\chapter{Metodi}

Questo capitolo tratterà le tecnologie, gli strumenti, le librerie e gli algoritmi utilizzati per la realizzazione dei modelli predittivi e dell’interfaccia successivamente creata.

\section{Tecnologie utilizzate}

\subsection{Machine Learning}

Il Machine Learning, o apprendimento automatico, è un campo di studio che si occupa di sviluppare algoritmi in grado di perfezionarsi automaticamente mediante l'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 attuare predizioni o prendere decisioni senza essere esplicitamente programmati per farlo.

Esistono tre categorie di approcci di Machine Learning: l'apprendimento supervisionato, l'apprendimento non supervisionato e l'apprendimento per rinforzo.

* L'apprendimento supervisionato consiste nell’indicare al calcolatore una regola generale che mappi gli input e gli output desiderati.

L'algoritmo di apprendimento è provvisto di dati di input di esempio e degli output corrispondenti e, attraverso successive iterazioni, viene successivamente costruito un modello matematico che può essere sfruttato per predire l'output associato ad 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 necessariamente estrapolare le informazioni significative dai dati di input pur non essendo a conoscenza dell'output desiderato.

* L'apprendimento per rinforzo prevede che un programma interagisca con un ambiente dinamico con lo scopo di raggiungere un obiettivo specifico, ad esempio guidare un veicolo o vincere un gioco contro un avversario.

Durante le varie iterazioni il programma riceve un feedback sotto forma di premio e cerca di massimizzare il suo “punteggio”, in modo da imparare e raggiungere l'obiettivo prefissato.

\subsection{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 complesso meccanismo alla base dell'apparato visivo umano.

Questa 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.

Si avvale pertanto anche di conoscenze di geometria, fisica, statistica e della teoria dell'apprendimento per descrivere il mondo in modo sensato, producendo risultati che possono indirizzarci alla corretta linea d'azione.

\subsection{Cuda}

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

CUDA® è una piattaforma di calcolo parallelo, un modello di programmazione sviluppato da NVIDIA per il calcolo generale su unità di elaborazione grafica (GPU).

Per mezzo di CUDA, gli sviluppatori possono ampliare significativamente la velocità delle applicazioni di calcolo sfruttando la potenza delle GPU.

Nelle applicazioni dotate di 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 realizzata su migliaia di core GPU in parallelo.

Quando ci si avvale di CUDA, gli sviluppatori hanno la possibilità di programmare in linguaggi popolari come C, C++, Fortran, Python e MATLAB, esprimendo il 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.

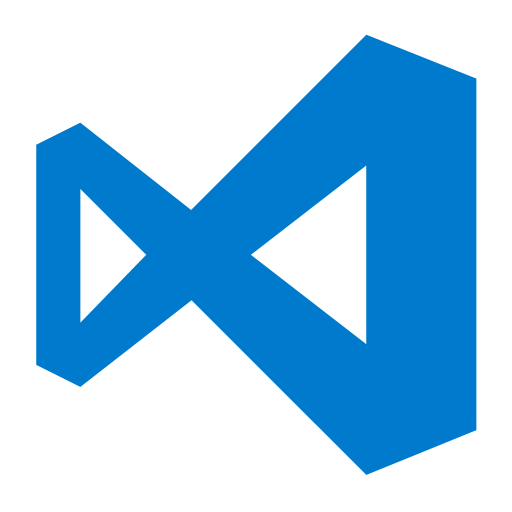
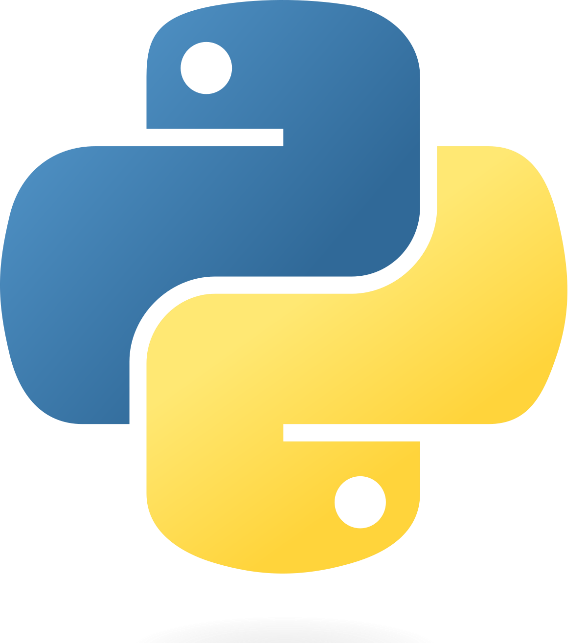


\section{Strumenti utilizzati}

\subsection{Visual Studio Code}

Visual Studio Code è un editor di testo 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 che presenta, oltre a diverse estensioni fornite direttamente da Microsoft, innumerevoli estensioni rese disponibili dalla community.

\section{Librerie utilizzate}

\subsection{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 mediante le sue strutture dati.

Il termine"Pandas" deriva dal termine econometrico "Panel Data", appellativo attribuito a un insieme di dati contenenti osservazioni sugli stessi individui durante più periodi di tempo.



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

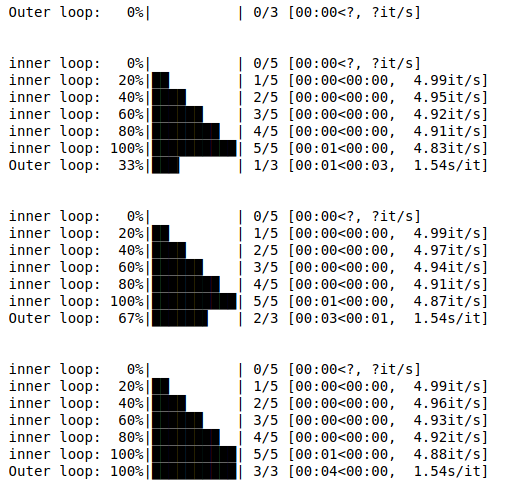
La libreria funziona su qualsiasi piattaforma ed è completamente indipendente dalle dipendenze.

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 di fondamentale importanza, in quanto ha permesso di gestire efficacemente l'elaborazione dei dati, evitando eventuali problemi tecnici, garantendo risultati accurati ed affidabili e portandomi a riconsiderare delle 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.



\subsection{Pickle}

La libreria pickle è molto versatile e può essere utilizzata per salvare e ripristinare qualsiasi tipo di oggetto Python, inclusi dizionari, liste, tuple, classi e istanze di oggetti personalizzati.

Inoltre, pickle supporta anche la serializzazione di oggetti multipli in un unico file, rendendo più semplice l'organizzazione dei dati.

La libreria offre anche diverse opzioni per controllare il comportamento della serializzazione, come la scelta del protocollo di serializzazione e la possibilità di escludere alcuni attributi dall'oggetto da serializzare.

Una caratteristica importante della libreria pickle è che gli oggetti serializzati possono essere utilizzati su diverse piattaforme e versioni di Python, purché il protocollo di serializzazione utilizzato sia compatibile.

Ciò significa che un oggetto serializzato su un computer Windows con Python 3.9 può essere deserializzato su un computer Linux con Python 2.7, ad esempio.

Tuttavia, è importante notare che non tutti gli oggetti possono essere serializzati correttamente, come ad esempio le funzioni e le istanze di oggetti di alcune librerie Python.

Inoltre, la compatibilità tra diverse versioni di Python non è garantita in tutti i casi; quindi, è necessario prestare attenzione a eventuali incompatibilità.

Immagine che contiene testo, cartone animato, schermata, Cartellone

Descrizione generata automaticamente

\subsection{Torch}

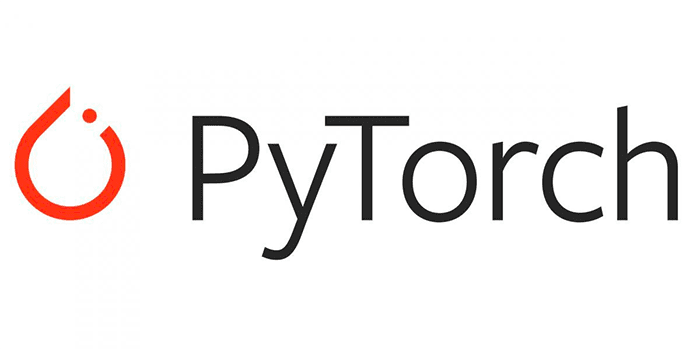
PyTorch è un popolare framework open-source di deep learning che consente agli sviluppatori di creare modelli di intelligenza artificiale in modo rapido ed efficiente. È stato sviluppato originariamente da Facebook AI Research, guadagnandosi una grande popolarità per la sua facilità d'utilizzo, la sua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

Una delle sue principali caratteristiche è la sua architettura a flusso di dati (data flow), la quale rende il framework particolarmente adatto per le applicazioni di deep learning.

Inoltre, PyTorch è dotato di un'ampia gamma di librerie e strumenti come PyTorch Lightning, che mirano a semplificare lo sviluppo di modelli di intelligenza artificiale.

Questa libreria è anche conosciuta per la sua flessibilità e scalabilità, in quanto permette di creare modelli di deep learning sia per computer singoli che per cluster di computer.

In aggiunta, supporta una vasta gamma di piattaforme hardware, come CPU, GPU e TPU, il che la rende adatta per le applicazioni in ambiti come il machine learning, la visione artificiale e il linguaggio naturale.



\subsection{OpenCV}

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

OpenCV (Open Source Computer Vision Library) è una libreria software open source finalizzata all’ambito della computer vision e del machine learning.

È stata ideata 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 cluster 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,
* 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.

Oltre alle aziende ben consolidate come Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda e Toyota, esistono molte startup come Applied Minds, VideoSurf e Zeitera che ne fanno un uso intensivo.

