Game Ratings Predictor

Scheda riassuntiva

1. **Titolo**:
   Video Game Ratings Predictor
2. **Autori**:
   Roberto Falconi (50%), Federico Guidi (50%)
3. **Classe di problema affrontato**:
   multiclass classification (one versus rest), supervised learning
4. **Dataset**:
   https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings
5. **Descrizione features X:**
   a. **Platform (stringa discretizzata):**
      piattaforma su cui è stato sviluppato il videogioco
   b. **Genre (stringa discretizzata):**
      genere di appartenenza del videogioco
   c. **Year_of_Realease (intero):**
      anno in cui è stato rilasciato il gioco
   d. **Publisher (stringa discretizzata):**
      azienda responsabile della produzione e commercializzazione dei loro prodotti, compresi aspetti pubblicitari, ricerche di mercato e finanziamento dello sviluppo del videogioco
   e. **Developer (stringa discretizzata):**
      azienda che si occupa dello sviluppo del videogioco
   f. **NA_Sales (intero):**
      vendite del videogioco in Nord America
   g. **EU_Sales (intero):**
      vendite del videogioco nell'Unione Europea
   h. **JP_Sales (intero):**
      vendite del videogioco in Giappone
   i. **Other_Sales (intero):**
      vendite del videogioco nel resto del mondo
   j. **Global_Sales (intero):**
      vendite globali del videogioco
   k. **Critic_Score (intero 1-100):**
      voto della critica
   l. **Critic_Count (intero):**
      numero di recensioni ricevute dalla critica per ottenere il Critic_Score
   m. **User_Score (intero 1-100):**
      voto degli utenti

n. **User_Count (intero):**
numero di recensioni ricevute dagli utenti per ottenere lo User_Score

6. **Descrizione y**:
è rappresentata dal Rating (una stringa di valore E|E10+|T|M|AO), che identifica la fascia d'età di destinazione del videogioco; è stata quindi applicata la tecnica del One-hot encoding per la discretizzazione della stessa

7. **Descrizione problema**:
a partire dal training set l'algoritmo elabora i dati e si allena per fornire i risultati richiesti, ovvero la y (cioè il Rating) prevista per quegli elementi (video games) presenti nel validation set, cercando di prevedere quale sarà esattamente la classe di Rating assegnato (una tra quelle elencate al punto 6)

8. **Tipo di modello applicato**:
Logistic Regression, Random Forest, k-NN con applicazione della tecnica Coarse-to-fine per identificare i parametri

9. **Tipo di stima**:
accuracy score, misclassification rate, confidenza, probabilità normalizzata che un elemento appartenga realmente ad una classe piuttosto che ad un'altra

10. **Metodo di validazione utilizzato:**
gli elementi presenti nel dataset vengono prima scansionati per assicurarsi che non vi siano elementi che presentino valori non validi (come tbd) o nulli (NaN), dopodiché vengono mischiati tra di loro in maniera casuale e vengono divisi in un training set (80%) ed un validation set (20%). Vengono calcolati sia un accuracy score (ed un misclassification rate), sia una cross-validation.

11. **Descrizione sintetica dei risultati ottenuti**:
Per restituire il Rating corretto sono stati sfruttati i tre modelli citati al punto 8 e sono stati comparati tra di loro per verificarne il migliore: Logistic Regression e Random Forest restituiscono valori di accuratezza molto simili tra di loro, entrambi pari all'85% circa, mentre il k-NN raggiunge un'accuratezza dell'80% circa ed un misclassification rate più elevato

12. **Linguaggio**:
Python

13. **Libreria**:
Pandas, NumPy, SciPy, matplotlib e Scikit-Learn

1. Introduzione

In questo corso di Metodi Quantitativi per l'Informatica, tenuto dal Professor Luigi Freda, siamo stati introdotti al Machine Learning. In seguito a studi approfonditi dei metodi insegnati a lezione, abbiamo sviluppato un progetto open-source, utilizzando il linguaggio di programmazione Python e le sue librerie Pandas, NumPy, SciPy, matplotlib e Scikit-Learn.

L'obiettivo del progetto è innanzitutto quello di mettere in pratica gli insegnamenti ricevuti sulla materia e di realizzare un programma che sia in grado di effettuare correttamente le elaborazioni di ricerca quantitativa richieste.[1] Per fare questo, ci siamo posti un obiettivo: riuscire a prevedere la fascia d'età di appartenenza di un videogioco. L'Entertainment Software Rating Board (ESRB) è un ente che si occupa della classificazione dei videogiochi pubblicati nel Nord America, istituito nel 1994 dalla Entertainment Software Association. Il nostro algoritmo dovrà essere in grado di prevedere tali classificazioni, seguendo la tecnica di apprendimento automatico supervised learning, che mira a istruire un sistema informatico in modo da consentirgli di risolvere dei compiti in maniera autonoma sulla base di una serie di esempi ideali, costituiti da coppie di input e di output desiderati, che gli vengono inizialmente forniti.[2]

Abbiamo trovato quindi un Dataset (insieme di dati strutturati in forma relazionale, cioè corrisponde al contenuto di una singola tabella di database, oppure ad una singola matrice di dati statistici, in cui ogni colonna della tabella rappresenta una particolare variabile, e ogni riga corrisponde ad un determinato membro del Dataset in questione) di 16720 elementi, scaricabile gratuitamente all'indirizzo https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings, dove ogni elemento rappresenta un videogioco di cui si conosce: titolo, piattaforma su cui è stato sviluppato, anno di rilascio, genere, produttore, sviluppatore, vendite in Nord America, in Europa, in Giappone e nel resto del mondo, voto della critica, numero di recensioni ricevute da quest'ultima, voto degli utenti e numero di recensioni ricevute da quest'ultimi e, infine, la classificazione assegnata dall'ESRB.

La classificazione ESRB, prevede 4 classi: Everyone (E), Everyone 10+ (E10+), Teen (T), Mature (M) e Adults Only (AO); tuttavia, di tutti i 16720 videogiochi presenti nel Dataset solo uno rientrava nella fascia (AO) e pertanto è stato escluso dalla trattazione, poiché risultava inutile ed impossibile riuscire a prevedere una futura classificazione dato un campione di un solo elemento.[3]

In questa relazione andremo ad esaminare il contenuto di `algoritmo.py` riga per riga, secondo un approccio del tipo TOP-DOWN. `algoritmo.py` è composto di tre funzioni: inizieremo con la funzione `main` e concluderemo con le funzioni ausiliarie `animate` e `plot_learning_curve`. Cercheremo dunque di essere i più chiari possibili su ogni aspetto dello stesso, cercando di spiegare non solo il semplice codice Python, ma anche la teoria e i concetti che vi si celano dietro.

2. Installazione

Per permettere un'esecuzione del programma che fosse la più accessibile ed immediata possibile, abbiamo pensato ad un pacchetto eseguibile per poter installare innanzitutto le già citate librerie Python fondamentali per l'esecuzione del progetto, e successivamente per poterlo eseguire o installare in maniera permanente.
Andiamo quindi ad esplicare i passaggi necessari, separandolii in base al proprio sistema operativo, in particolare la procedura risulta diversa tra i sistemi Linux-based e Windows.

Ubuntu, Debian e macOS

Prima di tutto risulta necessario avere Python e pip installati sul proprio computer, da terminale usiamo il comando

```
>> sudo apt-get install python-dev
```

Adesso assicuriamoci di avere pip alla sua ultima versione

```
>> pip install --upgrade pip
```

Passiamo adesso al scaricare la repository, quindi sulla directory dove si intende scaricarla

```
>> git clone https://robertofalconi95@gitlab.com/robertofalconi95/MQPI.git
```

Adesso installiamo tutti i package Python richiesti dall'applicazione

```
>> sudo pip install -r requirements.txt
```

Finalmente, possiamo far partire il programma direttamente con il comando

```
>> python algoritmo-runner.py
```

oppure possiamo decidere di installarlo permanentemente sul nostro dispositivo tramite

```
>> sudo python setup.py install --record files.txt
```

disinstallarlo con

```
>> uninstall it with cat files.txt | xargs rm -rf
```

e infine farlo partire ovunque nel termine tramite il semplice comando

```
>> algoritmo
```

Windows

Andiamo su https://www.python.org/downloads/windows/ per scaricare il .exe di Python ed installarlo sul nostro computer.
Andiamo quindi su https://bootstrap.pypa.io/get-pip.py ed installiamolo tramite Prompt dei comandi

```
>> python get-pip.py
```

Nella directory dove abbiamo installato Python entriamo nella cartella Scripts e creiamo un nuovo file (tasto destro) chiamato local.bat con la sola parola cmd dentro di esso.
Adesso con un doppio click su local.bat si aprirà un terminale sul quale utilizzare i comandi

```
>> pip install numpy
>> pip install scipy
>> pip install pandas
>> pip install -U scikit-learn
```

Adesso possiamo avviare il programma aprendo algoritmo-runner.py tramite Python Launcher.

3. Struttura del codice

```
# -*- coding: utf-8 -*-
```

La codifica UTF-8 del codice in Python acquisisce particolare rilevanza, in quanto quando si trova necessario lavorare con file di vario formato (siano essi .txt, .csv, ecc.) il Python Interpreter, 2.x o 3.x, richiede esplicitamente la conoscenza del formato con cui andrà a lavorare.
Nel nostro caso senza esplicitare il coding in UTF-8, il Python Interpreter restituiva l'errore:

```
Non-ASCII character '\xe2' in file algoritmo.py on line 179, but no encoding declared;
see http://www.python.org/peps/pep-0263.html for details
```

```
import itertools
import threading
import time
import sys

import pandas as pd
import numpy as np
import scipy
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import learning_curve
import matplotlib.pyplot as plt
```

Tutti gli import sono necessari per utilizzare le relative librerie di Python che vengono sfruttate all'interno del programma.
Nello specifico, abbiamo diviso gli import da uno spazio: nella prima parte abbiamo l'importazione di librerie standard e distribuiti unitamente a Python (come avviene, ad esempio, con `stdio.h` per il linguaggio di programmazione C) e che ci serviranno esclusivamente all'interno della funzione animate descritta al capitolo 9; nella seconda parte ci sono tutte librerie open-source e BSD-licensed necessarie per il funzionamento del nostro progetto.
Tra queste troviamo: Pandas (importato come pd), NumPy (non importato esplicitamente nel codice ma necessario per il funzionamento di `sklearn`), SciPy, matplotlib e Scikit-Learn (importato nel nostro codice come sklearn):

- Pandas fornisce alte performance e un utilizzo semplice ed immediato per la gestione di strutture dati e strumenti di analisi per le stesse da sfruttare proprio in programmi basati in Python; nel nostro caso viene sfruttata per il caricamento dei 16720 elementi presenti all'interno del file comma-separated values (CSV), che è proprio un formato baso su file di testo utilizzato per l'importazione e l'esportazione (ad esempio da fogli elettronici o database) di una tabella di dati.[4]
- NumPy è un pacchetto fondamentale per la computazione scientifica con Python. Tra le tante funzionalità contiene: una potente ed efficiente gestione di array multi-dimensionali di oggetti di tipo generico, che permettono a NumPy una integrazione stabile e veloce con una grande varietà di database.[5]

