Relazione caso di studio progetto di

ingegneria della conoscenza

PRESENTATO DA:

SIMONE DE GIROLAMO – MAT. 717450
VITO LUCE - MAT. 698859

https://github.com/lucevito/Ingegneria-della-Conoscenza

SOMMARIO

INTRODUZIONE

Il progetto consiste in un sistema di suggerimenti alla visione di anime utilizzando diverse tecniche di *Content based Filtering* e di *Collaborative filtering*. Il programma permette di avere raccomandazioni in base al nome di un anime o ai generi preferiti, assieme ad un sistema che determina se lo show è stato un successo o no sfruttando diversi modelli di apprendimento supervisionato e da uno non supervisionato. Inoltre, il programma esegue altre funzioni utili come il trovare alcuni dettagli di un determinato anime o di sapere se un determinato anime appartiene ad un determinato genere.

DATASET

Durante la scelta del dataset da usare, il gruppo si è soffermato su questo dataset (https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database), in quanto, oltre ad essere un dataset contente un gran numero di dati, conteneva il maggior numero di dati validi al suo interno. Questo non ha richiesto l'aggiunta di altre informazioni da dataset esterni, mantenendo quindi l'omogeneità dei dati.

Il dataset comprende due file, denominati rispettivamente Anime e Rating.

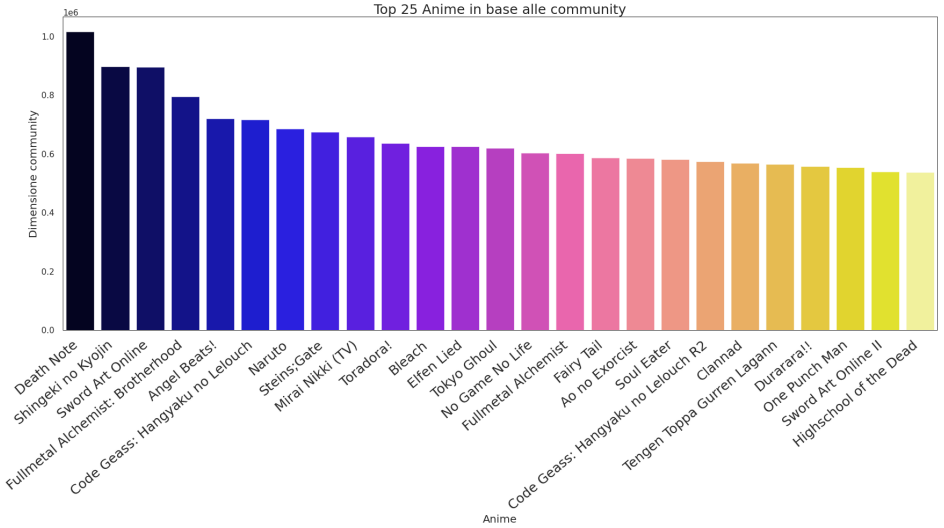
Il file Anime.csv contiene i campi

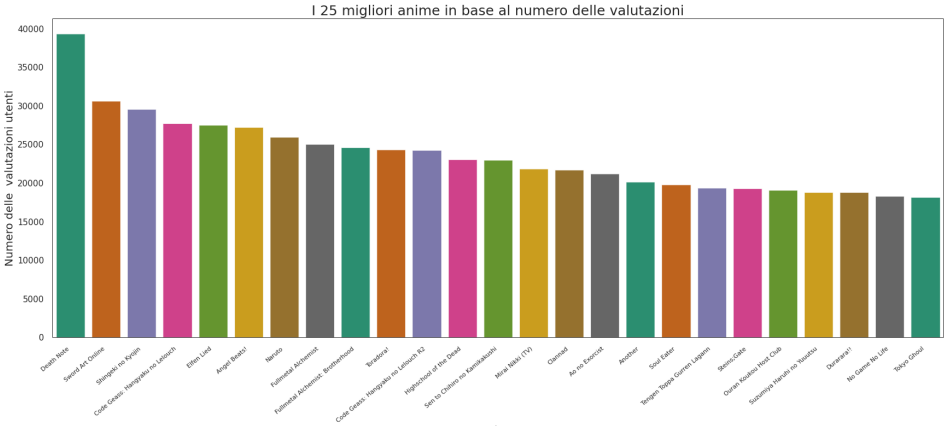
- **anime_id**: id univoco dell'anime (gli id sono estratti dal sito myanimelist.net);
- **name**: il nome completo dell'anime;
- **genre**: lista di generi separati da una virgola;
- **type**: film, serie TV, puntate speciali (OVA), etc…;
- **episodes**: quanti episodi contiene (1 se è un film);
- **rating**: punteggio medio da 1 a 10;
- **members**: numero dei membri della community che si trovano in un "gruppo" in cui si parla di quell'anime;