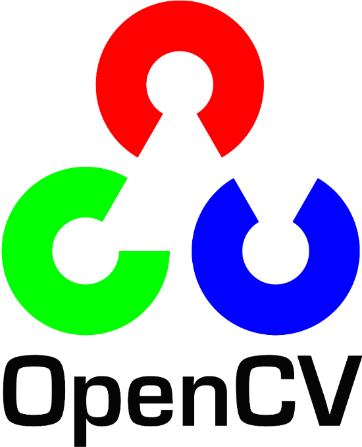
Esso è impiegato per molteplici applicazioni, tra cui unire immagini di Street View, rilevare intrusioni in video di sorveglianza in Israele, monitorare l'equipaggiamento minerario in Cina, aiutare i robot a navigare e raccogliere oggetti presso Willow Garage, rilevare gli incidenti di annegamento in 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.

È provvisto di interfacce per C++, Python, Java e MATLAB e supporta Windows, Linux, Android e Mac OS.

Attualmente sono in sviluppo interfacce complete per CUDA e OpenCL.

Ci sono oltre 500 algoritmi e circa 10 volte tante funzioni che compongono o supportano questi ultimi.

OpenCV è scritto nativamente in C++ e ha un'interfaccia template che funziona perfettamente con i contenitori STL.



\subsection{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.

Essa 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.

Il progetto scikit-learn è nato come 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, hanno preso il comando del progetto rilasciando in seguito 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" meglio conservate e popolari.

Nel 2019 si è poi stimato che scikit-learn fosse 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 dei dati, NumPy per la vettorizzazione degli array, Pandas dataframes, SciPy e molte altre.



\section{Algoritmi utilizzati per la creazione dei modelli predittivi}

\subsection{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 creato da Leo Breiman e Adele Cutler, che combina l'output di più alberi decisionali col fine di raggiungere un singolo risultato.

La sua predisposizione intuitiva e la sua flessibilità hanno nutrito l’aumento della sua adozione, in quanto gestisce sia problemi di classificazione che di regressione.

\subsubsection{Alberi decisionali}

Dato che il modello di random forest è composto da più alberi decisionali, è utile descrivere brevemente l'algoritmo dell'albero decisionale.

Gli alberi decisionali partono da una domanda di base, ad esempio "Dovrei fare surf?".

A partire da ciò 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 di ripartizione dei dati.

Ogni domanda aiuta l’albero a giungere a una decisione finale, indicata dal nodo foglia raggiunto.

Le osservazioni che soddisfano i criteri seguiranno il ramo "Sì", mentre 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.

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 particolar modo quando i singoli alberi non sono correlati tra loro.

\subsubsection{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 features per creare una foresta di alberi decisionali non correlati.

La casualità delle features, altrimenti nota come bagging delle caratteristiche o "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 random forest, poiché mentre gli alberi decisionali considerano tutte le possibili suddivisioni delle caratteristiche, le random forest selezionano solo un sottoinsieme di quelle caratteristiche.

Tornando 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 un altro utente.

Tenendo conto di tutta la potenziale variabilità dei dati, possiamo ridurre il rischio di overfitting, di bias e di varianza complessiva, ottenendo previsioni via via più precise.

\subsubsection{Come funziona}

Gli algoritmi delle random forest hanno tre iperparametri principali da impostare prima dell'allenamento.

Questi includono:

* la dimensione del nodo,
* il numero di alberi,
* il numero di caratteristiche campionate

Da lì, il classificatore della random forest può essere utilizzato per risolvere problemi di regressione o di classificazione.

L'algoritmo della random forest è costituito da una collezione di alberi decisionali, dove 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, ad esempio, un terzo viene messo da parte come dati di test, noti come campione fuori dalla borsa (oob), su cui ci soffermeremo in seguito.

Un'altra istanza di casualità viene quindi iniettata attraverso il bagging delle caratteristiche, incrementando la diversità nel dataset e riducendo la correlazione tra gli alberi decisionali.

A seconda del tipo di problema, la determinazione della previsione subirà variazioni; 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.

\subsubsection{Benefici e sfide del random forest}

Ci sono diversi vantaggi e sfide 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: gli 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 è presente un largo numero di alberi di decisione in un random forest, il classificatore non sovrastimerà il modello, poiché la media di alberi non correlati fra loro riduce la variazione complessiva e l'errore di previsione.

* + Flessibilità: il random forest può gestire sia compiti di regressione che di classificazione con un elevato grado di precisione, distinguendosi in quanto metodo popolare tra i data scientist.

Inoltre, la feature bagging rende il classificatore random forest uno strumento adeguato per la stima dei valori mancanti, in quanto mantiene l'accuratezza quando una parte dei dati è irreperibile.

* + Facile determinazione dell'importanza delle feature: il random forest rende facile valutare l'importanza delle variabili, o del contributo, al modello.

Sussistono 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.

* Principali sfide
  + È un processo che richiede tempo: gli algoritmi random forest possono gestire grandi set di dati e fornire previsioni più accurate; tuttavia il processo è rallentato dalla computazione dei dati per ogni singolo albero decisionale.
  + Richiede più risorse: le random forest, elaborando set di dati più grandi, richiedono più risorse per archiviare questi ultimi.
  + Più complesso: la previsione di un singolo albero decisionale è più facile da interpretare rispetto a una foresta di questi.

\subsection{K-nearest neighbors}

[da <https://www.ibm.com/it-it/topics/knn>]

Immagine che contiene testo, schermata, linea, Carattere

Descrizione generata automaticamente

L'algoritmo k-nearest neighbors, noto anche come KNN o k-NN, è un classificatore di apprendimento supervisionato non parametrico.

Esso sfrutta la prossimità per effettuare classificazioni o previsioni sul raggruppamento di un singolo punto dati.

Sebbene possa essere utilizzato per problemi di regressione o classificazione, viene generalmente impiegato in quanto algoritmo di classificazione, basandosi sul presupposto che dati simili, se analizzati o rappresentati nella giusta maniera, possono essere trovati l'uno vicino all'altro.

Per i problemi di classificazione un'etichetta di classe viene assegnata sulla base di un voto a maggioranza(ad es. viene utilizzata l'etichetta più frequentemente rappresentata attorno a un determinato punto dati).

I problemi di regressione utilizzano un concetto simile al problema di classificazione; in questo caso però viene presa la media dei k elementi vicini più vicini per effettuare una previsione su una classificazione.

Qui, ciò che si discosta notevolmente, è il fatto che la classificazione viene utilizzata per i valori discreti, mentre la regressione viene utilizzata per quelli continui.

Tuttavia, prima di poter effettuare una classificazione, è necessario definire il concetto di distanza.

La distanza euclidea, altra metrica di distanza popolare, misura il valore assoluto tra due punti.

Vale la pena notare che l'algoritmo KNN fa anche parte di una famiglia di modelli di "apprendimento pigro", il che significa che memorizza solo un set di dati di addestramento nella fase di addestramento.

Perviene quindi che tutto il calcolo avvenga quando si compie una classificazione o una previsione.

Poiché fa ampiamente affidamento sulla memoria per archiviare tutti i suoi dati di addestramento, viene anche definito metodo di apprendimento basato su istanze o basato sulla memoria.

Le idee iniziali sul modello KNN sono attribuite a Evelyn Fix e Joseph Hodges in questo articolo [TODO trova articolo] del 1951, mentre Thomas Cover ne amplia il concetto nella sua ricerca, “Nearest Neighbor Pattern Classification.” [TODO trova articolo]

Pur non riscuotendo un tale successo come in precedenza, è ancora uno dei primi algoritmi che si affronta nello studio della data science per la sua semplicità ed accuratezza.

Tuttavia, al crescere di un set di dati KNN diventa sempre più inefficiente, compromettendo le prestazioni del modello.

Viene riscontrato il suo impiego per semplici sistemi di raccomandazione, riconoscimento di modelli, data mining, previsioni dei mercati finanziari, rilevamento delle intrusioni e altro ancora.

\subsubsection{Calcolare il KNN: metriche di distanza}

Ricapitolando, l'obiettivo dell'algoritmo k-nearest neighbor è identificare i vicini più prossimi di un dato punto di query, in modo da poter assegnare un'etichetta di classe a quel punto.

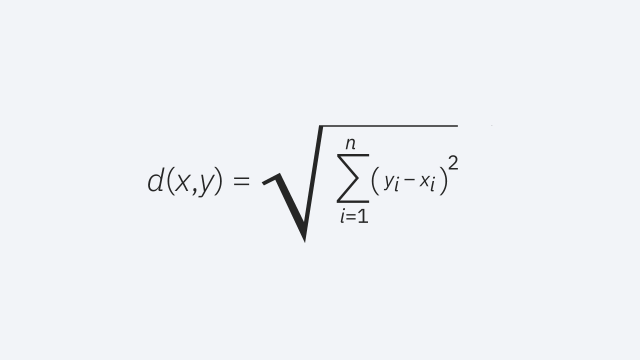
Per determinare quali punti dati sono più attigui ad un determinato punto di query, sarà necessario calcolare la distanza tra il punto di interrogazione e gli altri punti dati.

Le metriche di distanza aiutano a formare confini decisionali che suddividono i punti di query in regioni diverse.

Sebbene sussistano diverse misure di distanza, tra cui è possibile scegliere, l’articolo sul sito di IBM tratta esclusivamente quelle a seguire:

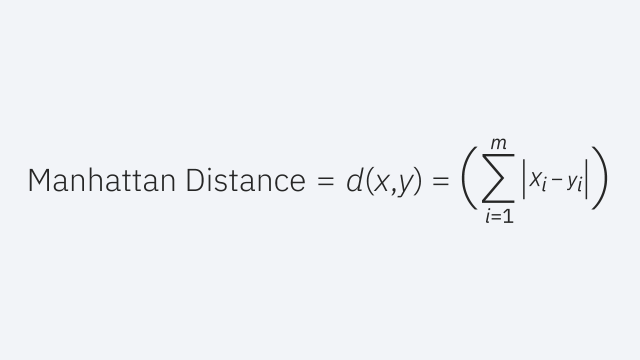
* Distanza euclidea (p=2): la misura della distanza più comunemente adottata; essa è limitata ai vettori con valori reali.