- SciPy è un insieme di strumenti scientifici necessari a Python per l'elaborazione di dati. Attualmente supporta funzioni speciali, integrazione, soluzione di equazioni differenziali ordinarie, ottimizzazione del gradiente (individuazione del massimo e del minimo di una funzione a più variabili), strumenti per la programmazione parallela, e molto altro. Mentre NumPy è più indicato per operazioni di base su Dataset, come indexing, sorting, reshaping, basic elementwise functions ecc., SciPy contiene tutto il codice numerico e ha più funzioni per operare con moduli di algebra lineare così come altri algoritmi per il calcolo numerico. I due pacchetti sono completamente compatibili tra loro e si completano a vicenda: è bene quindi installarli entrambi.[6]
- Matplotlib è una libreria che consente di tracciare grafici 2D con Python, che producono figure di qualità in vari formati digitali e ambienti interattivi multi piattaforma: permette quindi di generare grafici, istogrammi, eccetera.
- Per un semplice tracciamento, il modulo pyplot fornisce una interfaccia simile a MATLAB, che per fornire strumenti agli utenti, permette di controllare anche lo stile delle linee, le proprietà dei font utilizzati, le proprietà degli assi e così via.
- Matplotlib può essere utilizzato negli script di Python, nella shell IPython, nei server web application e nella interfacce grafiche destinate agli utenti.[7]
- Scikit-Learn è uno strumento semplice da usare ma efficiente per data mining e data analysis. È accessibile a chiunque e riusabile in vari contesti ed è costruito proprio sui precedentemente citati NumPy, SciPy e matplotlib.
- È il cuore di un algoritmo Python basato sul Machine Learning, in quanto è indispensabile per la realizzazione di un programma che adoperi i processi di Classification, Regression, Clustering, Dimensionality reduction, Model selection, Preprocessing.[8]
- Nel nostro caso, è stato utilizzato proprio per importare all'interno del programma le funzioni che ci permettessero di utilizzare Logistic Regression, Random Forest, k-NN e ulteriori funzioni sia per valutare i risultati ottenuti da quest'ultime (accuracy_score), sia per graficare tali risultati (learning_curve), sia per suddividere il dataset negli insiemi Training Set e Validation Set da passare come parametri ai tre metodi utilizzati nel progetto. Tutte queste funzioni, ad ogni modo, verranno approfondite più avanti nella trattazione quando verranno effettivamente utilizzate all'interno del codice Python.

```
RUN_ANALYSIS_LOGISTIC_PENALTY = False
RUN_ANALYSIS_RANDOMFOREST_NUMBERESTIMATOR = False
RUN_ANALYSIS_RANDOMFOREST_ENTROPY = False
RUN_ANALYSIS_RANDOMFOREST_ENTROPYANDNE40 = False
RUN_ANALYSIS_KNN_NUMBERNEIGHBORS = False
RUN_ANALYSIS_KNN_LEAFSIZE = False
RUN_ANALYSIS_KNN_P = False
```

Sono delle variabili globali che verranno descritte nei paragrafi successivi.
Servono per far stampare tutti i test che dimostreranno gli esperimenti da noi eseguiti e per dimostrare di conseguenza la veridicità delle nostre tesi.

4. Funzione main e preparazione del Dataset

```
def main():
    print('Avvio del programma')

    # Caricamento del dataset
    df = pd.read_csv('../MQPI/docs/Video_Games_Sales_as_at_22_Dec_2016.csv',
    encoding="utf-8")
```

Il def main() è necessario per racchiudere l'algoritmo in una funzione che verrà poi chiamata dall'interterprete per far eseguire tutto il progetto.

La print serve soltanto come flag, per avvisare l'utente della reale esecuzione (o almeno l'avvio della stessa) del programma.

df (dataframe, ovvero il nome con cui Pandas indica i Dataset) è una variabile che contiene tutti i dati da analizzare.
Quest'ultimi vengono caricati proprio tramite la funzione read_csv contenuta in Pandas, che viene chiamato nel nostro caso pd (pandas dataframe).[9]
All'interno delle parentesi della funzione read_csv, come argomenti della funzione stessa, passiamo l'indirizzo locale dove è contenuto il già citato file .csv, chiamato
`Video_Games_Sales_as_at_22_Dec_2016.csv`,
contenuto all'interno della folder docs, a sua volta nella directory principale MQPI (che è la repository scaricata da Git e che può essere collocata a piacere dall'utente nel proprio computer).
Ancora una volta, è importante specificare la codifica con cui il Python Interpreter dovrà leggere il file .csv attraverso la libreria Pandas.

```
    # Eliminazione dei Platform obsoleti

    df = df[df.Platform != "2600"]
    df = df[df.Platform != "3DO"]
    df = df[df.Platform != "DC"]
    df = df[df.Platform != "GEN"]
    df = df[df.Platform != "GG"]
    df = df[df.Platform != "NG"]
    df = df[df.Platform != "PCFX"]
    df = df[df.Platform != "SAT"]
    df = df[df.Platform != "SCD"]
    df = df[df.Platform != "TG16"]
    df = df[df.Platform != "WS"]
```

In questa sezione abbiamo eliminato tutti i videogiochi appartenente a console (platform) obsolete e datate, in quanto non sono più supportate dalle aziende produttrici (per dismissione o fallimento) e, di conseguenza, non rappresentavano né un campione sufficientemente ampio da cui prendere elementi per la futura elaborazione, né simboleggiavano un punto di interesse oggigiorno.

```
    # Eliminazione elementi non completi
```

```
df = df[df.User_Score != "tbd"]
df = df.dropna().reset_index(drop=True)
```

Adesso, è stato necessario rimuovere tutti gli elementi con informazioni mancanti, in quanto vengono caricati all'interno del programma e in seguito letti come valore NaN.
Il NaN è un simbolo di avvertimento indicante che il risultato di un'operazione è stata eseguita su operando non validi. Il suo nome è l'acronimo di Not a Number.
Per questo motivo Python non è in grado di gestirlo e di conseguenza porta con sé errori di valutazione, andando a inficiare sui risultati finali della classificazione.
Lasciando i NaN (ovvero senza aggiungere la seconda riga della porzione di codice indicata nella sezione soprastante), viene restituito l'errore:

```
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

Situazione analoga si presenta nel momento in cui vogliamo andare a eseguire il programma senza rimuovere gli elementi (videogiochi) che presentano un valore tbd.
tbd è presente all'interno del dataset, in corrispondenza di tutti quegli elementi che non hanno ancora ricevuto una valutazione dagli utenti, in quanto giochi meno diffusi, è infatti l'acronimo di "to be defined".
Volendo lasciare tali valori nel dataset, eliminando la prima riga della sezione di codice sopra riportata, verrà restituito l'errore:

```
ValueError: could not convert string to float: tbd
```

```
# Discretizzazione feature Platform

df["Platform_XB"] = 0
df["Platform_X360"] = 0
df["Platform_XOne"] = 0
df["Platform_PC"] = 0
df["Platform_PS"] = 0
df["Platform_PS2"] = 0
df["Platform_PS3"] = 0
df["Platform_PS4"] = 0
df["Platform_PSP"] = 0
df["Platform_PSV"] = 0
df["Platform_GB"] = 0
df["Platform_GBA"] = 0
df["Platform_DS"] = 0
df["Platform_3DS"] = 0
df["Platform_NES"] = 0
df["Platform_SNES"] = 0
df["Platform_N64"] = 0
df["Platform_GC"] = 0
df["Platform_Wii"] = 0
df["Platform_WiiU"] = 0

for elem in df.index.get_values():
    if df.get_value(elem, "Platform") == "XB": df.set_value(elem,
"Platform_XB", 1)
    if df.get_value(elem, "Platform") == "X360": df.set_value(elem,
"Platform_X360", 1)
    if df.get_value(elem, "Platform") == "XOne": df.set_value(elem,
"Platform_XOne", 1)
    if df.get_value(elem, "Platform") == "PS": df.set_value(elem,
```

```
"Platform_PS", 1)
    if df.get_value(elem, "Platform") == "PS2": df.set_value(elem,
"Platform_PS2", 1)
    if df.get_value(elem, "Platform") == "PS3": df.set_value(elem,
"Platform_PS3", 1)
    if df.get_value(elem, "Platform") == "PS4": df.set_value(elem,
"Platform_PS4", 1)
    if df.get_value(elem, "Platform") == "PSP": df.set_value(elem,
"Platform_PSP", 1)
    if df.get_value(elem, "Platform") == "PSV": df.set_value(elem,
"Platform_PSV", 1)
    if df.get_value(elem, "Platform") == "GB": df.set_value(elem,
"Platform_GB", 1)
    if df.get_value(elem, "Platform") == "GBA": df.set_value(elem,
"Platform_GBA", 1)
    if df.get_value(elem, "Platform") == "DS": df.set_value(elem,
"Platform_DS", 1)
    if df.get_value(elem, "Platform") == "3DS": df.set_value(elem,
"Platform_3DS", 1)
    if df.get_value(elem, "Platform") == "NES": df.set_value(elem,
"Platform_NES", 1)
    if df.get_value(elem, "Platform") == "SNES": df.set_value(elem,
"Platform_SNES", 1)
    if df.get_value(elem, "Platform") == "N64": df.set_value(elem,
"Platform_N64", 1)
    if df.get_value(elem, "Platform") == "GC": df.set_value(elem,
"Platform_GC", 1)
    if df.get_value(elem, "Platform") == "Wii": df.set_value(elem,
"Platform_Wii", 1)
    if df.get_value(elem, "Platform") == "WiiU": df.set_value(elem,
"Platform_WiiU", 1)

# Discretizzazione feature Genre

df["Genre_Action"] = 0
df["Genre_Adventure"] = 0
df["Genre_Fighting"] = 0
df["Genre_Misc"] = 0
df["Genre_Platform"] = 0
df["Genre_Puzzle"] = 0
df["Genre_Shooter"] = 0
df["Genre_Sports"] = 0
df["Genre_Simulation"] = 0
df["Genre_Strategy"] = 0
df["Genre_Racing"] = 0
df["Genre_Role-Playing"] = 0

for elem in df.index.get_values():
    if df.get_value(elem, "Genre") == "Action": df.set_value(elem,
"Genre_Action", 1)
    if df.get_value(elem, "Genre") == "Adventure": df.set_value(elem,
"Genre_Adventure", 1)
    if df.get_value(elem, "Genre") == "Fighting": df.set_value(elem,
"Genre_Fighting", 1)
    if df.get_value(elem, "Genre") == "Misc": df.set_value(elem, "Genre_Misc",
1)
    if df.get_value(elem, "Genre") == "Platform": df.set_value(elem,
"Genre_Platform", 1)
    if df.get_value(elem, "Genre") == "Puzzle": df.set_value(elem,
"Genre_Puzzle", 1)
    if df.get_value(elem, "Genre") == "Shooter": df.set_value(elem,
"Genre_Shooter", 1)
    if df.get_value(elem, "Genre") == "Sports": df.set_value(elem,
"Genre_Sports", 1)
```

```
    if df.get_value(elem, "Genre") == "Simulation": df.set_value(elem,
"Genre_Simulation", 1)
    if df.get_value(elem, "Genre") == "Strategy": df.set_value(elem,
"Genre_Strategy", 1)
    if df.get_value(elem, "Genre") == "Racing": df.set_value(elem,
"Genre_Racing", 1)
    if df.get_value(elem, "Genre") == "Role-Playing": df.set_value(elem,
"Genre_Role-Playing", 1)