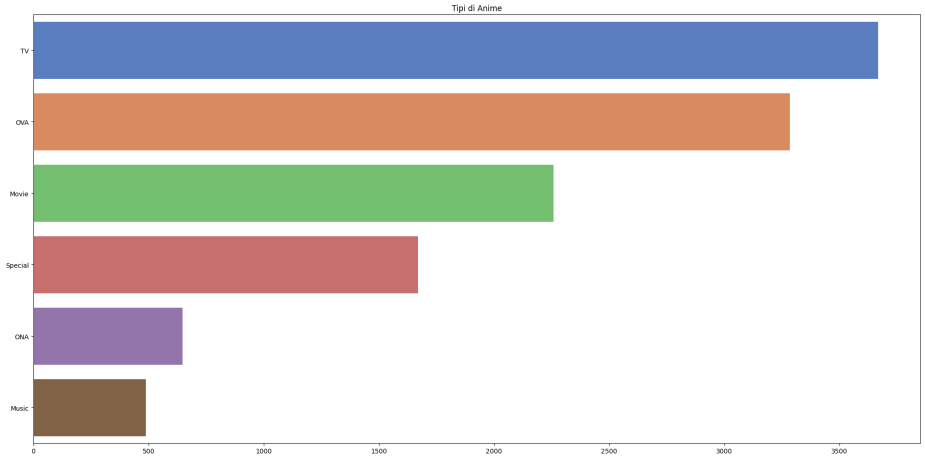
Il dataset Rating.csv contiene i campi

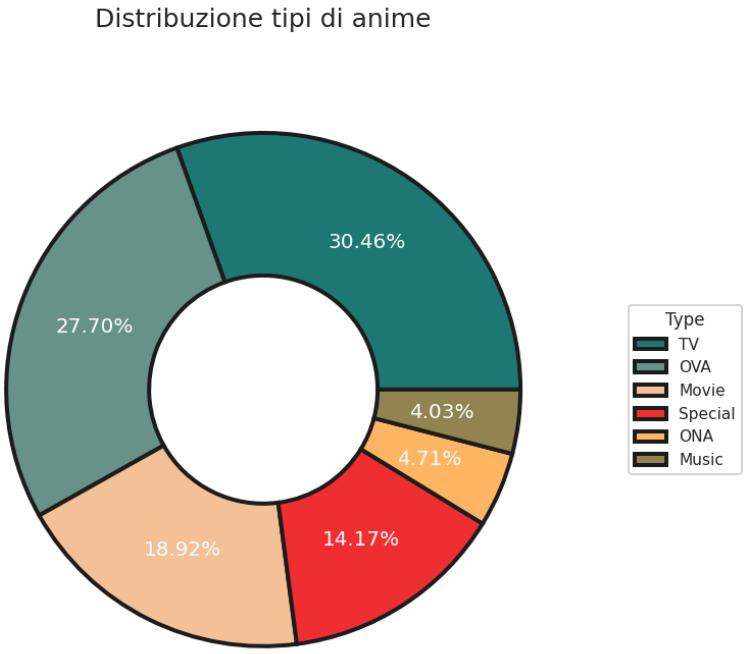
- **user_id:** identificativo casuale per un utente;
- **anime_id:** l'anime che l'utente ha votato;
- **rating:** punteggio da 1 a 10 che l'utente ha assegnato all'anime (-1 se l'utente lo ha visto ma non ha assegnato un punteggio);

INFORMAZIONI GRAFICHE









PRE-PROCESSING

Prima di utilizzare effettivamente i dati, sono state effettuate alcune operazioni di pre-processing dei dati per operare sui dataset, ossia:

- pulizia del testo dalle impurità (i.e. testo contente parole come “&quot”, “.hack//”, “&#039”, etc..)

```
anime_data['name'] = anime_data['name'].apply(text_cleaning)
```

- estrazione dei generi

```
anime_data['genre'].str.split(',').astype(str)
```

- generazione di un indice di anime

```
pd.Series(anime_data.index, index=anime_data['name']).drop_duplicates()
```

- riformattazione delle features (generi)

```
anime_data['genre'] = anime_data['genre'].fillna('')
    anime_data['genre'] = anime_data['genre'].astype('str')
    anime_data['genre'] = anime_data['genre'].str.split(', ')
    genre_columns_temp=anime_data.genre.apply(pd.Series).stack().str.get_dummies().
sum(level=0)
    anime_data = anime_data.drop(['genre'], axis=1)
    anime_data = pd.concat([anime_data,genre_columns_temp],axis=1)
```