Utilizzando la formula seguente traccia una linea retta tra il punto di query e l'altro punto che si vuole misurare.



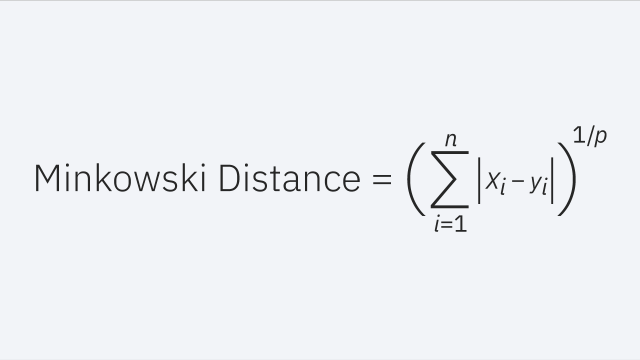
* Distanza di Manhattan (p=1): un'altra metrica di distanza particolarmente nota che si propone di misurare il valore assoluto tra due punti.

è anche riconosciuta come distanza del taxi o distanza del blocco cittadino, poiché è comunemente visualizzata con una griglia che illustra come si potrebbe percorrere il tragitto da un indirizzo all'altro attraverso le strade della città.



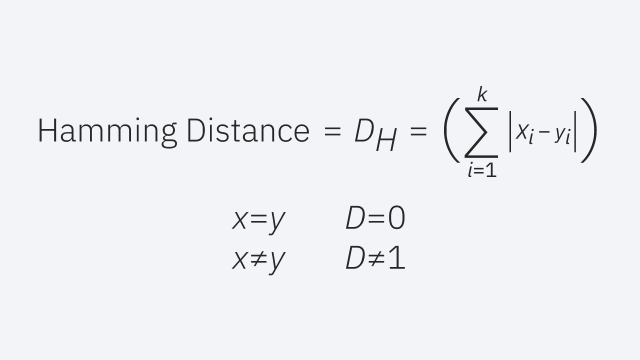
* Distanza di Minkowski: questa misura della distanza è la forma generalizzata delle metriche di distanza euclidea e di Manhattan. Il parametro, p, nella formula seguente, consente la creazione di altre metriche di distanza.

La distanza Euclidea è rappresentata da questa formula quando p è uguale a due e la distanza di Manhattan è indicata con p uguale a uno.

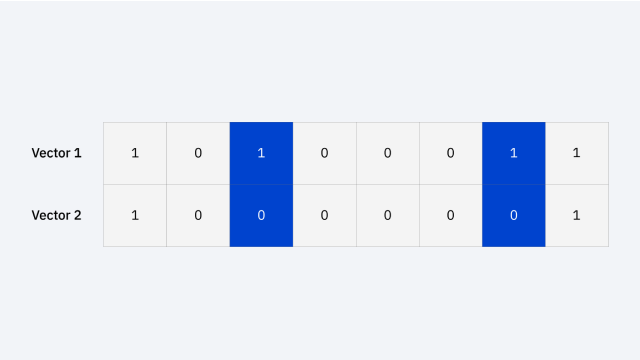


* Distanza di Hamming: questa tecnica viene utilizzata tipicamente con vettori booleani o stringa, identificando i punti in cui i vettori non trovano corrispondenza.

Di conseguenza, è stata anche definita metrica di sovrapposizione.



Ad esempio, se avessi le seguenti stringhe, la distanza di Hamming sarebbe due poiché solo due dei valori differiscono.



\subsubsection{Calcolare KNN: definizione di k}

Il valore k nell'algoritmo k-NN definisce quanti vicini verranno controllati per determinare la classificazione di un punto di query specifico.

Di fatti, se k=1, l'istanza verrà assegnata alla stessa classe del suo singolo neighbors più vicino.

Definire k può essere un atto di bilanciamento, in quanto valori diversi possono portare a overfitting o underfitting.

Valori inferiori a k possono essere caratterizzati da una variabilità elevata ma una bassa distorsione, mentre valori maggiori di k possono portare a una distorsione elevata e una variabilità inferiore.

La scelta di k dipenderà in particolar modo dai dati di input, dal momento che i dati con più valori anomali o rumore probabilmente opereranno in modo più proficuo tanto più elevati sono i valori di k.

In generale si consiglia di adoperare un numero dispari per k, col fine di evitare pareggi nella classificazione.

\subsubsection{Applicazioni di k-NN nell'apprendimento automatico}

L'algoritmo k-NN è stato utilizzato all'interno di una varietà di applicazioni, in maggior misura all'interno della classificazione. Alcuni di questi casi d'uso includono:

* Pre-elaborazione dei dati: poichè i dataset presentano spesso valori mancanti, l'algoritmo KNN può stimare tali valori in un processo noto come imputazione dei dati mancanti.
* Reccomander systems: adoperando i dati del flusso di clic dai siti web, l'algoritmo KNN è stato utilizzato per fornire consigli automatici agli utenti su contenuti aggiuntivi.

Da tale ricerca emerge che un utente è assegnato a un particolare gruppo e, sulla base del comportamento di quest'ultimo, riceve un consiglio.

Tuttavia, dati i problemi di scalabilità con KNN, questo approccio potrebbe non risultare ottimale in caso di impiego di dataset più grandi.

* Finanza: è stato utilizzato anche in una varietà di casi di utilizzo finanziari ed economici.

Ipoteticamente, un articolo mostra in che modo l'utilizzo di KNN sui dati di credito può aiutare le banche nella valutazione dei rischi su un prestito a un'organizzazione o a un individuo. Viene quindi sfruttato per determinare l'affidabilità creditizia del richiedente di tale prestito.

Un altro articolo ne sottolinea l'uso nelle previsioni del mercato azionario, nei tassi di cambio, nel trading di fetures e nelle analisi sul riciclaggio di denaro.

* Assistenza sanitaria: KNN ha riscontrato applicazioni anche nel settore dell'assistenza sanitaria, effettuando previsioni sul rischio di infarto e cancro alla prostata.

L'algoritmo funziona calcolando le espressioni geniche più probabili.

* Riconoscimento dei pattern: KNN ha anche aiutato a identificare i pattern, come nel testo e nella classificazione digitale. Ciò è stato particolarmente utile per identificare i numeri scritti a mano in cui ci si potrebbe imbattere su moduli o buste postali.

\subsubsection{Vantaggi e svantaggi dell'algoritmo KNN}

K-NN possiede i suoi punti di forza e di debolezza. A seconda del progetto e dell'applicazione, potrebbe rivelarsi o meno la scelta giusta.

* Vantaggi
  + Facile da implementare: data la semplicità e l'accuratezza dell'algoritmo, è uno dei primi classificatori che un data scientist alle prime armi apprenderà.
  + Si adatta facilmente all'aggiunta di nuovi campioni di addestramento, l'algoritmo si adatta per tenere conto di eventuali nuovi dati, a fronte dell'archivio di tutti i dati di addestramento in memoria.
  + Pochi iperparametri: KNN necessita solo di un valore k e una metrica di distanza, il che, a confronto con altri algoritmi di machine learning, è minore in quantità.
* Svantaggi
  + Non è provvisto di una buona scalabilità: essendo KNN un algoritmo cosiddetto pigro, occupa più memoria e spazio di storage dei dati rispetto ad altri classificatori.

Sebbene diverse strutture di dati ,come Ball-Tree, siano state create per affrontare le inefficienze computazionali, un classificatore diverso potrebbe dimostrarsi ideale.

* + Maledizione della dimensionalità: l'algoritmo tende a cadere vittima della “maledizione della dimensionalità”; ciò significa che non ricopre adeguatamente il proprio ruolo con input di dati ad alta dimensionalità.

Questo è a volte indicato anche come il fenomeno del picco, nel quale, dopo che l'algoritmo raggiunge l'ottimale numero di funzioni, le funzioni aggiuntive aumentano la quantità di errori di classificazione, soprattutto quando la dimensione del campione è inferiore.

* + Propensione al sovradimensionamento dei dati: a causa della "maledizione della dimensionalità", KNN è anche più propenso al sovradimensionamento dei dati.

Sebbene le tecniche di selezione delle caratteristiche e di riduzione della dimensionalità vengano sfruttate per evitare che ciò accada, il valore di k può inevitabilmente influire sul comportamento del modello.

Valori più bassi di k possono sovra-alimentare i dati, mentre valori più alti di k tendono a "smussare" i valori di previsione, poiché si sta attuando la media dei valori su un'area o un neighborhood più grande.

Tuttavia, se il valore di k è troppo alto, può essere inferiore ai dati.

TODO riprendi correzione con asia da qui

\subsection{Naive Bayes Classifier}

[da https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c]

Un classificatore di Bayes ingenuo è un modello di apprendimento automatico probabilistico utilizzato per compiti di classificazione. Il fulcro del classificatore si basa sul teorema di Bayes.

Teorema di Bayes:

Utilizzando il teorema di Bayes, possiamo trovare la probabilità che A accada, conseguentemente all’avvenimento di B. Qui, B è la prova e A è l'ipotesi. L'assunzione alla base del teorema è che i predittori/ caratteristiche siano indipendenti: in altre parole, la presenza di una particolare caratteristica non inﬂuisce sull'altra. Pertanto, viene riconosciuto in quanto ingenuo.

Immagine che contiene testo, Carattere, linea, bianco

Descrizione generata automaticamente

Volendo oﬀrire un esempio al fine di una migliore comprensione, consideriamo il problema di giocare a golf. Il dataset è rappresentato come segue.

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

Classifichiamo se la giornata si configura adatta per una partita di golf, basandoci sugli attributi di tale giornata. Le colonne rappresentano questi attributi, mentre le righe rappresentano le singole voci.