# Discretizzazione feature Rating

df["Rating_Everyone"] = 0
df["Rating_Everyone10"] = 0
df["Rating_Teen"] = 0
df["Rating_Mature"] = 0
df["Rating_Adult"] = 0

for elem in df.index.get_values():
    if df.get_value(elem, "Rating") == "E": df.set_value(elem,
"Rating_Everyone", 1)
    if df.get_value(elem, "Rating") == "E10+": df.set_value(elem,
"Rating_Everyone10", 1)
    if df.get_value(elem, "Rating") == "T": df.set_value(elem, "Rating_Teen",
1)
    if df.get_value(elem, "Rating") == "M": df.set_value(elem,
"Rating_Mature", 1)
    if df.get_value(elem, "Rating") == "AO": df.set_value(elem,
"Rating_Adult", 1)
```

Nelle porzioni di codice riportate nelle pagine precedenti, abbiamo effettuato una discretizzazione delle features, nello specifico, è stata applicata la tecnica del One-hot encoding.[10]
Le features sono le colonne del dataset, ciascun elemento è pertanto individuato da un vettore di features.[11]
Esse sono quindi delle caratteristiche cruciali che descrivono un elemento, sono dunque fondamentali per il funzionamento dell'intero algoritmo al fine di effettuare una pattern recognition, una regression o, come nel nostro caso, una classification.

Perché discretizzare?
Chiaramente, come abbiamo riportato precedentemente nell'esempio del tbd, Python non è in grado di convertire autonomamente string in float, non è quindi capace di assimilare le informazioni che derivano da una singola stringa di testo.
Tuttavia alcune delle nostre feature, non sono numeriche, ma sono proprio delle stringhe. È impossibile pensare di abbandonare un numero così elevato di informazioni perché sono informazioni non solo rilevanti ma fondamentali.
Per poter quindi eludere questa problematica e sfruttare questi dati necessari ai nostri scopi, dobbiamo mettere in opera un exploit: trasformare manualmente stringhe di testo in numeri che siano però comprensibili dall'interprete e quindi assimilabili per poter ricavarne a sua volta risultati finale dopo una elaborazione.
Dobbiamo quindi trasformare una stringa di testo (caratteristica del mondo del continuo e del reale) in una variabile binaria (sì / no, vero / falso, 1 / 0).

Per discretizzare ciascuna feature (nel codice riportato precedentemente abbiamo discretizzato solo le feature Platform, Genre e Rating), abbiamo creato tante feature quante sono le classi di appartenenza delle stesse:

ad esempio se la feature Rating presentava 5 possibili valori (E, E10+, T, M, AO), abbiamo creato altrettante feature (Rating_E, Rating_E10+, Rating_T, Rating_M, Rating_AO).
Questa volta, le nuove features non sono più stringhe, ma appartengono ad un mondo discretizzato e pertanto comprensibile al Python Interpreter: sono tutte variabili binarie con valore 1 se precedentemente appartenevano alla classe della nuova feature, 0 altrimenti.[12]

Per fare tutto ciò, tramite codice Python, abbiamo quindi aggiunto una nuova feature per classe con il codice:

```
df["Platform_XB"] = 0
```

e successivamente, all'interno di un ciclo for-each, che passa in rassegna ogni elemento presente nel dataset, tramite il codice:

```
for elem in df.index.get_values():
```

abbiamo verificato se l'elemento corrente fosse appartenente alla sua relativa classe, in tal caso viene assegnato valore 1, altrimenti viene lasciato a 0. Questo è stato effettuato con il codice:

```
if df.get_value(elem, "Platform") == "XB": df.set_value(elem, "Platform_XB", 1)
```

```
# Discretizzazione feature Publisher

publisher_list = []

for elem in df.Publisher:
    if elem not in publisher_list:
        publisher_list.append(elem)

for elem in publisher_list:
    df[elem] = 0

for elem in df.index.get_values():
    df.set_value(elem, df.get_value(elem, "Publisher"), 1)



# Discretizzazione feature Developer

developer_list = []

for elem in df.Developer:
    if elem not in developer_list:
        developer_list.append(elem)

for elem in developer_list:
    df[elem] = 0

for elem in df.index.get_values():
    df.set_value(elem, df.get_value(elem, "Developer"), 1)
```

Dopo aver discrettizato le precedent feature manualmente, ci siamo resi conto dell'impossibilità di continuare nello stesso modo anche per le features Publisher e Developer, in quanto non presentavano un range di valori molto limitato come accadeva per le features precedenti, ma essendo impossibile elencarli tutti siamo ricorsi a un processo generico tramite cicli for-each

Creiamo innanzitutto una lista di appoggio vuota, tramite il codice:

```
publisher_list = []
```

Adesso andiamo ad aggiungere (senza duplicati tutti i possibili Publisher presenti nel Dataset per ogni elemento):

```
for elem in df.Publisher:
    if elem not in publisher_list:
        publisher_list.append(elem)
```

A questo punto creiamo una nuova feature nel dataset per ogni publisher (andando a scorrere l'array dei publisher) inizialmente settata a 0:

```
for elem in developer_list:
    df[elem] = 0
```

Infine, tramite un ciclo, controlliamo se ogni element appartiene realmente alla nuova feature assegnandogli 1 o lasciando 0 in caso contrario:

```
for elem in df.index.get_values():
    df.set_value(elem, df.get_value(elem, "Publisher"), 1)
```

```
      # Eliminazione feature discretizzate

      del df['Name']
      del df['Platform']
      del df['Genre']
      del df['Rating']
      del df['Publisher']
      del df['Developer']
```

Una volta che abbiamo effettuato una discretizzazione per ogni feature che hanno, adesso, come valori delle variabili binarie, abbiamo ancora presenti nel dataset le features originarie con variabili continue (stringhe), dobbiamo quindi eliminarle poiché come abbiamo già largamente discusso non sono utili ai nostri scopi ma anzi, fanno restituire un errore di sintassi all'interprete.

```
ValueError: could not convert string to float: Nintendo
```

```
      # Eliminazione feature con elementi inferiori a 2

      del df['Rating_Adult']
```

Come abbiamo già analizzato nell'introduzione, un campione di un solo elemento non è né ragionevolmente trattabile a fini scientifici né è possibile farlo: pur volendo, non è possibile dividere il campione in un training set e un validation set (argomenti che vedremo poco più avanti).

```
      # Conversione di tutti i valori degli elementi in float

      df = df.astype('float64')
```

Per l'elaborazione del programma non è necessario effettuare una conversione dal tipo numerico corrente di ogni feature per ogni elemento in quanto Python riesce ad effettuare la conversione autonomamente.
Tuttavia, per evitare possibili problemi di compatibilità tra il progetto attuale ed eventuali aggiornamenti futuri, o tra le varie versioni di sistemi operativi su cui è possibile far girare il progetto, abbiamo deciso di aggiungere la precedente riga di codice per evitare risvolti negativi.

```
# Mischio in maniera casuale il dataset

df = df.sample(frac=1)
```

Randomizzare l'ordine degli elementi nel dataset è un passo importante per attribuire rilevanza scientifica al progetto: permutando gli elementi rendiamo più veritieri i risultati in quanto evitiamo di effettuare analisi e calcoli su situazioni predeterminate o sempre uguali.

5. Preparazione del Training Set e del Validation Set

```
print("Binary rating values\n")

df2 = df
```

Nella porzione di codice soprastante, innanzitutto indichiamo con una flag tramite print l'avvio dell'analisi e dei calcoli a cui abbiamo fino ad ora soltanto preparato il dataset.

```
y = df2['Rating_Everyone'].values
```

Come primo passo, creiamo una variabile che è l'esatta copia del dataset (ovviamente non dell'originale ma di quello che abbiamo appena elaborato).
Questo passo può risultare utile per non perdere di vista e soprattutto per non sovrascrivere la nostra variabile df che manteniamo come punto di riferimento.

La nostra `y`, ovvero la colonna (feature) di valori che vogliamo predire è Rating, ovvero la classificazione ESRB di ogni riga (elemento / videogioco).
Tuttavia, la feature Rating rientrava tra le feature che sono state discretizzate; aquesto punto non abbiamo più una sola colonna Rating, in realtà ne abbiamo quattro:
Rating_Everyone, Rating_Everyone10, Rating_Teen e Rating_Mature (una delle cinque originarie, Rating_Adult, è stata scartata per insufficienza di elementi).

```
df2 = df2.drop(['Rating_Everyone'],axis=1)
df2 = df2.drop(['Rating_Everyone10'],axis=1)
df2 = df2.drop(['Rating_Teen'],axis=1)
df2 = df2.drop(['Rating_Mature'],axis=1)
```

Salviamo allora una delle quattro nuove colonne nella suddetta variabile `y`. È necessario aggiungere il `.values` per salvare proprio i singoli valori in una lista convertendoli in float, altrimenti Python ancora una volta darebbe problemi di conversione.

```
X = df2.values
```

Dopo aver salvato l'output e impedito che l'algoritmo prenda in input (parte della y) i valori delle classi complementari a quella che vogliamo predire tramite le drop, dobbiamo effettivamente passare gli input all'algoritmo affinché possa elaborarli e restituire gli output desiderati.
Per farlo creiamo una variabile X che contenga tutti i valori del dataset eccetto i valori provenienti dall'originaria feature Ratings.

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.20)
```

`train_test_split` è una funzione standard del pacchetto `sklearn`:

```
sklearn.model_selection.train_test_split(*arrays, **options)
```

Permette di separare casualmente più array o, come nel nostro caso, matrici, in più parti.
È quindi uno strumento veloce per separare input e output a seconda di come viene specificato nel parametro options, nella nostra situazione andiamo a suddividere in training set e test set, assegnando a quest'ultimo il 20% del totale e al primo la parte restante.
Le operazioni descritte nel capitolo 5 e nei seguenti capitoli 6, 7 e 8, verranno ripetute nuovamente per l'applicazione dei metodi trattati anche sulle classi Rating_Everyone10+, Rating_Teen e Rating_Mature, infatti l'unica differenza nel codice sarà la sostituzione della riga

```
y = df2['Rating_Everyone'].values
```

con le rispettive

```
y = df2['Rating_Everyone'].values
y = df2['Rating_Everyone'].values
y = df2['Rating_Everyone'].values
```

In queste righe quindi, proprio per non rendere eccessivamente tediosa la relazione, ci siamo impegnati a ricordarvi questo aspetto.

Sbagliando si impara…
Durante la fase di realizzazione del codice, e all'avvio di una primissima esecuzione dello stesso, abbiamo rilevato un comportamento errato da parte di Logistic Regression e Random Forest: riportavano rispettivamente dei valori anomali pari a 99.9% e 97.7% di accuracy.
Come mai? La risposta è stata presto individuata dopo una rapida rilettura del codice: non erano stati ancora rimosse le componenti della y complementari a quella di cui si stava effettuando la classificazione. Il modello in questa situazione (assolutamente erronea) viene ovviamente aiutato in maniera esponenziale e, soprattutto, come non dovrebbe.

```
WRONG Binary rating values

('Logistic Regression Rating_Everyone accuracy: ', 0.99926632428466622)
('Logistic Regression Rating_Everyone misclassification: ', 1)
('Random Forest Rating_Everyone accuracy: ', 0.97725605282465156)
('Random Forest Rating_Everyone misclassification: ', 31)
('K-Nearest Neighbors Rating_Everyone accuracy: ', 0.79897285399853268)
('K-Nearest Neighbors Rating_Everyone misclassification: ', 274)
```

In questa situazione, tuttavia, non ci siamo limitati a correggere il problema. Ma ci siamo posti una nuova domanda: come mai k-NN non raggiunge un livello così buono?

