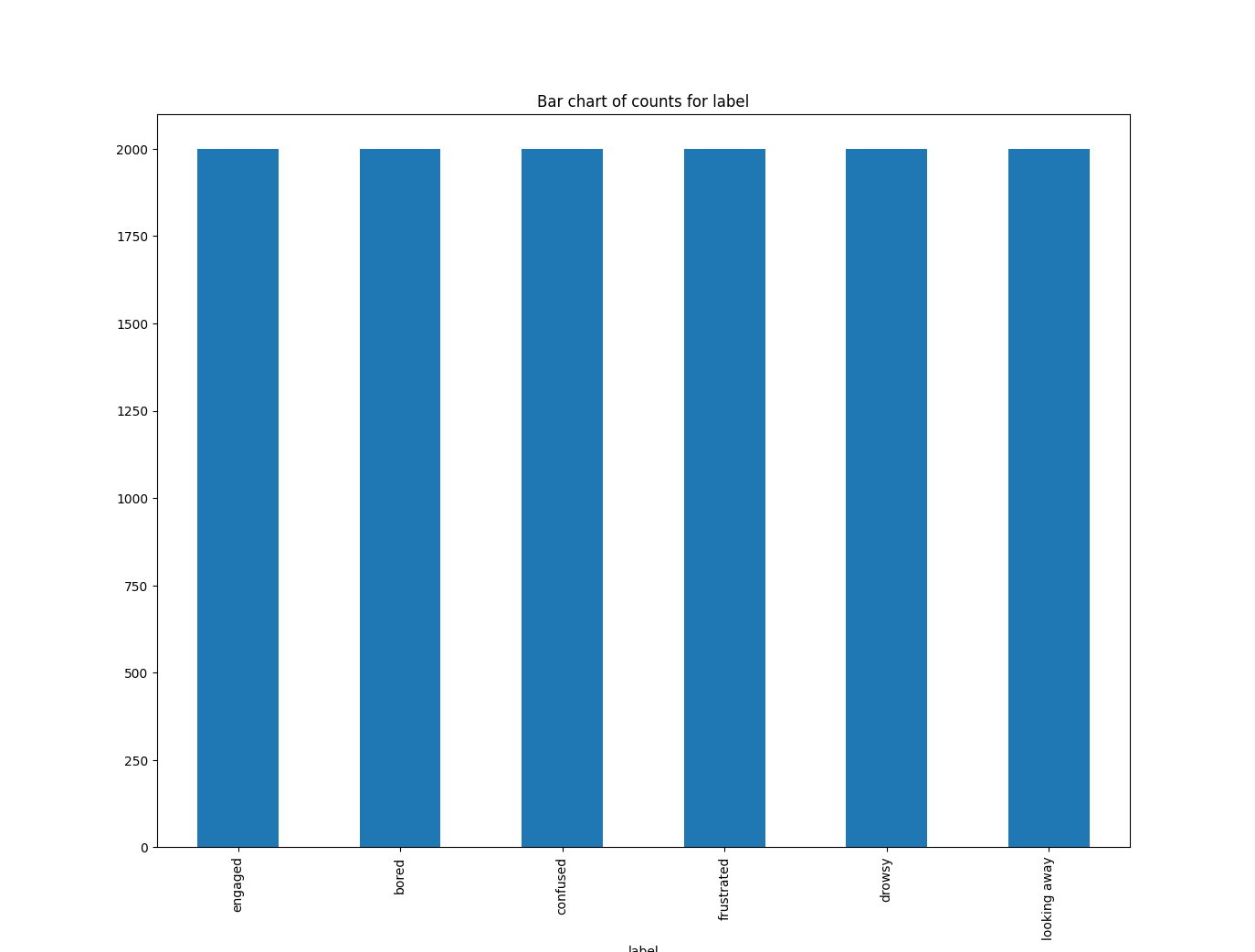
In sostanza, la funzione sample è utilizzata per ridurre la dimensione del dataframe originale, selezionando solo un sottoinsieme casuale delle righe. Questo può essere utile per ridurre i tempi di calcolo durante l'addestramento dei modelli di machine learning, in particolare quando il dataset originale è molto grande e non è necessario utilizzare tutte le righe per ottenere un buon modello.

Giusto per l’appunto, prima di effettuare il resampling del dataset ho provato ad effettuare delle analisi su nuove immagini utilizzando un classificatore generato dal dataset as-his e i risultati sono stati quelli aspettati, ovvero:  
Quasi sempre veniva rilevato che la persona ripresa nell’immagine aveva delle Action Units che portavano alla predizione “engaged” e le altre volte veniva rilevato lo stato d’animo “bored”, ignorando del tutto gli altri valori presenti per la colonna label.

Dopo aver effettuato vari test sono arrivato alla conclusione di effettuare un resample per ottenere 2000 istanze per ogni label in quanto questo sembra il numero di sample che mantiene più tranquillamente dei risultati e che li fa risultare più corenti nel tempo, i risultati relativi ottenuti sono riportati nell’ultimo capitolo.

Di seguito riporto i grafici riguardanti il dataset presentati all’inizio, generati dopo aver effettuato il resampling:





Altre funzionalità di py-feat