Prendendo a campione la prima riga del dataset, possiamo osservare che non è adatta per giocare a golf se il cielo è nuvoloso, la temperatura è calda, l'umidità è alta e non c'è vento. Qui facciamo due assunzioni:

come già detto, consideriamo che questi predittori siano indipendenti (ovvero, se la temperatura è calda, non significa necessariamente che l'umidità sia alta). Un'altra assunzione fatta qui è che tutti i predittori abbiano un eﬀetto uguale sul risultato. Cioè, il fatto che ci sia vento non ha una maggiore importanza nella decisione di giocare a golf o meno. (?.???.???? NON CAPISCO!FORMULA MEGLIO)

Rifacendosi al secondo esempio, il teorema di Bayes può essere quindi riscritto come segue:

Immagine che contiene testo, Carattere, bianco, linea

Descrizione generata automaticamente

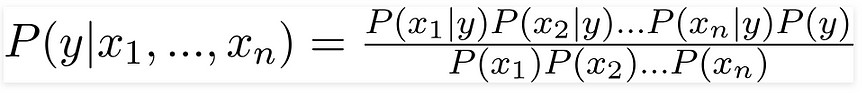
La variabile y è la variabile di classe (giocare a golf), la quale rappresenta quanto sia adatta la giornata o meno per giocare a golf. La variabile X rappresenta i parametri/caratteristiche.

X è dato come segue:

Immagine che contiene Carattere, testo, calligrafia, linea

Descrizione generata automaticamente

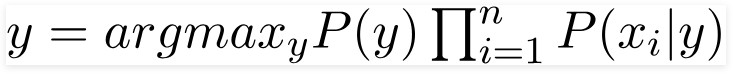
Qui x\_1, x\_2... x\_n raffigurano le caratteristiche, cioè possono essere mappati su cielo, temperatura, umidità e vento. Sostituendo X ed espandendo utilizzando la regola della catena otteniamo:



Ora è possibile ottenere i valori per ognuno guardando il dataset e sostituendoli nell'equazione. Per tutte le voci nel dataset, il denominatore non cambia, rimane statico. Pertanto, il denominatore può essere rimosso e può essere introdotta una proporzionalità.



Nel nostro caso, la variabile di classe (y) ha solo due risultati, sì o no. Potrebbero esserci casi in cui la classificazione potrebbe rivelarsi multivariata. Pertanto, sarà necessario trovare la classe y con la massima probabilità.



Utilizzando questa funzione ci è possibile ottenere la classe, dato il predittore.

Tipi di Classificatori Bayesiani Ingenui:

Classificatore Bayesiano Multinomiale:

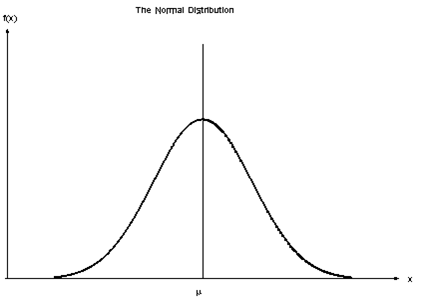
Esso è principalmente utilizzato per problemi di classificazione dei documenti; ad esempio, se un documento appartiene alla categoria di sport, politica, tecnologia, ecc. Le caratteristiche/predittori impiegati dal classificatore saranno la frequenza delle parole presenti nel documento.

Classificatore Bayesiano di Bernoulli:

Questo classificatore è simile al precedentemente citato, pur essendo i predittori variabili booleane. I parametri che usiamo per prevedere la variabile di classe assumono solo valori sì o no (se una parola compare nel testo o meno).

Classificatore Bayesiano Gaussiano:

Quando i predittori assumono un valore continuo e non discreto, assumiamo che questi valori siano estratti da una distribuzione gaussiana.



Poiché il modo in cui i valori sono presenti nel dataset cambia, la formula per la probabilità condizionata cambia in questo caso,

Immagine che contiene testo, Carattere, calligrafia, linea

Descrizione generata automaticamente

Conclusione:

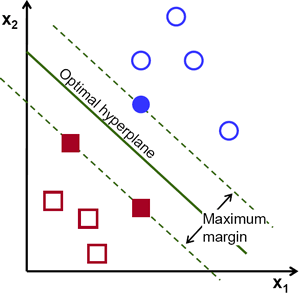
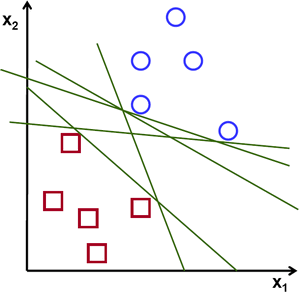
Gli algoritmi Bayesiani Ingenui sono principalmente utilizzati nell'analisi del sentiment, nel filtraggio dello spam, nei sistemi di raccomandazione, ecc. Sono veloci ed facili da implementare, ma il loro più grande svantaggio è la necessità che i predittori siano indipendenti. Nella maggior parte dei casi reali, i predittori sono dipendenti, il che ostacola le prestazioni del classificatore.

Support Vector Machine

Il support vector machine è un altro algoritmo semplice che ogni esperto di machine learning dovrebbe far rientrare nel proprio arsenale. Il support vector machine è favorito maggiormente data la produzione di una significativa accuratezza mediante una minor potenza di calcolo. Il Support Vector Machine, abbreviato come SVM, può essere utilizzato sia per compiti di regressione che di classificazione. Tuttavia, viene ampiamente sfruttato negli obiettivi di classificazione.

Cosa è il Support Vector Machine?

L'obiettivo dell'algoritmo support vector machine è quello di trovare un iperpiano in uno spazio N- dimensionale (N - il numero di caratteristiche) che classifichi distintamente i punti dati.



Per separare le due classi di punti dati sono presenti molti iperpiani possibili da cui poter scegliere. Il nostro obiettivo è trovare un piano che possegga il margine massimo, ovvero la massima distanza tra i punti dati di entrambe le classi. Massimizzare la distanza del margine fornisce un rinforzo in modo che i futuri punti dati possano essere classificati con maggiore fiducia.

Iperpiani e support vector

Immagine che contiene diagramma, linea, testo

Descrizione generata automaticamente

Gli iperpiani sono i confini decisionali che aiutano a classificare i punti dati. I punti dati che cadono su entrambi i lati dell'iperpiano possono essere attribuiti a diverse classi. Inoltre, la dimensione dell'iperpiano dipende dal numero di caratteristiche. Se il numero di caratteristiche di input è 2, l'iperpiano è solo una linea. Se il numero di caratteristiche di input è 3, l'iperpiano diventa un piano bidimensionale. Diventa difficile immaginare quando il numero di caratteristiche supera 3.

Immagine che contiene diagramma, linea, schermata, Diagramma

Descrizione generata automaticamente

I support vector sono i punti dati più vicini all'iperpiano e inﬂuenzano la posizione e l'orientamento di quest’ultimo . Impiegando questi support vector, massimizziamo il margine del classificatore. L'eliminazione dei support vector cambierà la posizione dell'iperpiano. Questi sono i punti che supportano la costruzione del nostro SVM.

Intuizione del grande margine

Nella regressione logistica, prendiamo l'output della funzione lineare e schiacciamo il valore nell'intervallo di [0,1] utilizzando la funzione sigmoide. Se il valore schiacciato è maggiore di un valore soglia (0,5), gli assegnamo una etichetta 1, altrimenti gli assegnamo un'etichetta 0. Nell'SVM, prendiamo l'output della funzione lineare e se quell'output è maggiore di 1, lo identifichiamo con una classe e se l'output è -1, lo identifichiamo con un'altra classe. Poiché i valori della soglia sono cambiati in 1 e -1 nell'SVM, otteniamo questo intervallo ([-1,1]) che viene utilizzato come margine.

Funzione di costo e aggiornamenti del gradiente

Nell'algoritmo SVM, cerchiamo di massimizzare la distanza tra i punti dati e l'iperpiano. La funzione di perdita che ci aiuta a massimizzare la distanza è la hinge loss.

Immagine che contiene testo, Carattere, calligrafia, bianco

Descrizione generata automaticamente

Il costo è 0 se il valore previsto e il valore eﬀettivo hanno lo stesso segno. In caso contrario, calcoliamo il valore di perdita. Aggiungiamo anche un parametro di regolarizzazione alla funzione di costo. L'obiettivo del

parametro di regolarizzazione è di bilanciare la massimizzazione della distanza con la perdita. Dopo aver aggiunto il parametro di regolarizzazione, la funzione di costo appare come segue.

Immagine che contiene Carattere, testo, bianco, tipografia

Descrizione generata automaticamente

Ora che abbiamo acquisito la funzione di perdita, utilizziamo le derivate parziali rispetto ai pesi per trovare i gradienti. Così facendo, possiamo aggiornare i nostri pesi.

Immagine che contiene testo, Carattere, calligrafia, bianco

Descrizione generata automaticamente

Quando non vi è alcuna errata classificazione, ovvero il nostro modello prevede correttamente la classe del nostro punto dati, sarà necessario aggiornare solo il gradiente dal parametro di regolarizzazione.



Quando ci si trova di fronte ad una errata classificazione, ossia il nostro modello commette un errore sulla previsione della classe del nostro punto dati, includiamo la perdita insieme al parametro di regolarizzazione per eseguire l'aggiornamento del gradiente.



\chapter{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.

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

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

* enagegd con 55707 samples
* bored con 16086 samples
* confused con 1041 samples
* looking away con 409 samples
* frustated con 893 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 dell’Action Unit passato in input al metodo (non riportato interamente in quanto prevede azioni preliminari trascurabili) sia presente e, successivamente, in caso fosse provvista di un valore maggiore o uguale a 0.5 (il range di valori è fra 0 e 1) restituisce la stringa che ho descritto nello spazio sottostante; in caso contrario consegnerà una stringa vuota.

La frase restituita dall’algoritmo, nel caso in cui il valore sia maggiore o uguale a cinque, è composta dal nome FACS della relativa Action Unit, con la successiva aggiunta del muscolo analizzato da questa Action Unit e il valore che è stato prelevato.