Il k-NN può essere definito, sotto questo aspetto, "pigro". [13] Infatti non sfrutta il training set per elaborare una funzione discriminante, bensì lo memorizza.
D'altra parte, la fase di predizione nel k-NN è particolarmente esosa di risorse: ogni volta che un utente necessita di un'elaborazione per realizzare una previsione, il k-NN va a cercare i vicini più prossimi nell'intero training set (come vedremo più avanti in realtà ci sono strategie per accelerare un po' questo processo tramite BallTrees e KDtrees, ma il succo del discorso non cambia in quanto per quanto si possa accelerare questo processo il problema permane).
Insomma: mentre Logistic Regression e Random Forest hanno una fase di apprendimento e un model fitting, il k-NN ne è sprovvisto e per tanto non può beneficiare di un aiuto importante come la conoscenza delle classi a cui un certo elemento non appartiene.

6. Logistic Regression

Qui di seguito, vedremo trattati alcuni degli argomenti principali del corso di Metodi Quantitativi per l'Informatica, nel nostro caso seguono i classificatori Logistic Regression, Random Forest e k-NN.

Introduzione alla Logistic Regression
La Logistic Regression è una generalizzazione della Linear Regression (dove $y \in \mathbb{R}$) nel caso in cui la y appartenga ai valori 0 oppure 1 ($y \in \{0, 1\}$). È un modello di regressione dove la variabile dipendente è di tipo categorico, ovvero può assumere determinati valori corrispondenti a determinate classi. Nel nostro caso, la variabile dipendente è binaria e può assumere solo due valori, ovvero se l'elemento (videogioco) in analisi appartiene o meno a una determinata classe (fascia d'età assegnata dall'ESRB).
La Logistic Regression viene utilizzata in molti campi, ad esempio nel campo medico può essere utilizzata per predire se un paziente ha o meno una particolare malattia basandosi su alcune caratteristiche (peso, età, altezza, risultati delle analisi del sangue, ecc.).
Come altre forme di analisi della regressione, la Logistic Regression fa uso di una o più variabili indipendenti che possono essere o continue o categoriche. A differenza della Linear Regression, comunque, la Logistic Regression viene utilizzata per predire variabili dipendenti binarie, trattando la variabile dipendente come il risultato di una distribuzione di Bernoulli, piuttosto che un esito continuo.

Spiegazione della Logistic Regression
Una spiegazione della Logistic Regression può cominciare a partire dalla Logistic function, ovvero una comune funzione sigmoidea (a forma di "S"), caratterizzata dall'equazione:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Può prendere come parametri qualsiasi input t, ($t \in \mathbb{R}$) e restituisce come output valori compresi tra 0 e 1, che possono essere quindi interpretati come una probabilità.
Assumiamo t come una funzione lineare che può essere espressa in funzione di una singola variabile indipendente x. Possiamo quindi esprimere t come segue:

$$t = \beta_0 + \beta_1 x + \beta_0 + \beta_1 x + \cdots + \beta_k x_k = X\beta$$

e la Logistic function così:

$$F(x) = \frac{1}{1 + e^{-X\beta}}$$

La stima della probabilità di $F(x)$ avviene effettuando prima la stima dei parametri β (utilizzando il metodo della massima verosimiglianza).
Adesso possiamo esprimere la funzione inversa della logistic, il logit:

$$g(F(x)) = \ln\left(\frac{F(x)}{1-F(x)}\right) = \beta_0 + \beta_1 x + \cdots + \beta_k x_k = X\beta$$

e di conseguenza, elevando a potenza ambo i membri:

$$\frac{F(x)}{1-F(x)} = e^{X\beta}$$

Nelle ultime tre equazioni abbiamo quindi definito:

- $F(X)$: la probabilità (ovviamente compresa tra 0 e 1) che l'evento y si verifichi;
- $e^{X\beta}$: l'odds, ovvero il rapporto tra la probabilità di un evento (p) e la probabilità che tale evento non accada $(1-p)$, il logaritmo naturale dell'odds è proprio il logit. Il rapporto tra due odds è detto odds ratio e viene utilizzato, come accennato precedentemente, nella statistica sanitaria.[14]
- g: è il logit, una funzione che si applica a valori compresi nell'intervallo (0,1), tipicamente valori rappresentanti probabilità.[15] È la funzione link della Logistic Regression.

Benché sia tecnicamente possibile applicare alla variabile y la regressione lineare

$$y = \alpha_0 + \alpha_1 x_1 + \cdots + \alpha_k x_k = X\alpha$$

ciò viene evitato in quanto porterebbe in generale a stime che vanno da meno infinito a più infinito e dunque fuori dall'intervallo [0,1] previsto per le probabilità.

A seguire nelle porzioni di codice, ci sarà la spiegazione dettagliata e minuziosa di particolari dei parametri dei vari modelli. Vedremo quindi esplicitati, oltre al parametro stesso, anche il valore da noi individuato e le motivazioni che ci hanno portato ad effettuare tale scelta.
Per cercare di raggiungere i risultati migliori, abbiamo provato tutte le combinazioni possibili tra i vari parametri seguendo la tecnica del Coarse-to-fine.[16]

Applicazione della Logistic Regression

```
log_reg = LogisticRegression(penalty='l1', dual=False, C=1.0,
          fit_intercept=True, intercept_scaling=1,
          class_weight=None, random_state=None, solver='liblinear',
          max_iter=100, multi_class='ovr',
          verbose=0, warm_start=False, n_jobs=-1)
```

Tramite questa funzione andiamo ad applicare proprio la Logistic Regression, che approfondiremo più avanti, sul nostro dataset.
Intanto, possiamo analizzare la funzione `sklearn.linear_model.LogisticRegression`[17] nel dettaglio:

- Penalty: fornisce la possibilità di controllare le proprietà dei coefficienti della regressione, oltre alla semplice massimizzazione della funzione di verosimiglianza, si può pensare di ottimizzare anche la penalty.
  Può essere di tipo L1 o L2. L1 migliora le performance nel caso di sparse feature spaces (che presentano numerosi 0).

Nel nostro caso abbiamo utilizzato quindi L1 e riportiamo qui di seguito il nostro esperimento. Facciamo notare la possibilità di ristampare quanto segue (ovviamente elaborando nuovi risultati) impostando la variabile globale `RUN_ANALYSIS_LOGISTIC_PENALTY` a True.

L1:
```
('Logistic Regression Rating_Everyone misclassification: ', 193)
('Logistic Regression Rating_Everyone10+ misclassification: ', 148)
('Logistic Regression Rating_Teen misclassification: ', 287)
('Logistic Regression Rating_Mature misclassification: ', 175)
```
misclassification: 803
L2:
```
('Logistic Regression Rating_Everyone misclassification: ', 206)
('Logistic Regression Rating_Everyone10+ misclassification: ', 173)
('Logistic Regression Rating_Teen misclassification: ', 313)
('Logistic Regression Rating_Mature misclassification: ', 180)
```
misclassification: 860

- Dual: è la formulazione che può essere implementata. Può essere primale o duale, quest'ultima è compatibile soltanto con la penalty L2 e il solver liblinear. Viene raccomandato quando si hanno n_samples > n_features.[18]
  Nel nostro caso abbiamo scelto False poiché non compatibile con L1.
- C: è una regolarizzazione artificiale, ovvero, se è attiva, l'algoritmo cerca di trovare una correlazione tra le feature degli elementi presenti nel dataset.

  Se ad esempio abbiamo:
  Miles, dog, small, FRIENDLY
  Fifi, cat, small, ENEMY
  Muffy, cat, small, ENEMY
  Rufus, dog, large, FRIENDLY
  Aloysius, dog, large, FRIENDLY
  Tom, cat, large, ENEMY

  l'algoritmo identificherà i cani come amici e i gatti come nemici.
  Nel nostro caso non esiste alcuna relazione del genere tra le varie feature di un dato elemento, e pertanto C è stato lasciato di default a 1.0.[19]
- Fit_intercept: specifica se l'intercept term (ovvero il valore atteso della y quando tutte le X sono pari a 0) deve essere aggiunto alla funzione di decisione.
  In pratica, è possibile rimuoverlo quando si è sicuri che sia anch'esso pari a 0 o quando si è certi che le X non saranno mai tutte uguali a 0.
  Nel nostro caso abbiamo una vastissima quantità di X che sono uguali e diverse da 0, perciò abbiamo impostato il fit_intercept su True.[20]
- Intercept_scaling: nel caso in cui fit_intercept sia settato su True, serve per diminuire il peso di quest'ultimo sul risultato finale che ha l'algoritmo.
  Nel nostro caso abbiamo già discusso l'importanza dell'intercept term e pertanto abbiamo deciso di non diminuire il suo peso.
- Class_weight: serve per attribuire un peso diverso per le varie classi.
  Nel nostro caso ci è parso ragionevole lasciare una rilevanza eguale per tutte le classi, in quanto non è emerso nessuna motivazione sul perché una classe dovesse essere privilegiata o svantaggiata rispetto alle altre.

- Random_state: serve per mischiare i dati prima dell'applicazione.
  Nel nostro caso abbiamo già permutato gli elementi del dataset e successivamente suddiviso quest'ultimo in training set e validation set costituendoli a partire da elementi presi di nuovo in maniera casuale.
- Solver: sono i diversi metodi di applicazione della logistic regression.
  liblinear è consigliato per i piccoli dataset, in quanto è il più preciso ma anche il più lento, mentre sag al contrario è consigliato per quelli più grandi in quanto più rapido ma anche più superficiale.
  Per problemi che trattano le multiclassi, possono essere utilizzati esclusivamente newton-cg, sag e lbfgs.
  Nel nostro caso quindi, avendo più classi su cui applicare la logistic regression ma ognuna trattata singolarmente, la scelta poteva ricadere soltanto su liblinear e sag. Inizialmente abbiamo quindi provato con sag, in quanto abbiamo ritenuto il nostro dataset sufficientemente grande.
  Tuttavia, abbiamo riscontrato l'impossibilità di applicare tale solver con la penalty L1:

  ```
  ValueError: Solver sag supports only l2 penalties, got l1 penalty.
  ```

  Volendo procedere proprio con la tecnica del Coarse-to-fine abbiamo quindi cercato di procedere correggendo l'errore provando a reimpostare L2, anche se questo avrebbe, come già dimostrato, aumentato il missclassification rate nel caso del solver liblinear.
  Tuttavia, questa volta, abbiamo ottenuto un nuovo errore:

  ```
  The max_iter was reached which means the coef_ did not converge.
  ```

  Studiando la nuova situazione abbiamo verificato che in alcuni casi il modello può non raggiungere la convergenza. La non-convergenza di un modello infatti indica che i coefficienti sono "privi di significato", ovvero che il processo iterativo non è stato in grado di trovare delle soluzioni appropiate; per risolvere questo problema è necessaria l'applicazione di un solver che sia più approfondito anche se meno rapido.
  Per risolvere questo inconveniente della convergenza non ci è rimasto che reimpostare il solver liblinear e di conseguenza la L1 penalty.

- Max_iter: è un parametro di rilevale importanza per i solver newton-cg, sag e lbfgs, in quanto rappresenta il numero massimo di iterazioni di cui può avvalersi il modello per raggiungere la convergenza.
  Come abbiamo già analizzato nel punto precedente, non siamo riusciti a raggiungere la convergenza con il solver sag.
  Abbiamo così incrementato il numero di iterazioni massimo dallo standard 100 fino a 500 iterazioni massime, tuttavia abbiamo riscontrato esclusivamente un tempo di caricamento superiore e non la soluzione al problema.
  Nello specifico, con il solver = 'liblinear' e il max_iter=100 abbiamo riscontrato un tempo di elaborazione pari a 15 secondi; con il solver = 'sag' e il max_iter = 500 abbiamo riscontrato un tempo di elaborazione pari a 60 secondi e il verificarsi nuovamente del problema della convergenza che non veniva raggiunta.
  Si poteva quindi procedere ad incrementare ulteriormente il max_iter. Tuttavia, già con 500 iterazioni il solver sag superava notevolmente il tempo richiesto dal solver liblinear per risolvere il problema; ma il vantaggio del solver sag (per dataset sufficientemente grandi) dovrebbe essere proprio quello di un tempo di elaborazione inferiore al costo di una

accuratezza inferiore rispetto al solver liblinear. Visto che i tempi di elaborazione risultano superiori, chiaramente non è necessario procedere con ulteriori tentativi per far funzionare il solver sag che a questo punto viene definitivamente scartato.
Nel nostro caso quindi è stato lasciato ad un valore di default pari a 100.

- Multi_class: è un parametro che serve per regolare il peso delle multiclassi. Nel nostro caso quindi non viene utilizzato e lasciato al parametro 'ovr'.
- Warm_start: quando è settato a True riutilizza le soluzioni delle precedenti chiamate a funzione per effettuare una nuova elaborazione, altrimenti semplicemente cancella i risultati una volta restituiti.
- Nel nostro caso non abbiamo voluto creare dei precedenti e pertanto abbiamo impostato il valore a False.
- N_jobs: rappresenta semplicemente il numero di core della CPU utilizzati durante l'elaborazione, se viene impostato a -1 vengono utilizzati tutti i core.
  Nel nostro caso abbiamo deciso di sfruttare al massimo la CPU impostando a -1 il parametro.

```
log_reg.fit(Xtrain, ytrain)
y_val_l = log_reg.predict(Xtest)
```

In scikit-learn uno stimatore per la classificazione è un oggetto Python che implementa i metodi `fit(X, y)` e `predict(T)`.
Tutti gli stimatori per il supervised learning implementano il primo di questi due metodi per adattarsi al modello e il secondo per predire il valore della y dato un certo array di valori X.[21]

```
ris = accuracy_score(ytest, y_val_l)
mis = accuracy_score(ytest, y_val_l, normalize=False)
print("Logistic Regression Rating_Everyone accuracy: ", ris)
print("Logistic Regression Rating_Everyone misclassification: ",
ytest.size - mis)
```

La funzione `accuracy_score` rappresenta il metodo standard fornito con sklearn per calcolare la percentuale di accuratezza che ha ottenuto la predizione.
Al contrario se si desidera un misclassification rate sarà sufficiente sottrarre l'accuratezza a cento.
Nel nostro caso abbiamo deciso di stampare a schermo l'accuratezza in percentuale del progetto e il numero esatto di volte che la predizione è fallita, normalizzando l'accuratezza otteniamo il numero di successi, che sottratto alla dimensione del test set ci dà il numero esatto di volte in cui la predizione è fallita.

7. Random Forest

Introduzione alla Random Forest
Le Random Forest sono un metodo di apprendimento d'insieme (utilizzato per la classificazione, regressione e altro) che operano costruendo una moltitudine di alberi di decisione durante l'allenamento del modello e dando come output la classe che è la moda delle classi (nel caso della classificazione) oppure la predizione della media (nel caso della regressione) dei singoli alberi.
Le Random Forest correggono l'abitudine degli alberi di decisione di fare overfitting rispetto al proprio training set. In particolare, gli alberi che sono cresciuti molto in profondità tendono a imparare pattern altamente irregolari: si adattano ai loro training set, hanno un bias basso ma una varianza alta. Le Random Forest sono un modo per fare una media tra più alberi di decisione, allenati in parti diverse dello stesso training set, con l'obiettivo di ridurne la varianza[22]. Tutto questo si

traduce in un piccolo aumento del bias e qualche perdita di interpretabilità, ma generalmente aumenta di molto le performance del modello finale.

Spiegazione della Random Forest

L'algoritmo di allenamento delle Random Forests applica la tecnica generale del bagging ai tree learners. Dato un training set $X = x_1, \dots, x_n$ a cui corrisponde una $Y = y_1, \dots, y_n$, il bagging seleziona ripetutamente per B volte un campione casuale con sostituzione dal training set e adatta gli alberi a questi campioni. Quindi:

Per $b = 1, \dots, B$:

1. Campiona, con sostituzione, B esempi da X, Y e li chiama X_b, Y_b.
2. Allena un albero f_b (di decisione o di regressione) su X_b, Y_b.

Dopo l'allenamento, le predizioni per i campioni non visitati x' possono essere fatte svolgendo una media sulle predizioni di tutti gli alberi di regressione su x': [23]

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

Oppure prendendo la maggioranza dei voti nel caso degli alberi di decisione.

Questa procedura di bootstrap conduce a performance migliori del modello perché diminuisce la varianza senza aumentare il bias. Ciò significa che mentre le predizioni di un singolo albero sono molto sensibili al rumore nel proprio training set, la media di molti alberi non lo è, purché gli alberi non sono correlati. Allenare semplicemente molti alberi in un singolo training set risulterebbe in alberi fortemente correlati tra loro (o addirittura lo stesso albero ripetuto più volte, se l'algoritmo è di tipo deterministico); il campionamento di tipo bootstrap è un modo per slegare i vari alberi mostrando loro diversi training set.

Il numero di campioni/alberi, ovvero B, è un parametro libero che dipende dalla natura e dimensione del training set; un numero ottimale di alberi B può essere trovato mediante cross-validation oppure osservando l'errore out-of-bag.

La procedura appena descritta viene utilizzata in linea generale per gli algoritmi di bagging. Le Random Forests differiscono solamente in un aspetto da questo schema: utilizzano un algoritmo modificato di apprendimento dell'albero che seleziona, ad ogni candidate-split, un sottoinsieme casuale di features. Questo processo viene talvolta chiamato "feature bagging". La ragione dietro ciò risiede nella correlazione degli alberi in un campionamento ordinario del boostrap: se una o più feature sono predictors molto forti per l'output, verrebbero selezionati in molti tra i B alberi, rendendoli di fatto correlati tra loro.

Applicazione della Random Forest

```
radm = RandomForestClassifier(n_estimators=240, criterion='gini',
        max_depth=None, min_samples_split=2, min_samples_leaf=1,
        min_weight_fraction_leaf=0.0, max_features='auto',
        max_leaf_nodes=None, min_impurity_split=1e-07, bootstrap=True,
        oob_score=True, n_jobs=-1, random_state=None, verbose=0,
        warm_start=False, class_weight=None)
```

Tramite questo metodo Python del pacchetto sklearn andiamo ad applicare il metodo Random Forest sul nostro dataset.

Approfondiremo in seguito il Random Forest, nel frattempo passiamo allo studio della funzione `sklearn.ensemble.RandomForestClassifier` [24] nel dettaglio:

- N_estimator: indica il numero di alberi nella foresta.
  Per trovare il valore ideale, abbiamo effettuato delle combinazioni con gli altri parametri da modificare. Con il criterio entropy, abbiamo scelto 20 in quanto, sperimentando il metodo del quadrato lineare, abbiamo individuato in n_estimator = 40 un'assenza del miglioramento desiderato rispetto al risultato ottenuto con 20; tuttavia questo parametro, come abbiamo detto poco fa, verrà modificato nuovamente quando imposteremo il criterio gini.
  Di seguito riportiamo 6 esecuzioni consecutive del programma che ci hanno portato all'appena citata analisi. Facciamo notare che è possibile ristampare il seguente codice impostando la variabile globale `RUN_ANALYSIS_RANDOMFOREST_NUMBERESTIMATOR` a True.

```
n_estimator = 10:
('Random Forest Rating_Everyone misclassification: ', 212)
('Random Forest Rating_Everyone10+ misclassification: ', 138)
('Random Forest Rating_Teen misclassification: ', 354)
('Random Forest Rating_Mature misclassification: ', 195)
misclassification: 899

n_estimator = 10:
('Random Forest Rating_Everyone misclassification: ', 223)
('Random Forest Rating_Everyone10+ misclassification: ', 176)
('Random Forest Rating_Teen misclassification: ', 358)
('Random Forest Rating_Mature misclassification: ', 171)
misclassification: 928

n_estimator = 20:
('Random Forest Rating_Everyone misclassification: ', 198)
('Random Forest Rating_Everyone10+ misclassification: ', 166)
('Random Forest Rating_Teen misclassification: ', 298)
('Random Forest Rating_Mature misclassification: ', 194)
misclassification:  856

n_estimator = 20:
('Random Forest Rating_Everyone misclassification: ', 190)
('Random Forest Rating_Everyone10+ misclassification: ', 159)
('Random Forest Rating_Teen misclassification: ', 330)
('Random Forest Rating_Mature misclassification: ', 185)
misclassification: 864


n_estimator = 40:
('Random Forest Rating_Everyone misclassification: ', 197)
('Random Forest Rating_Everyone10+ misclassification: ', 165)
('Random Forest Rating_Teen misclassification: ', 346)
('Random Forest Rating_Mature misclassification: ', 181)
misclassification:  889

n_estimator = 40:
('Random Forest Rating_Everyone misclassification: ', 210)
('Random Forest Rating_Everyone10+ misclassification: ', 157)
('Random Forest Rating_Teen misclassification: ', 346)
('Random Forest Rating_Mature misclassification: ', 188)
misclassification:  901
```

- Criterion: è la funzione per misurare la qualità di una divisione di un sottoalbero. I criteri supportati sono "gini" e "entropy".
  Gini:$i(N) = -\sum_j P(\omega_j) \log_2 P(\omega_j)$
  Entropy:$i(N) = \sum_{i \neq j} P(\omega_i)\, P(\omega_j) = \frac{1}{2}[1 - \sum_j P^2(\omega_j)$