La frase presente nel dataset per ognuno dei samples è il risultato del concatenamento delle frasi generate per ogni Action Unit 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 nel codice, durante la creazione dell’oggetto Detector è possibile specificare il parametro \mintinline[bgcolor=bg]{python}{device}, permettendo l’esecuzione delle operazioni utilizzando la tecnologia cuda.

Per controllare che sia effettivamente possibile utilizzare questa funzionalità è stata usata la libreria torch per python.

Il parametro face\_model imposta il modello di rilevamento del viso da utilizzare. Ho ritenuto opportuno impostarlo su "retinaface", popolare modello di rilevamento del viso che utilizza una CNN (Convolutional Neural Networks).

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

Il parametro au\_model prepara il modello utilizzato alla 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 avvia il modello utilizzato per la rilevazione delle emozioni dalle espressioni facciali, ovvero "resmasknet", implementato utilizzando il lavoro di [20].

Il parametro facepose\_model stabilisce il modello utilizzato per la stima della posa della testa "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 da me utilizzate poiché 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 traslocati nel formato json mediante il metodo \mintinline[bgcolor=bg]{python}{detector.detect\_image(imagePath).to\_json()}, aggregati e salvati su un file, sempre in questo formato, così da poterli mostrare più chiaramente; 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} fa riferimento 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 optato per l’estrazione di un’immagine per ogni secondo di video, scrivendo 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 organizzate in modo diverso rispetto alle analisi per le immagini, in quanto ognuno dei campi citati prima (FaceRectX, FaceRectY, …) contengono 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) questi ultimi 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 riscontrato difficoltà nel riconoscere il volto della persona presente nel video, o questa non era presente all’interno dell’immagine; ciò 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 eseguito il seguente codice:

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

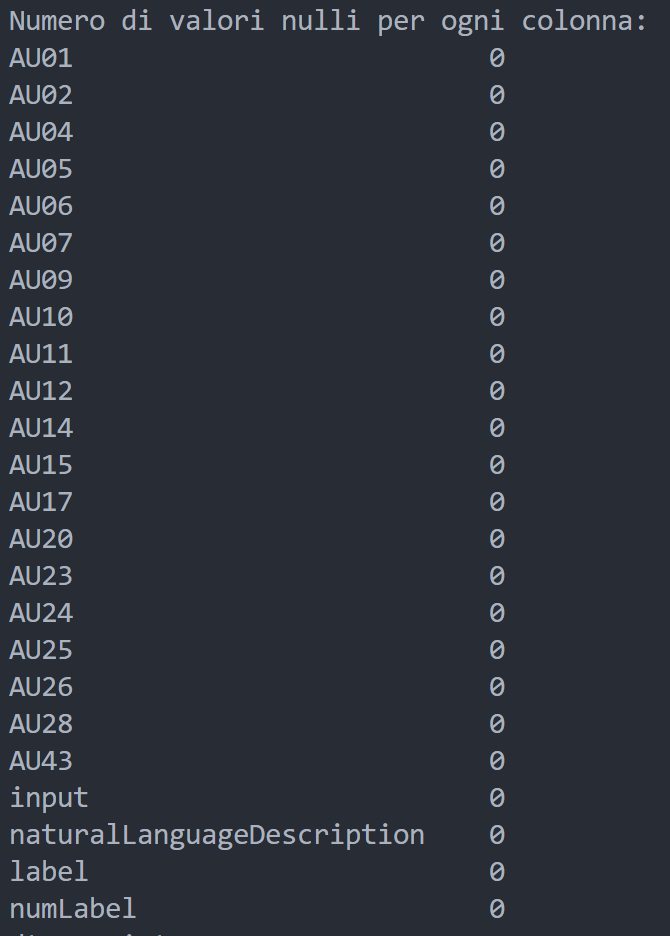
nullVals = df.isnull()

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

print(nullVals.sum())

\end{minted}

che dà in output:



Come è possibile osservare dall’immagine, il dataset risultato 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); di conseguenza ho 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

\chapter{Modelli predittivi utilizzati}

Random forest classifier

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

Per la creazione del classificatore ho in primis letto il file csv contenente il dataset pre-elaborato (modifiche precedentemente trattate e sulle quali aggiungerò considerazioni, in quanto è risultato efficace applicarne ulteriori per migliorare la precisione del predittore creato), rimosso le colonne non necessarie e, infine, ho diviso il dataset in set di addestramento e di test usando la funzione train\_test\_split della libreria sklearn.model\_selection; tramite 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 creato il classificatore utilizzando l’oggetto a disposizione fornito dalla libreria sklearn.ensable.

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

Il metodo è il seguente:

\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}

Eseguendo dei test attraverso il metodo \mintinline[bgcolor=bg]{python}{score(Xtest, yTest)} offerto dal random forest classifier generato dalla libreria sklearn.ensemble ho potuto calcolare l’accuracy del modello da me generato.

I risultati di questa analisi sono che il random forest classifier ottiene una precisione dell’82,3periodico% sul dataset di test generato nel metodo \mintinline[bgcolor=bg]{python}{getXtrainYTrain}



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

Immagine che contiene grafico, grafico a torta

Descrizione generata automaticamente

Come è possibile constatare, 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 largamente la predizione effettuata dal modello.

-----

K-nearest Neighbors Classifier

Un ulteriore classificatore realizzato per effettuare predizioni sul dataset è il K-nearest Neighbors Classifier, attraverso il quale ho effettuato delle query fornendo i dati delle Action Units da nuove immagini.

Come per il random forest classifier, per la creazione del classificatore ho in primis letto il file csv contenente il dataset pre-elaborato, rimosso le colonne non necessarie e, infine, ho diviso il dataset in set di addestramento e di test usando la funzione train\_test\_split della libreria sklearn.model\_selection; tramite l’output di questo metodo ho ricavato i pandas’s dataframes Xtrain, Xtest, yTrain, yTest.

Il metodo sopracitato per l’estrazione di questi dataframe è lo stesso riportato in pochi paragrafi precedenti.

Una volta ottenuti questi dataset ho creato il classificatore utilizzando l’oggetto a disposizione fornito dalla libreria sklearn.neighbors.

Il K-nearest Neighbors Classifier viene creato impostando il parametro relativo al numero di elementi vicini da utilizzare settato a 15 e viene addestrato con l’utilizzo dei due dataframe Xtrain e yTrain restituiti dalla funzione getXtrainYTrain().

Il metodo è il seguente

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

KnnClassifier = KNeighborsClassifier(n\_neighbors=1)

Xtrain, yTrain = getXtrainYTrain ()

KnnClassifier.fit(Xtrain, yTrain)

return KnnClassifier

\end{minted}

Ho ritenuto opportuno valorizzare il campo con 1, in quanto è risultato il quantitativo necessario per non ottenere delle rilevazioni “ballerine” all’interno dell’interfaccia grafica da me realizzata e, allo stesso tempo, avere un valore di accuracy, calcolato come mostrato successivamente.

Ho poi eseguito il calcolo dell’accuratezza delle predizioni sul dataframe di test realizzato precedentemente attraverso il seguente codice:

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

from sklearn.metrics import accuracy\_score

yPred = KnnClassifier.predict(Xtest)

accuracy = accuracy\_score(yTest, yPred)

print("Accuracy: ", accuracy)

\end{minted}

ed ho ottenuto il seguente risultato:

Immagine che contiene testo, Carattere, Elementi grafici, grafica

Descrizione generata automaticamente

Il classificatore K-nearest neighbors (KNN) determina l'importanza delle caratteristiche basandosi sulla loro influenza sulla metrica di distanza utilizzata per calcolare la vicinanza tra le istanze. Più le istanze sono vicine, più sono simili. Pertanto, le caratteristiche più importanti sono quelle che contribuiscono maggiormente al calcolo della distanza.

Un modo per visualizzare l'importanza delle caratteristiche per un classificatore KNN è quello di utilizzare una mappa di calore della matrice di correlazione a coppie delle caratteristiche. Le caratteristiche con correlazioni elevate avranno un impatto minore sul calcolo della distanza, mentre le caratteristiche non correlate riporteranno l' effetto contrario.

Il seguente codice viene utilizzato per creare una heatmap della matrice di correlazione delle caratteristiche:

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

def visualizeHeatMapCorrelationMatrix(Xtrain):

corr = Xtrain.corr()

sns.heatmap(corr, cmap='coolwarm')

plt.show()

\end{minted}

Questo codice creerà una heatmap della matrice di correlazione delle caratteristiche nel set di addestramento. Il colore di ogni quadrato nella heatmap rappresenta la correlazione tra due caratteristiche.

Le caratteristiche altamente correlate saranno vicine tra loro nella heatmap, mentre le caratteristiche non correlate risulteranno distanti.

Immagine che contiene modello, schermata, Policromia, Rettangolo

Descrizione generata automaticamente

Dalla heatmap riportata è possibile dedurre che, oltre all’ovvia correlazione di ogni Action Units a se stessa, ad esempio, la AU1 e la AU2 sono strettamente correlate, o ancora, AU12 e AU6 sono ancora più strettamente correlate, indi, influenzeranno particolarmente le predizioni risultanti.

---------

Support vector machine (SVM) classifier

Un altro classificatore adoperato per effettuare delle predizioni sul dataset è basato sull’algoritmo Support vector machine.

La modalità di realizzazione è molto simile a quella per i precedenti classificatori, con modalità di utilizzo pedissequa al Random Forest classifier.

Viene sempre letto in memoria il dataset pre elaborato e resamplato, con successiva rimozione delle colonne non necessarie e viene suddiviso in set di addestramento e set di test.

L’oggetto che viene creato per effettuare le predizioni proviene sempre dalla libreria sklearn, in questo caso il modulo naive\_bayes e la sua relativa classe MultinomialNB.

Eseguendo dei test attraverso il metodo Eseguendo dei test attraverso il metodo \mintinline[bgcolor=bg]

{python}{score(Xtest, yTest)} offerto dalla classe MultinomialNB ho potuto calcolare l’accuracy del modello da me generato:



Naive Bayes classifier

Un ulteriore classificatore di cui ci si serve per effettuare delle predizioni sul dataset è basato sull’algoritmo Naive Bayes.