Nel nostro caso con entropy abbiamo inizialmente riscontrato peggioramenti rispetto a gini: nelle sei esecuzioni precedenti è stato utilizzato proprio il criterio gini, nelle quattro successive iterazioni è stato utilizzato entropy, e come potete notare non una sola iterazione delle seguenti ha superato i risultati ottenuti con gini a n_estimator = 20. Ancora una volta è possibile visualizzare quanto segue in tempo reale impostando RUN_ANALYSIS_RANDOMFOREST_ENTROPY a True.

```
n_estimator = 20:
('Random Forest Rating_Everyone misclassification: ', 227)
('Random Forest Rating_Everyone10+ misclassification: ', 157)
('Random Forest Rating_Teen misclassification: ', 314)
('Random Forest Rating_Mature misclassification: ', 182)
misclassification:  880

n_estimator = 20:
('Random Forest Rating_Everyone misclassification: ', 199)
('Random Forest Rating_Everyone10+ misclassification: ', 170)
('Random Forest Rating_Teen misclassification: ', 347)
('Random Forest Rating_Mature misclassification: ', 164)
misclassification:  880

n_estimator = 20:
('Random Forest Rating_Everyone misclassification: ', 224)
('Random Forest Rating_Everyone10+ misclassification: ', 154)
('Random Forest Rating_Teen misclassification: ', 329)
('Random Forest Rating_Mature misclassification: ', 171)
misclassification:  878

n_estimator = 20:
('Random Forest Rating_Everyone misclassification: ', 228)
('Random Forest Rating_Everyone10+ misclassification: ', 159)
('Random Forest Rating_Teen misclassification: ', 322)
('Random Forest Rating_Mature misclassification: ', 192)
misclassification:  901
```

Tuttavia, abbiamo ritenuto opportuno approfondire la vicenda, incrementando nuovamente gli n_estimator a 40 e provando nuovamente con entropy.
I risultati seguenti (ricalbolabili impostando la macro RUN_ANALYSIS_RANDOMFOREST_ENTROPYANDNE40 a True) ci hanno portato ad impostare proprio questa configurazione (nessuna delle seguenti iterazioni ha portato un solo risultato che fosse peggiore (anzi hanno dato esclusivamente risultati migliori) di tutti quelli ottenuti fino ad ora nelle precedenti 10 iterazioni).

```
n_estimator = 40:
('Random Forest Rating_Everyone misclassification: ', 193)
('Random Forest Rating_Everyone10+ misclassification: ', 133)
('Random Forest Rating_Teen misclassification: ', 316)
('Random Forest Rating_Mature misclassification: ', 178)
misclassification:  820

n_estimator = 40:
('Random Forest Rating_Everyone misclassification: ', 207)
('Random Forest Rating_Everyone10+ misclassification: ', 150)
('Random Forest Rating_Teen misclassification: ', 318)
('Random Forest Rating_Mature misclassification: ', 176)
misclassification:  851
```

```
n_estimator = 40:
('Random Forest Rating_Everyone misclassification: ', 195)
('Random Forest Rating_Everyone10+ misclassification: ', 154)
('Random Forest Rating_Teen misclassification: ', 320)
('Random Forest Rating_Mature misclassification: ', 175)
misclassification:  844

n_estimator = 40:
('Random Forest Rating_Everyone misclassification: ', 185)
('Random Forest Rating_Everyone10+ misclassification: ', 147)
('Random Forest Rating_Teen misclassification: ', 326)
('Random Forest Rating_Mature misclassification: ', 171)
misclassification:  829
```

A questo punto, abbiamo ritenuto necessario, seguendo sempre la tecnica del Coarse, di incrementare linearmente il `n_estimator`, finché non abbiamo incontrato un peggioramento nelle configurazioni `n_estimator = 280` e `n_estimator = 320`, riportandoci all'ultima configurazione migliore riscontrata pari a 240.

- Max_features: è il numero di features da considerare quando considerare quando occorre effettuare lo divisione in sottoalberi.
  Nel nostro caso non abbiamo trovato validi motivi per non considerare tutte le features.
- Max_dept: è la profondità massima che può raggiungere un albero.
  Nel nostro caso ancora una volta non abbiamo trovato validi motivi per limitare questa possibilità tipica del Random Forest.
- Min_samples_split: è il numero minimo di elementi richiesti per effettuare la divisione di un albero in più sottoalberi a partire da un nomo.
  Nel nostro caso abbiamo deciso di mettere il minimo possibile per effettuare una divisione (due) in quanto vogliamo che la Random Forest abbia tutta la libertà possibile.
- Min_samples_leaf: è il numero minimo di elementi richiesti affinché un nodo sia la foglia di un albero.
  Nel nostro caso abbiamo scelto 1, ancora una volta il minimo possibile per garantire alla Random Forest la più ampia libertà che potevamo fornirle.
- Min_weight_fraction_leaft: permette di pesare in maniera differente le varie foglie attribuendo un peso minimo.
  Nel nostro abbiamo scelto 0, ancora una volta il minimo possibile.
- Max_leaf_nodes: è il numero massimo di foglie che un albero può raggiungere.
  Nel nostro caso non abbiamo trovato motivi per limitare questo numero.
- Min_impurity_split: è un valore tipico di ogni nodo. Se un nodo ha un livello di impurità superiore a questa soglia viene suddiviso in due nodi (creando due sottoalberi), altrimenti viene considerato come una foglia.
  Nel nostro caso abbiamo mantenuto il minimo consentito pari a 1e-7 in modo da consentire alla Random Forest di estendere i suoi alberi il più possibile e di conseguenza ridurre al minimo possibile il missclassification rate.
- Bootstrap: l'algoritmo applica la tecnica generale del bootstrap aggregating (or bagging) ai tree learners, che abbiamo ampiamente discusso e osservato nell'introduzione di questo paragrafo.
  Nel nostro caso abbiamo settato il parametro a True per sfruttare tale potenzialità.
- Oob_score: Dopo che ogni albero è cresciuto, i valori di un dato predictor vengono permutati casualmente nella out-of-bag e l'errore di predizione dell'albero nella modificata OOB è

comparato con la OOB originale. Questo processo viene ripetuto per tutti gli input e viene fatta una media tra tutti gli alberi in modo da ottenere un valore proporzionale all'accuratezza ottenuta nell'esecuzione di questo processo.
Infine, le variabili vengono classificate proporzionalmente alla diminuzione globale dell'accuratezza che la loro permutazione ha indotto. [25]
Le variabili più importanti sono quelle che conducono alle maggiori perdite in termini di accuratezza.[26]
Anche in questo caso abbiamo impostato il parametro a True per sfruttarne le possibilità.

- Verbose: serve per mostrare informazioni (talvolta prolisse) per una certa attività. Impostando una verbose superiore a 2 è possibile visualizzare ulteriori informazioni sul processo di costruzione dell'albero.
  Tali informazioni non sono necessarie per l'analisi dei dati che ci riguarda, pertanto abbiamo impostato il valore del parametro a 0.
- Class_weight: Abbiamo provato a metterci nei panni della Random Forest. Effettivamente non abbiamo trovato nessuna motivazione plausibile per cui sulla carta una classe dovesse avere un peso più significativo più di un'altra.
- N_jobs: vale lo stesso discorso steso per la Logistic Regression.
- Random_state: idem.
- Warm_start: idem.

```
radm.fit(Xtrain, ytrain)
y_val_l = radm.predict(Xtest)
ris = accuracy_score(ytest, y_val_l)
mis = accuracy_score(ytest, y_val_l, normalize=False)
print("Random Forest Rating_Everyone accuracy: ", ris)
print("Random Forest Rating_Everyone misclassification: ", ytest.size - mis)
```

Vale un discorso analogo a quanto già esplicato nel capitolo dedicato alla Logistic Regression.

8. k-Nearest Neighbors (k-NN)

Introduzione al k-NN
Il k-nearest neighbor (k-NN) è un algoritmo utilizzato nel riconoscimento di pattern per la classificazione di oggetti basandosi sulle caratteristiche degli oggetti vicini a quello considerato. È l'algoritmo più semplice fra quelli utilizzati nel Machine Learning. Un oggetto viene classificato in base alla maggioranza dei voti dei suoi k vicini. k è un intero positivo tipicamente non molto grande. Se $k = 1$ allora l'oggetto viene assegnato alla classe del suo vicino. In un contesto binario in cui sono presenti esclusivamente due classi è opportuno scegliere k dispari per evitare di ritrovarsi in situazioni di parità.

Più formalmente:

$$p(y = c \mid \boldsymbol{x}, D, k) = \frac{1}{N} \sum_{i \in N_k(x,D)} \mathbb{I}(y_i = c)$$

Dove:

- $N_k(x, D)$ sono i k punti più vicini ad x.
- $\mathbb{I}(e) = \begin{cases} 1, & \text{se } e = \text{True} \\ 0, & \text{se } e = \text{False} \end{cases}$ è la funzione indicatrice.

Questo metodo può essere utilizzato per la tecnica di regressione assegnando all'oggetto la media dei valori dei k oggetti suoi vicini.
Considerando solo i voti dei k oggetti vicini c'è l'inconveniente dovuto alla predominanza delle classi con più oggetti. Infatti in questo caso può risultare utile pesare i contributi dei vicini in modo da dare, nel calcolo della media, maggior importanza in base alla distanza dall'oggetto considerato.

Spiegazione del k-NN
La scelta di k dipende dalle caratteristiche dei dati. Generalmente all'aumentare di k si riduce il rumore che compromette la classificazione, ma il criterio di scelta per la classe diventa più labile. La scelta può esser presa attraverso tecniche di euristica come ad esempio la cross-validation. Durante la fase di training, lo spazio viene partizionato in regioni in base alle posizioni e alle caratteristiche degli oggetti di training. Questo può essere considerato come il training-set per l'algoritmo anche se non è esplicitamente richiesto dalle condizioni iniziali.
Ai fini del calcolo della distanza gli oggetti sono rappresentati attraverso vettori di posizione in uno spazio multidimensionale. Di solito viene usata la distanza euclidea, ma anche altri tipi di distanza sono ugualmente utilizzabili: ad esempio la distanza di Manhattan e la distanza di Hamming (nel caso in cui si debbano manipolare stringhe e non numeri). Infatti l'algoritmo è sensibile alla struttura locale dei dati.
Infine, per quanto riguarda la classificazione, un punto (che rappresenta un oggetto) è assegnato alla classe C se questa è la più frequente fra i k esempi più vicini all'oggetto sotto esame e la vicinanza si misura in base alla distanza fra punti. I vicini sono presi da un insieme di oggetti per cui è nota la classificazione corretta. Nel caso della regressione per il calcolo della media (classificazione) si usa il valore della proprietà considerata.

Applicazione del k-NN

```
knn = KNeighborsClassifier(n_neighbors=10, weights='uniform',
       algorithm='auto', leaf_size=30, n_jobs=-1)
```

Tramite questa funzione del pacchetto `sklearn.neighbors.KneighborsClassifier` [27] possiamo applicare il principio del k-Nearest Neighbors (k-NN).
Prima di addentrarci nei concetti teorici che vi si nascondo dietro, sarà nostra premura occuparci di spiegare la funzione.
Tra i parametri troviamo quindi:

- N_neighbors: rappresenta il numero di vicini da utilizzare per le indaggini del k-NN. Nel nostro caso abbiamo scelto di tenere in considerazione 10 vicini, poiché raddoppiando tale valore si è ottenuto un lieve miglioramento nel misclassification rate, tuttavia raddoppiandolo ulteriormente si riscontrava nel k-NN un aumento di incomprensione e, di conseguenza, del missclassification rate. È possibile comunque ristampare rapidamente quanto segue (chiaramente con una nuova esecuzione e quindi nuovi risultati) impostando a True la variabile globale `RUN_ANALYSIS_KNN_NUMBERNEIGHBORS`

  ```
  n_neighbors=5
  ('K-Nearest Neighbors Rating_Everyone misclassification: ', 350)
  ('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 196)
  ('K-Nearest Neighbors Rating_Teen misclassification: ', 485)
  ('K-Nearest Neighbors Rating_Mature misclassification: ', 255)
  misclassification: 1286
  ```

```
n_neighbors=5
('K-Nearest Neighbors Rating_Everyone misclassification: ', 349)
('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 199)
('K-Nearest Neighbors Rating_Teen misclassification: ', 490)
('K-Nearest Neighbors Rating_Mature misclassification: ', 256)
misclassification: 1294

n_neighbors=10
('K-Nearest Neighbors Rating_Everyone misclassification: ', 343)
('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 175)
('K-Nearest Neighbors Rating_Teen misclassification: ', 472)
('K-Nearest Neighbors Rating_Mature misclassification: ', 255)
misclassification: 1245

n_neighbors=10
('K-Nearest Neighbors Rating_Everyone misclassification: ', 339)
('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 179)
('K-Nearest Neighbors Rating_Teen misclassification: ', 468)
('K-Nearest Neighbors Rating_Mature misclassification: ', 256)
misclassification: 1242

n_neighbors=20
('K-Nearest Neighbors Rating_Everyone misclassification: ', 364)
('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 188)
('K-Nearest Neighbors Rating_Teen misclassification: ', 472)
('K-Nearest Neighbors Rating_Mature misclassification: ', 251)
misclassification: 1275

n_neighbors=20
('K-Nearest Neighbors Rating_Everyone misclassification: ', 358)
('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 191)
('K-Nearest Neighbors Rating_Teen misclassification: ', 480)
('K-Nearest Neighbors Rating_Mature misclassification: ', 255)
misclassification: 1284
```

- Weights: permette di attribuire un peso differente per ogni vicino. Nel nostro caso non abbiamo trovato motivazioni tali per cui un elemento o una feature dovesse avere più peso rispetto ad un altro, perciò abbiamo optato per un peso uniform.
- Algorithm: è l'algoritmo utilizzato per calcolare il Nearest Neighbors.
  Può essere utilizzato BallTree[28], kd-tree[29] o auto, per permettere all'interprete di scegliere autonomamente quale utilizzare a seconda della situazione.
  Nel nostro caso, visto che la presenza di vicini (e quanto esse lo siano realmente) rimane sempre affidato al caso per via delle permutazioni che vengono eseguite ogni volta, abbiamo deciso di utilizzare auto in modo da affidare all'interprete il compito di valutare quale sia il metodo da applicare.
- Leaf_size: stabilisce la dimensione delle foglie del BallTree o del KDTree. Il valore ottimale dipende dalla natura del problema.
  Nel nostro caso abbiamo lasciato il default 30, in quanto raddoppiando il valore della dimensione delle foglie si ottenevano risultati disastrosi, anche a differenza di n_neighbors.
  Si può effettuare un nuovo calcolo di quanto segue impostando a True la variabile globale `RUN_ANALYSIS_KNN_LEAFSIZE`.

```
n_neighbors=5
('K-Nearest Neighbors Rating_Everyone misclassification: ', 354)
('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 197)
```

```
('K-Nearest Neighbors Rating_Teen misclassification: ', 515)
('K-Nearest Neighbors Rating_Mature misclassification: ', 279)
misclassification: 1345

n_neighbors=10
('K-Nearest Neighbors Rating_Everyone misclassification: ', 339)
('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 186)
('K-Nearest Neighbors Rating_Teen misclassification: ', 495)
('K-Nearest Neighbors Rating_Mature misclassification: ', 284)
misclassification: 1304

n_neighbors=20
('K-Nearest Neighbors Rating_Everyone misclassification: ', 370)
('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 181)
('K-Nearest Neighbors Rating_Teen misclassification: ', 490)
('K-Nearest Neighbors Rating_Mature misclassification: ', 260)
misclassification: 1301
```

- P: fornisce la possibilità di modificare la metrica utilizzata per calcolare la distanza. Con 1 si sceglie la distanza di Manhattan, con 2 la distanza di Minkowski.
  Tutti i risultati fino ad ora sono stati calcolati con p=1.
  Impostiamo la variabile globale `RUN_ANALYSIS_KNN_P` per fare le opportune verifiche con p=2 e ci accorgeremo di risultati notevolmente peggiori.

```
n_neighbors=5 p=2
('K-Nearest Neighbors Rating_Everyone misclassification: ', 405)
('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 204)
('K-Nearest Neighbors Rating_Teen misclassification: ', 536)
('K-Nearest Neighbors Rating_Mature misclassification: ', 297)
misclassification: 1442

n_neighbors=10 p=2
('K-Nearest Neighbors Rating_Everyone misclassification: ', 353)
('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 203)
('K-Nearest Neighbors Rating_Teen misclassification: ', 551)
('K-Nearest Neighbors Rating_Mature misclassification: ', 259)
misclassification: 1366

n_neighbors=20 p=2
('K-Nearest Neighbors Rating_Everyone misclassification: ', 391)
('K-Nearest Neighbors Rating_Everyone10+ misclassification: ', 189)
('K-Nearest Neighbors Rating_Teen misclassification: ', 492)
('K-Nearest Neighbors Rating_Mature misclassification: ', 279)
misclassification: 1356
```

- N_jobs: come per Logistic Regression e Random Forest abbiamo impostato -1.

```
knn.fit(Xtrain, ytrain)
y_val_l = knn.predict(Xtest)
ris = accuracy_score(ytest, y_val_l)
mis = accuracy_score(ytest, y_val_l, normalize=False)
print("k-NN Rating_Everyone accuracy: ", ris)
print("k-NN Forest Rating_Everyone misclassification: ", ytest.size - mis)
```

Vale un discorso analogo a quanto già esplicato nel capitolo dedicato alla Logistic Regression.

9. Ripetizione delle applicazioni dei modelli per le altre classi

A questo punto, dopo aver applicato Logistic Regression, Random Forest e k-NN sulla classe Rating_Everyone, l'operazione verrà ripetuta per le classi Rating_Everyone10+, Rating_Teen e Rating_Mature.
Riportiamo rapidamente di seguito il codice applicando il discorso sostenuto fino ad ora sulla classe Rating_Everyone10, invitando il lettore a notare l'effettiva analogia con la classe precedentemente trattata; per facilitare tale scopo verrà comunque messa in evidenza di seguito l'unica effettiva differenza.
Ovviamente, per salvare i risultati parziali vengono utilizzate delle variabili differenti.

```
df2 = df
y = df2['Rating_Everyone10'].values
```

Precedentemente avevamo passato come y la classe Rating_Everyone.

```
df2 = df2.drop(['Rating_Everyone'], axis=1)
df2 = df2.drop(['Rating_Everyone10'], axis=1)
df2 = df2.drop(['Rating_Teen'], axis=1)
df2 = df2.drop(['Rating_Mature'], axis=1)
X = df2.values
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.20)

plot_learning_curve(RandomForestClassifier(), "Rating_Everyone10+ Learning
Curves (Random Forest)", X, y, n_jobs=-1)
plot_learning_curve(KNeighborsClassifier(), "Rating_Everyone10+ Learning Curves
(k-NN)", X, y, n_jobs=-1)

log_reg = LogisticRegression(penalty='l1', dual=False, C=1.0,
fit_intercept=False, intercept_scaling=1,
                             class_weight=None, random_state=None,
solver='liblinear', max_iter=100,
                             multi_class='ovr',
                             verbose=0, warm_start=False, n_jobs=-1)
log_reg.fit(Xtrain, ytrain)
y_val_l = log_reg.predict(Xtest)
ris = accuracy_score(ytest, y_val_l)
mis = accuracy_score(ytest, y_val_l, normalize=False)
print("Logistic Regression Rating_Everyone10+ accuracy: ", ris)
print("Logistic Regression Rating_Everyone10+ misclassification: ", ytest.size
- mis)

radm = RandomForestClassifier(n_estimators=40, criterion='entropy',
max_depth=None, min_samples_split=2,
                              min_samples_leaf=1,
                              min_weight_fraction_leaf=0.0,
max_features='auto', max_leaf_nodes=None,
                              min_impurity_split=1e-07, bootstrap=True,
                              oob_score=True, n_jobs=-1, random_state=None,
verbose=0, warm_start=False,
                              class_weight=None)
radm.fit(Xtrain, ytrain)
y_val_l = radm.predict(Xtest)
ris = accuracy_score(ytest, y_val_l)
mis = accuracy_score(ytest, y_val_l, normalize=False)
print("Random Forest Rating_Everyone10+ accuracy: ", ris)
print("Random Forest Rating_Everyone10+ misclassification: ", ytest.size - mis)
knn = KNeighborsClassifier(n_neighbors=10, weights='uniform', algorithm='auto',
```

```
leaf_size=30, p=1, metric='minkowski', metric_params=None, n_jobs=1)
knn.fit(Xtrain, ytrain)
y_val_l = knn.predict(Xtest)
ris = accuracy_score(ytest, y_val_l)
mis = accuracy_score(ytest, y_val_l, normalize=False)
print("K-Nearest Neighbors Rating_Everyone10+ accuracy: ", ris)
print("K-Nearest Neighbors Rating_Everyone10+ misclassification: ", ytest.size
- mis)
print("\n")
```

10. Funzione `animate`

```
def animate(i, stop_event):
    for c in itertools.cycle(['|', '/', '-', '\\']):
        if stop_event.is_set():
            break
        sys.stdout.write('\r' + c + ' Sistemo il dataset... ' + c)
        sys.stdout.flush()
        time.sleep(0.1)
```

È una semplice funzione con l'unico scopo di mostrare una breve animazione durante la fase di elaborazione del dataset. All'interno del main, troviamo la chiamata a funzione effettuata da un thread nuovo:

```
t_stop = threading.Event()
t = threading.Thread(target=animate, args=(1, t_stop))
t.start()
```

Si hanno quindi due thread che operano in situazione di concorrenza all'interno del programma: uno si occupa di attivare il ciclo for all'interno di `animate` in attesa di essere interrotto, l'altro continua l'esecuzione del main ed interromperà il fratello proprio quando raggiungerà:

```
t_stop.set()
sys.stdout.write('\rFatto!')
time.sleep(1.5)
```

A questo punto il testo "Sistemo il dataset…" cambia in "Fatto!", il secondo thread muore e si procede con l'esecuzione del programma, ovvero con il capitolo 5 della relazione.

11. Funzione `plot_learning_curve`

```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, n_jobs=1,
    train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y,
    cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()
```

```
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    plt.legend(loc="best")
    return plt
```

È una funzione presente su Scikit-Learn[30]. Dopo aver letto la documentazione correlata (liberamente consultabile al corrispettivo indirizzo) siamo riusciti a padroneggiarla per poter tracciare le learning curves di Random Forest e k-NN per tutte le classi trattate all'interno del progetto.

12. Conclusioni

Arrivati a questo punto, è ovviamente necessario tirare le somme sulla bontà dei risultati ottenuti fino ad ora ed esprimere in chiaro il risultato finale del progetto.