La modalità di realizzazione è molto simile a quella per i precedenti classificatori e la modalità di utilizzo è pari a quella del Random Forest classifier.

Viene sempre letto in memoria il dataset pre elaborato e resamplato, vengono rimosse le colonne non necessarie e viene suddiviso in set di addestramento e set di test.

L’oggetto che viene creato per effettuare le predizioni proviene sempre dalla libreria sklearn, in questo caso il modulo naive\_bayes e la sua relativa classe MultinomialNB.

Eseguendo dei test attraverso il metodo Eseguendo dei test attraverso il metodo \mintinline[bgcolor=bg]

{python}{score(Xtest, yTest)} offerto dalla classe MultinomialNB ho potuto calcolare l’accuracy del modello da me generato:



Il ~40% di accuracy è il valore più alto che sono riuscito ad ottenere modificando i diversi valori presi in input dal costruttore di MultinomialNB.

\chapter{Resampling del dataset per migliorare la precisione delle predizioni}

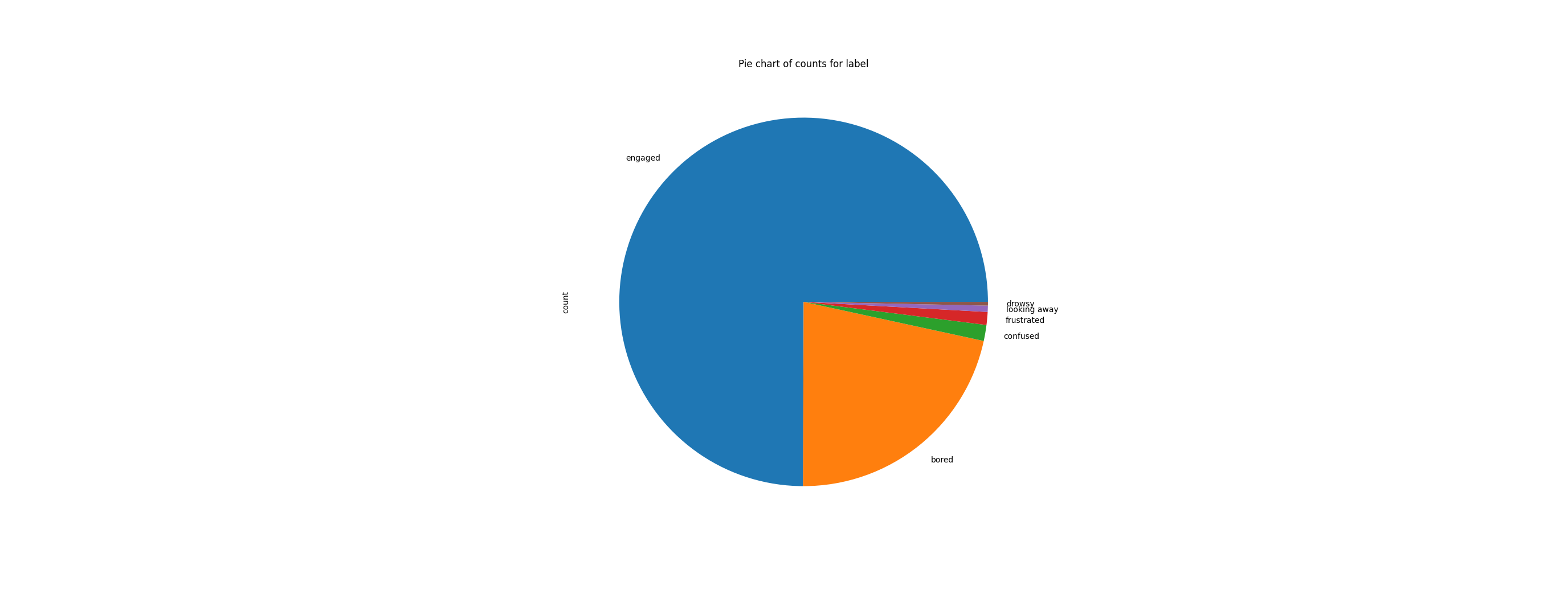
Il dataset risultato da me realizzato presenta una notevole complicazione: se decidessi di effettuare delle predizioni attraverso un classificatore random forest generato direttamente dal dataset as-his, un numero di campioni (samples) per ognuno dei valori della colonna labels risulterebbe sbilanciato.

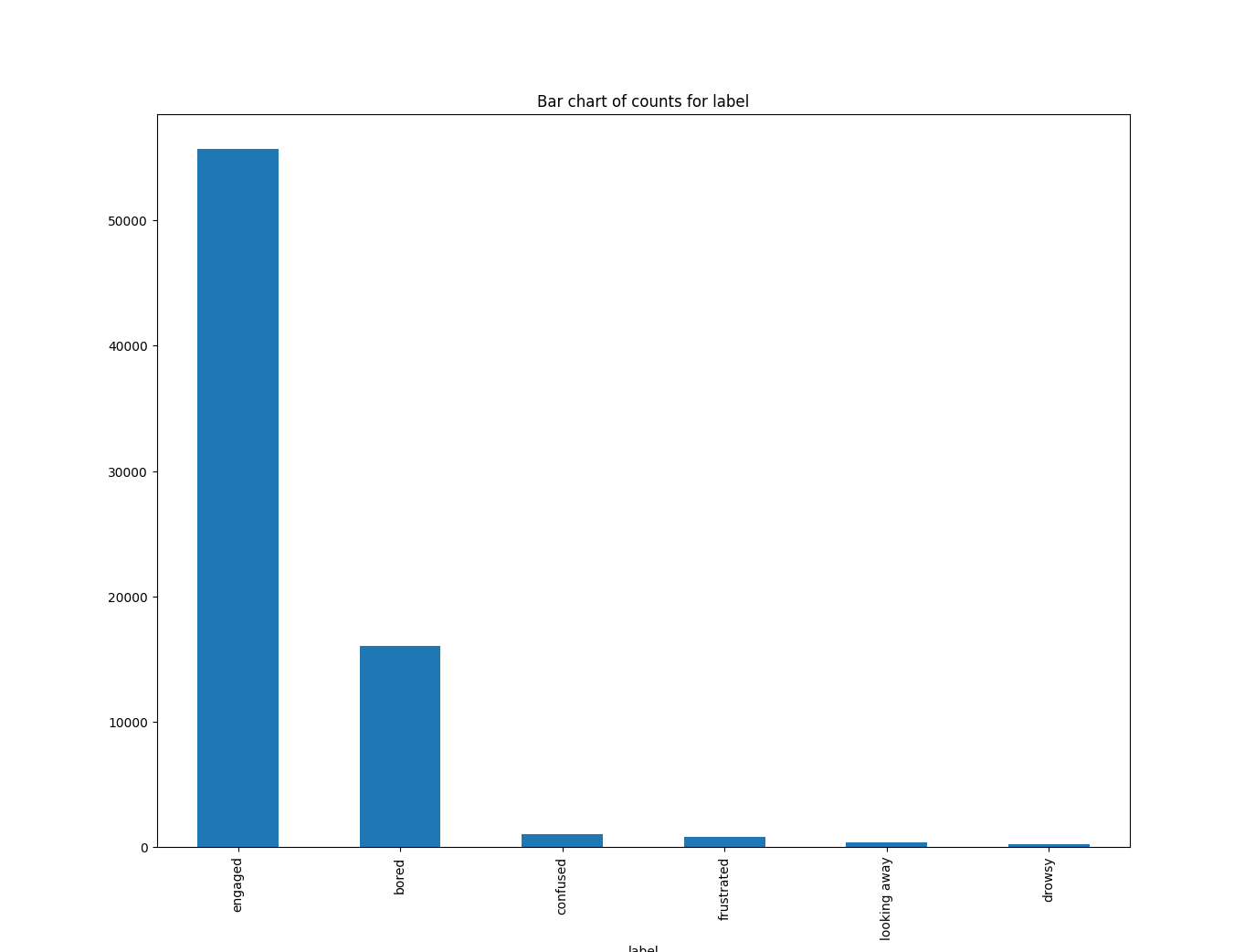
Per sbilanciato intendo il fatto che sono presenti molti valori per alcune delle classi, e troppi pochi, a confronto, per altri.

Immagine che contiene testo

Descrizione generata automaticamente

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





Il resampling, che prevede una modifica della 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. Esistono 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.

Ciò può essere effettuato in modo casuale, pur essendo 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.

Sussistono 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 e migliorare le prestazioni dei modelli di machine learning.

Il codice utilizzato per effettuare l’undersampling dei sample per ognuna delle labels è il seguente:

\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}

Il codice utilizzato per effettuare l’oversampling dei sample per ognuna delle labels è il seguente:

\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}

Questo invece è il codice che gestisce le chiamate per entrambi i metodi riportati sopra:

\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 al fine di 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 puntualizza 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

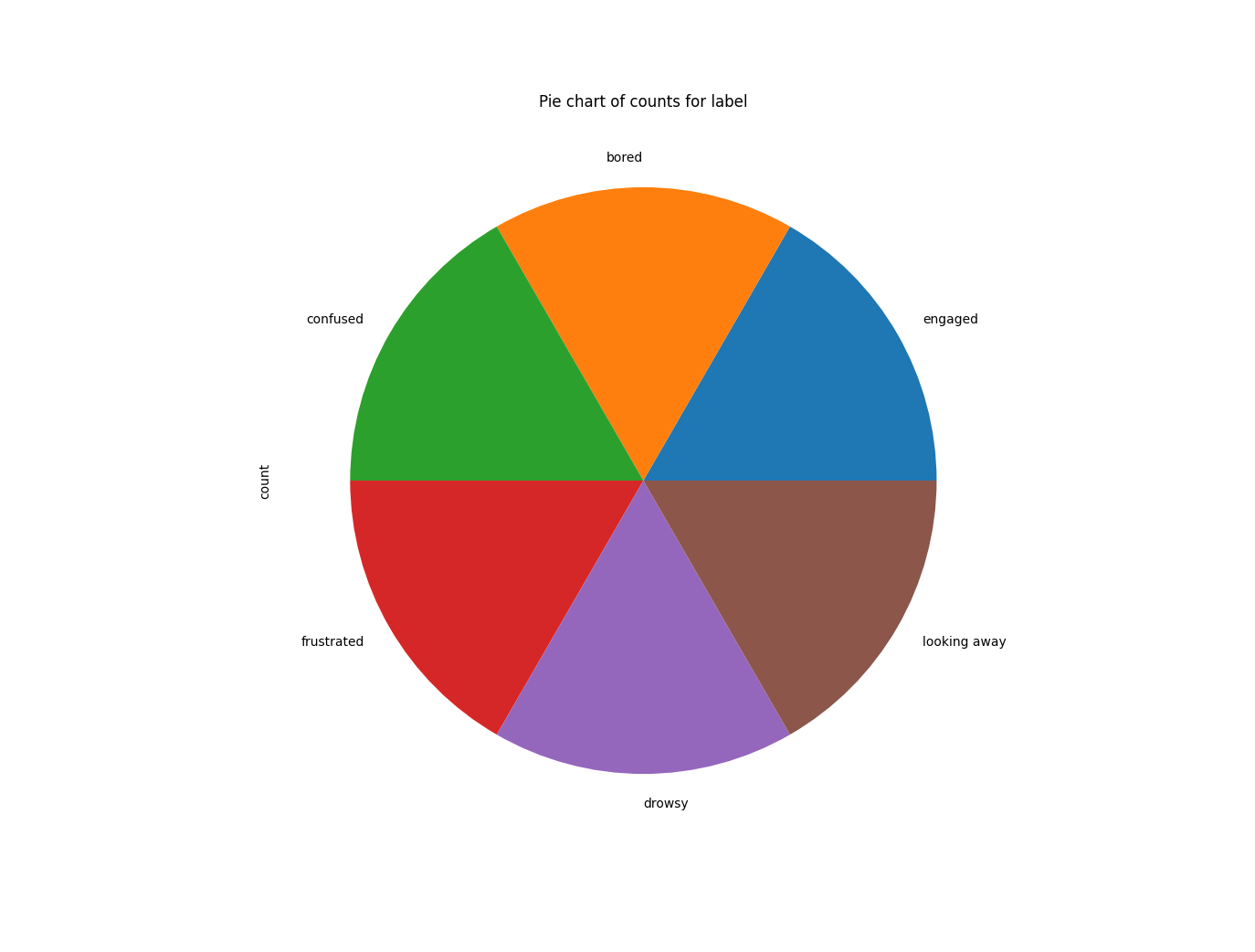
La funzione sample restituisce un nuovo dataframe contenente solo il sottoinsieme delle righe selezionate casualmente dal dataframe originale.

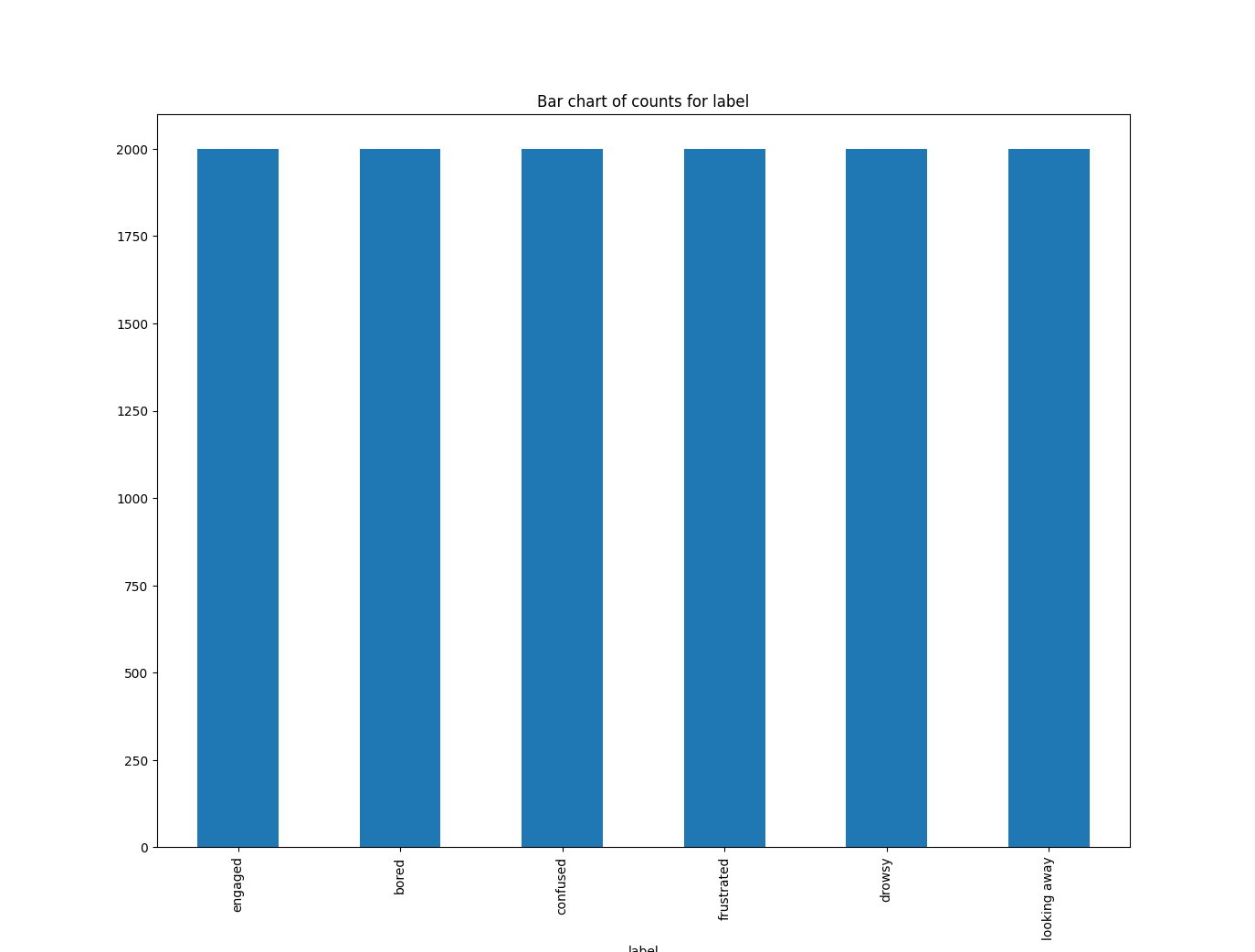
Fondamentalmente, la funzione sample è utilizzata per ridurre la dimensione del dataframe originale, selezionando solo un sottoinsieme casuale delle righe. Ciò può dimostrarsi utile per ridurre i tempi di calcolo durante l'addestramento dei modelli di machine learning, in particolare quando il dataset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

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 si sono rivelati 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 messo in pratica vari test, sono giunto alla conclusione di eseguire un resample per ottenere 2000 istanze per ogni label, in quanto questo sembra il numero di sample in grado di far conseguire risultati 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:





\chapter{Quale modello predittivo è il migliore?}

Ho collezionato diverse metriche per valutare quale dei modelli predittivi sviluppati fosse il migliore: Accuracy

La metrica dell'accuracy è una delle metriche più comuni utilizzate per valutare la performance dei modelli predittivi. Essa misura la percentuale di predizioni corrette rispetto al numero totale di predizioni effettuate. Ad esempio, se un modello ha effettuato 100 predizioni e 80 di queste sono corrette, l'accuracy sarebbe del 80%.

L'accuracy è una metrica particolarmente utile in caso, similmente al mio, di classi bilanciate. Ciò significa che il numero di esempi per ogni classe di output è simile. In questo contesto, può fornire una valutazione equilibrata della capacità del modello di predire entrambe le classi in modo corretto. Tuttavia, se le classi non dovessero essere bilanciate, tale metrica potrebbe risultare fuorviante, dal momento che il modello potrebbe essere molto accurato nella predizione della classe maggioritaria ma non essere in grado di predire la classe di minoranza in modo corretto.

Un'altra considerazione importante quando si utilizza l'accuracy è il tipo di errore che si vuole minimizzare. Ad esempio, se il costo di un falso positivo è molto alto, allora un modello che massimizza l'accuracy potrebbe non essere la scelta migliore.

Immagine che contiene testo, Carattere, linea, bianco

Descrizione generata automaticamente

Precision

La metrica di precisione è una delle metriche più comuni utilizzate per valutare le prestazioni di un modello predittivo. La precisione è definita come la proporzione di predizioni corrette rispetto al totale delle predizioni fatte dal modello. In altre parole, la precisione misura quanto spesso il modello predittivo fornisce una risposta corretta rispetto al totale delle risposte fornite. È una misura di quanto preciso sia il modello nel predire le classi corrette.

Per calcolare la precisione di un modello predittivo occorre confrontare le predizioni effettuate dal modello con le etichette di classe corrette. La precisione è calcolata dividendo il numero di predizioni corrette per il totale delle predizioni effettuate dal modello. Ad esempio, se un modello predittivo ha effettuato 100 predizioni e 80 di queste predizioni sono corrette, la precisione del modello è del 80%.

La metrica precision è definita come il rapporto tra il numero di veri positivi (TP) e il numero di tutti i casi predetti positivi (TP + FP). In altre parole, la precisione è la percentuale di volte in cui il modello ha predetto correttamente una classe positiva rispetto a tutte le volte in cui ha predetto quella classe.

Pur considerandosi utile, presenta alcune limitazioni. In particolare, la precisione non tiene conto dei falsi negativi, ovvero dei casi in cui il modello ha sbagliato a predire una classe positiva. Inoltre può essere influenzata dal numero di casi positivi e negativi presenti nel set di dati.