Fino ad ora, è stato possibile notare l'effettiva bontà che hanno sul misclassification rate Logistic Regression e Random Forest, a dispetto di un k-NN che invece, nonostante l'applicazione della tecnica di rilevazione della migliore configurazione possibile Course-to-fine, non sia riuscito a brillare.

Adesso, è nostro desiderio ripercorrere rapidamente le tappe susseguite fino ad ora.

Abbiamo ricavato dal dataset la Design Matrix $\boldsymbol{X}, \boldsymbol{X} \in R^{N \times D}$ e $\boldsymbol{y}, \boldsymbol{y} \in R^{N}$ nonché vettore delle $y_i,\ y_i \in \{E, E10, T, M\}$:

$$\boldsymbol{X} = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nn} \end{bmatrix}$$

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$

È stato necessario, come discusso nel capitolo 5, effettuare una discretizzazione One-hot Encoding che ci ha portato ad una siffatta scomposizione del vettore $\boldsymbol{y}$:

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{w}_E + \boldsymbol{w}_{E10} + \boldsymbol{w}_T + \boldsymbol{w}_M \\ \boldsymbol{w}_E, \boldsymbol{w}_E &\in \{0,1\} \\ \boldsymbol{w}_{E10}, \boldsymbol{w}_{E10} &\in \{0,1\} \\ \boldsymbol{w}_T, \boldsymbol{w}_T &\in \{0,1\} \\ \boldsymbol{w}_M, \boldsymbol{w}_M &\in \{0,1\} \end{aligned}$$

A questo punto è stata applicata la metodologia One Versus the Rest (OVR), anche conosciuta come One Versus All (OVA), e contrapposta alla One Versus One (OVO), al fine di poterci riportare ad un problema di tipo binary class a dispetto del problema multi-class che realmente abbiamo.
È stato quindi applicato un classificatore binario su ogni vettore $\boldsymbol{w}_i, i = \{E, E10, T, M\}$.

Questo procedimento, descritto nel capitoli 7, 8 e 9, ci ha portato alle funzioni dei classificatori parziali

$$f_E, f_{E10}, f_T, f_M$$

Negli ultimi tre capitoli citati precedentemente abbiamo la funzione `predict` di Scikit-Learn, ottenendo i vettori $\widetilde{\boldsymbol{y}}_E, \widetilde{\boldsymbol{y}}_{E10}, \widetilde{\boldsymbol{y}}_T, \widetilde{\boldsymbol{y}}_M$. Nel dettaglio:

$$\widetilde{\boldsymbol{y}}_i, i = \{E, E10, T, M\},$$
$$\widetilde{\boldsymbol{y}}_i = \tilde{y}_1, \dots, \tilde{y}_n,$$
$$\tilde{y}_1, \dots, \tilde{y}_n \in \{0,1\}$$

che rappresentano dunque la $\boldsymbol{y}$ prevista dal classificatore. Applicando `accuracy_score` abbiamo calcolato la percentuale di accuratezza del modello utilizzato e il misclassification rate.

Adesso però vogliamo normalizzare le probabilità che ogni elemento appartenga ad una certa classe e, tenendo conto del fattore di confidenza che si ha per ogni classificatore, assegnare poi l'elemento alla classe.

Applicando quindi la fuzione `predict_proba` si ottengono i vettori:

$$\widehat{\boldsymbol{y}}_i, i = \{E, E10, T, M\},$$
$$\widehat{\boldsymbol{y}}_i = \hat{y}_1, \dots, \hat{y}_n,$$
$$\hat{y}_1, \dots, \hat{y}_n \in [0,1]$$

che rappresentano la probabilità per ogni elemento di appartenere ad una certa classe.
Ora vogliamo normalizzare tutte le probabilità portandole tutte sullo stesso piano $\left(\sum_i \overline{\boldsymbol{y}}_i = 1\right)$:

$$\overline{\boldsymbol{y}}_i, i = \{E, E10, T, M\},$$
$$\overline{\boldsymbol{y}}_i = \frac{\widehat{\boldsymbol{y}}_i}{\widehat{\boldsymbol{y}}_E + \widehat{\boldsymbol{y}}_{E10} + \widehat{\boldsymbol{y}}_T + \widehat{\boldsymbol{y}}_M}$$

Infine, applichiamo la majority rule [31]

$$Se\ \overline{\boldsymbol{y}}_i > 0.6\ \Rightarrow\ \overline{\boldsymbol{y}}_i \in i, i = \{E, E10, T, M\}.$$

Quanto descritto fino ad ora è stato ovviamente tradotto in Python ed applicato nel codice; verrà riportato nel seguito la parte concernente Random Forest, ma il l'applicazione a Logistic Regression e k-NN è assolutamente analoga.

```
# --- Random Forest ---

yB1 = radm1.predict_proba(X)
yB2 = radm2.predict_proba(X)
yB3 = radm3.predict_proba(X)
yB4 = radm4.predict_proba(X)
```

```
y_pred_final_rf = np.empty(len(df), dtype=str)
i = 0
for elem in y_pred_final_rf:
    a = yB1[i][1] / (yB1[i][1] + yB2[i][1] + yB3[i][1] + yB4[i][1])
    b = yB2[i][1] / (yB1[i][1] + yB2[i][1] + yB3[i][1] + yB4[i][1])
    c = yB3[i][1] / (yB1[i][1] + yB2[i][1] + yB3[i][1] + yB4[i][1])
    d = yB4[i][1] / (yB1[i][1] + yB2[i][1] + yB3[i][1] + yB4[i][1])
    m = max(a, b, c, d)
    if m == a:
        y_pred_final_rf[i] = 'E'
    elif m == b:
        y_pred_final_rf[i] = 'E10+'
    elif m == c:
        y_pred_final_rf[i] = 'T'
    else:
        y_pred_final_rf[i] = 'M'
    i += 1

print("Random Forest: %0.2f" % (accuracy_score(y_true, y_pred_final_rf)))

yA1 = log_reg1.predict_proba(X)
yA2 = log_reg2.predict_proba(X)
yA3 = log_reg3.predict_proba(X)
yA4 = log_reg4.predict_proba(X)

y_pred_final_log = np.empty(len(df), dtype=str)
i = 0
for elem in y_pred_final_log:
    a = yA1[i][1] / (yA1[i][1] + yA2[i][1] + yA3[i][1] + yA4[i][1])
    b = yA2[i][1] / (yA1[i][1] + yA2[i][1] + yA3[i][1] + yA4[i][1])
    c = yA3[i][1] / (yA1[i][1] + yA2[i][1] + yA3[i][1] + yA4[i][1])
    d = yA4[i][1] / (yA1[i][1] + yA2[i][1] + yA3[i][1] + yA4[i][1])
    m = max(a, b, c, d)
    if m == a:
        y_pred_final_log[i] = 'E'
    elif m == b:
        y_pred_final_log[i] = 'E10+'
    elif m == c:
        y_pred_final_log[i] = 'T'
    else:
        y_pred_final_log[i] = 'M'
    i += 1

print("Logistic Regression: %0.2f" % (accuracy_score(y_true,
y_pred_final_log)))

yC1 = knn1.predict_proba(X)
yC2 = knn2.predict_proba(X)
yC3 = knn3.predict_proba(X)
yC4 = knn4.predict_proba(X)

y_pred_final_knn = np.empty(len(df), dtype=str)
i = 0
for elem in y_pred_final_knn:
    a = yC1[i][1] / (yC1[i][1] + yC2[i][1] + yC3[i][1] + yC4[i][1])
    b = yC2[i][1] / (yC1[i][1] + yC2[i][1] + yC3[i][1] + yC4[i][1])
    c = yC3[i][1] / (yC1[i][1] + yC2[i][1] + yC3[i][1] + yC4[i][1])
    d = yC4[i][1] / (yC1[i][1] + yC2[i][1] + yC3[i][1] + yC4[i][1])
    m = max(a, b, c, d)
```

```
    if m == a:
        y_pred_final_knn[i] = 'E'
    elif m == b:
        y_pred_final_knn[i] = 'E10+'
    elif m == c:
        y_pred_final_knn[i] = 'T'
    else:
        y_pred_final_knn[i] = 'M'
    i += 1

print("k-NN: %0.2f" % (accuracy_score(y_true, y_pred_final_knn)))
```

Qual è il verdetto finale?
Riportiamo di seguito i valori finali dei modelli applicati tenendo conto della confidenza.
Come vedremo nella porzione copiata qui di seguito dal terminale (e ovviamente ricalcolabile avviando semplicemente il programma ed attendendo la sua esecuzione), nonostante i valori molto simili di Logistic Regression e Random Forest, in realtà nella nostra situazione quest'ultimo è stato decretato il classificatore più robusto ed efficace, con un risultato che ci ha lasciato pienamente soddisfatti.

```
Random Forest: 94.84%
Logistic Regression: 63.79%
k-NN: 55.35%
```

Bibliografia

Fonti nel documento:

[1] *Given, Lisa M. (2008). The Sage encyclopedia of qualitative research methods. Los Angeles, Calif.: Sage Publications. ISBN 1-4129-4163-6.*

[2] Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar (2012) Foundations of Machine Learning, The MIT Press ISBN 9780262018258.

[3] https://www.esrb.org

[4] http://pandas.pydata.org

[5] http://www.numpy.org

[6] https://www.scipy.org

[7] http://matplotlib.org

[8] http://scikit-learn.org/

[9] http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

[10] https://www.quora.com/What-are-good-ways-to-handle-discrete-and-continuous-inputs-together

[11] Bishop, Christopher (2006). Pattern recognition and machine learning. Berlin: Springer.

[12] Fayyad, Usama M.; Irani, Keki B. (1993) *"Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning". hdl:2014/35171. , Proceedings of the International Joint Conference on Uncertainty in AI* (Q334 .I571 1993), pp. 1022-1027

[13] https://www.quora.com/Why-is-Nearest-Neighbor-a-Lazy-Algorithm

[14] *Fulton, Mendez, Bastian, Musal (2012). "Confusion Between Odds and Probability, a Pandemic?" (PDF). Journal of Statistics Education. Retrieved 11 July 2014.*

[15] http://goldbook.iupac.org/html/L/L03613.html

[16] https://www.robots.ox.ac.uk/~vgg/publications/2011/Pedersoli11/pedersoli11.pdf

[17] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[18] https://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf

[19] https://www.quora.com/What-is-regularization-in-machine-learning

[20] http://www.theanalysisfactor.com/interpreting-the-intercept-in-a-regression-model/

[21] http://scikit-learn.org/stable/tutorial/statistical_inference/supervised_learning.html

[22] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2008). *The Elements of Statistical Learning* (2nd ed.). Springer. pp. 587-588.

[23] Gareth James; Daniela Witten; Trevor Hastie; Robert Tibshirani (2013). *An Introduction to Statistical Learning*. Springer. pp. 316–321.

[24] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[25] http://stats.stackexchange.com/questions/152807/which-samples-are-used-in-random forests-for-calculating-variable-importance

[26] http://machinelearning202.pbworks.com/w/file/fetch/60606349/breiman_randomforests.pdf

[27] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[28] https://arxiv.org/abs/1511.00628

[29] http://www.alglib.net/other/nearestneighbors.php

[30] http://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html

[31] https://en.wikipedia.org/wiki/Majority_rule