Immagine che contiene testo, Carattere, bianco, diagramma

Descrizione generata automaticamente

Recall

La metrica recall è una misura impiegata per la valutazione dela capacità di un modello predittivo di identificare correttamente i veri positivi tra tutti i positivi effettivi. In altre parole, la recall comunica quanti dei casi positivi presenti in un insieme sono stati correttamente individuati dal modello.

Per calcolare la recall si divide il numero di veri positivi per la somma dei veri positivi e dei falsi negativi. In questo modo otteniamo una percentuale che fa emergere quanti dei casi positivi sono stati individuati dal modello.

Una delle principali ragioni dell’importanza di tale metrica è che ci permette di valutare la capacità del modello di individuare correttamente i casi positivi, anche se questo significa avere un alto numero di falsi positivi.

Un altro aspetto da considerare quando si utilizza la recall come metrica è che può essere influenzata dal bilanciamento delle classi. Ad esempio, se abbiamo un insieme di dati in cui la maggior parte dei casi sono negativi, potrebbe essere difficile ottenere una buona recall anche con un modello altamente performante.

Infine, è importante notare che la recall non ci comunica nulla sulla capacità del modello di classificare correttamente i casi negativi. Pertanto, è sempre importante considerare anche altre metriche come la precisione e l'accuratezza complessiva del modello quando si valutano le prestazioni complessive.

Immagine che contiene Carattere, testo, bianco, diagramma

Descrizione generata automaticamente

Balanced Accuracy

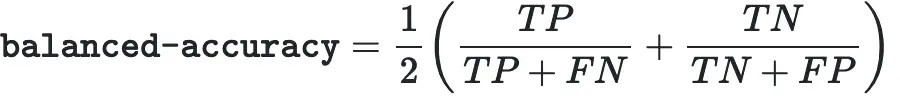
La metrica di Balanced Accuracy (o accuratezza bilanciata) è una misura di valutazione della performance di un modello predittivo che tiene conto della distribuzione bilanciata delle classi target all'interno del dataset di test. In altre parole, è una misura che considera l'accuratezza del modello in modo equilibrato per entrambe le classi target, evitando così di favorire una classe rispetto all'altra.

Per calcolare la metrica di Balanced Accuracy, si prende in considerazione la media aritmetica tra l'accuratezza della classe positiva (TPR) e quella della classe negativa (TNR), ovvero:

Balanced Accuracy = (TPR + TNR) / 2

Dove TPR (True Positive Rate) rappresenta la proporzione di veri positivi rispetto a tutti i casi positivi e TNR (True Negative Rate) rappresenta la proporzione di veri negativi rispetto a tutti i casi negativi.

In caso di dataset sbilanciati, in cui una classe target è più rappresentata dell'altra, la metrica di Balanced Accuracy risulta essere più informativa rispetto alla semplice accuratezza (Accuracy) in quanto fornisce una valutazione più equilibrata delle performance del modello per entrambe le classi target.



Tutte le metriche sono state calcolate a partire dai diversi modelli predittivi attraverso questo metodo:

\begin{minted}[bgcolor=bg]{python} def getModelScores(yTest, yPred):

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, balanced\_accuracy\_score acc = accuracy\_score(yTest, yPred)

prec = precision\_score (yTest, yPred, average="weighted") recall = recall\_score(yTest, yPred, average="weighted") balaAcc = balanced\_accuracy\_score(yTest, yPred)

return acc, prec, recall, balaAcc

\end{minted}

dove yTest è il dataframe estrapolato attraverso il metodo getXtrainYTrainXtestYTest precedentemente citato e yPred è la predizione effettuata sul dataframe Xtest, estratto sempre dallo stesso metodo, da parte del modello predittivo.

Ho opportunamente inserito tutte le metriche calcolate all’interno della tabella seguente per porre a confronto i modelli predittivi da me implementati:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | Balanced Accuracy |
| K-Nearest Neighbors | 78.54% | 76.96% | 78.54% | 78.04% |
| Random forest | 82.33% | 82.05% | 82.33% | 81.97% |
| Support vector machine | 55.70% | 55.47% | 55.70% | 55.17% |
| Naive bayes | 40.16% | 35.55% | 40.16% | 39.49% |

Come già evidenziato nel primo capitolo riguardante lo stato dell’arte, dalle analisi effettuate in \ cite{FaceExpreRecoImgSeqTwoFoldRandomForestClass} l’algoritmo di random forest risulta essere il più affidabile per effettuare le previsioni su dataset di Action Units col fine di un rilevamento delle emozioni FACS e, come si evince dalla tabella sopra riportata, lo stesso è vero per gli stati d’animo (o moods) che vengono trattati nel mio caso di studio.

\chapter{Realizzazione interfaccia grafica}

Una volta generati i modelli predittivi per effettuare nuove analisi, ho pensato che creare questo modello senza poi poterne usufruire, sia per fare dei test che come applicativo vero e proprio, sarebbe risultato fine a sé stesso.

Ho quindi scelto di creare una semplice interfaccia grafica attraverso la quale poter effettuare delle predizioni sullo stato d’animo della persona ripresa dalla webcam.

Una volta avviato il programma per l’interfaccia grafica, si presenta una schermata che dà la possibilità di scegliere fra i modelli predittivi creati:

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Una volta scelto l’algoritmo da utilizzare, questo viene, se precedentemente utilizzato, prelevato dalla memoria, altrimenti viene generato al momento e poi salvato su file binario attraverso i metodi disponibili nella libreria pickle (utilizzata sia per la scrittura che per la lettura di questi file contenti i modelli).

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

def getRandomForestClassifier():

filePathRandomForestClassifier = join(dirname(abspath(\_\_file\_\_)), "randomForestClassifier.pickle")

if isfile(filePathRandomForestClassifier):

with open(filePathRandomForestClassifier, "rb") as f:

randomForestClassifier = pickle.load(f)

else:

print("Creazione random forest classifier")

randomForestClassifier = RandomForestClassifier(n\_estimators=100, verbose=True, random\_state=69)

Xtrain, yTrain, Xtest, yTest = getXtrainYTrain()

randomForestClassifier.fit(Xtrain, yTrain)

#visualizeFeaturesImportances(randomForestClassifier, datasetWithoutLabelCol)

#randomForestClassifierVisualize(randomForestClassifier)

with open(filePathRandomForestClassifier, "wb") as f:

pickle.dump(randomForestClassifier, f)

relativeTestResult = randomForestClassifier.score(Xtest, yTest)

print("Relative test result:", relativeTestResult)

return randomForestClassifier

\end{minted}

La schermata che si presenta successivamente è questa:

Immagine che contiene Viso umano, testo, schermata, uomo

Descrizione generata automaticamente

Sulla sinistra è possibile notare la webcam dalla quale vengono estratti i frame per effettuare le analisi e che riprende, ovviamente, il soggetto di fronte alla fotocamera.

Sulla destra è invece possibile osservare il testo, suddiviso in:

La prima riga, dove è presente il mood rilevato, ossia la label con la percentuale di predizione più alta secondo il modello predittivo precedentemente scelto sull’immagine che è stata catturata in quel momento.

Subito sotto sono presenti tutte le label e le relative percentuali di predizione calcolate.

È poi riportato il mood più frequente nell’ultimo minuto per decidere quale label fra quelle estratte vada riportata qui; ho poi immagazzinato in una struttura dati dizionario (dict di python) ognuna delle label con la percentuale di predizione più alta raccolte nell’ultimo minuto, e le ho utilizzate come chiave; come valori ho invece immagazzinato il timestamp nel quale è stata effettuata la predizione.

Ogni minuto questo dizionario viene aggiornato rimuovendo le coppie chiave di valore più “vecchie” di un minuto.

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

def addToBestClassesLastMinute (bestClass):

bestClassesLastMinute[time.time()] = bestClass

def removeOldKeys(bestClassesLastMinute):

currentTime = time.time()

oneMinuteAgo = currentTime - 60

return {k:v for k,v in bestClassesLastMinute.items() if k > oneMinuteAgo}

\end{minted}

Infine, viene mostrato quanto tempo è passato fra una predizione e l’altra, così da fornire un’idea delle prestazioni del programma.

Ovviamente le prestazioni dell’interfaccia variano in base alla macchina sulla quale viene compilata:

il personal computer che ho utilizzato per eseguire il programma, dotato di queste specifiche:

* Ryzen 7 5800H
* 16GB ram DDR4
* GeForce RTX 3060 6GB (mobile)
* SSD m.2

mi ha permesso di ottenere intorno alle 100 predizioni al minuto e, di fatti, il video mostrato nella schermata risulta andare a scatti.

Eseguendo invece la stessa interfaccia su una macchina più prestante, fornitami dal collega Francesco Saverio Cassano, con le seguenti caratteristiche tecniche:

* i7-13700kf (no overclock)
* GeForce RTX 3080 10GB
* 32GB ram DDR5
* SSD m.2

Il numero di predizioni per minuto risulta essere [TODO inserisci numero di predizioni per minuto su pc di saverio] e il video risulta, ovviamente, essere più fluido.

N.B.: la webcam utilizzata per l’esecuzione è sempre la stessa.

È importante mettere in luce che per effettuare le predizioni l’immagine mostrata sullo schermo viene salvata sul disco, e che le prestazioni dipendono anche dal tipo di disco sul quale viene eseguita l’analisi; ho difatti riscontrato un decremento notevole delle performance nel momento in cui ho provato ad eseguire il programma su un hard disk classico rispetto ad un SSD, tipologia di disco utilizzata in entrambe le macchine sopracitate.

Un’altra differenza importante è data dal fatto che entrambe le macchine presentate sono dotate di una scheda video di casa Nvdia e possono quindi sfruttare la tecnologia CUDA per effettuare l’estrazione delle Action Units dall’immagine prelevata, il che migliora esaurientemente le prestazioni.

Effettuando delle predizioni impostando il parametro del costruttore della classe Detector offerta da py-feat ho rilevato un decremento di performance, che porta il delta tempo fra una predizione e l’altra da poco meno di un secondo (~0.6/0.7s) a ~1.6/1.7 secondi.