- Generazione di una tabella pivot partendo da un anime e la lista dei voti dati dagli utenti
- Riduzione della tabella pivot in base ad un numero superiore di 500 voti per anime

```
    anime_dataclone = anime_data[['anime_id', 'name']].copy()
    anime_fulldata=pd.merge(anime_dataclone,rating_data,on='anime_id')
    del anime_dataclone
    del rating_data
    anime_fulldata["rating"].replace({-1: np.nan}, inplace=True)
    anime_fulldata = anime_fulldata.dropna(axis = 0, how ='any')
    counts = anime_fulldata['user_id'].value_counts()
    anime_fulldata = anime_fulldata[anime_fulldata['user_id'].isin(counts[counts >=
500].index)]
    anime_pivot=anime_fulldata.pivot_table(index='name',columns='user_id',values='r
ating').fillna(0)
    del anime_fulldata
    return anime_pivot
```

Tramite queste operazioni preliminari si va ad alleggerire il carico durante le successive operazioni, dato che l'indice degli anime funziona in base al numero di voti inserito per ognuno di essi, e siccome alcuni hanno pochissimi voti o non ne hanno alcuno (con valore -1), diventa possibile alleggerire il carico sulla memoria.

Inoltre, viene aggiunto un nuovo campo (**success**) estratto dal rating dell'anime, e viene determinato con un valore binario con 1 se ha un rating maggiore di 6, altrimenti 0 se ha un valore minore di 7. Il motivo per cui il gruppo ha deciso di usare valori binari rispetto a valori booleani è perché questo avrebbe comportato un ulteriore passo di pre-processing nel momento in cui era necessario creare una base di conoscenza per tutti quei metodi di apprendimento supervisionato che non accettano valori booleani ma solo valori numerici.

COLLABORATIVE FILTERING

Con il *Collaborative Filtering* si intende una classe di strumenti e meccanismi che consentono il recupero di informazioni predittive relativamente agli interessi di un insieme dato di utenti a partire da una massa ampia e tuttavia indifferenziata di conoscenza. In questo caso, il gruppo ha utilizzato la fattorizzazione di matrice (presente nell'algoritmo NN search) che decompongono la matrice di interazione user-item nel prodotto di due matrici di dimensionalità inferiore.

NEAREST NEIGHBOR SEARCH

È una forma di ricerca di prossimità, dove lo scopo è l'ottimizzazione per trova il punto in un dato insieme più vicino a un dato punto. La vicinanza è tipicamente espressa in termini di una funzione di dissomiglianza: meno simili sono gli oggetti, maggiori sono i valori della funzione. L'implementazione utilizzata è **Brute-force**, che prevede il calcolo a forza bruta delle distanze tra tutte le coppie di punti nel set di dati: questo perché dati N campioni nelle D dimensioni, l' approccio ha una complessità $O[DN^2]$. Questa implementazione è molto veloce per dataset piccoli come questo, ma al crescere dei dati, diventa molto più lungo e quasi impossibile da utilizzare a motivo del tempo impiegato.

Questo algoritmo viene utilizzato tramite distanza del coseno per trovare i diversi indici delle anime che sono più vicini all'anime che abbiamo inserito in input. Questo modello prende in considerazione solo gli anime con più di 500 voti, per evitare di sovraccaricare troppo la macchina.

```
    anime_matrix = csr_matrix(anime_pivot.values)
    # using the k-nearest neighbors algorithm
    model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')
    model_knn.fit(anime_matrix)
    distances, indices = model_knn.kneighbors(anime_pivot.loc
                        [anime_pivot.index == title].values.reshape(1, -1),
n_neighbors = 11)
```

FIGURA 1: PARTE DI CODICE ALGORITMO NEAREST NEGHBOR SEARCH CON SEGUENTE CASO D'USO

```
Do you want proceed with a name test : Shingeki no Kyojin a.k.a. Attack of titan ?
Enter y for a yes, something else for no : y

Using Collaborative Filtering, the Top 10 most similar anime are :
Warning : to reduce the use of ram,
the size of the database will be reduced, anime with at least 500 votes will be considered

0: Shingeki no Kyojin, with distance of 0.0:
1: Sword Art Online, with distance of 0.1335946872659557:
2: Angel Beats!, with distance of 0.14903678429608036:
3: No Game No Life, with distance of 0.16254528367400767:
4: Hatarаku Maou-sama!, with distance of 0.1638429001306072:
5: Steins;Gate, with distance of 0.16728424722076785:
6: Highschool of the Dead, with distance of 0.16957425602499732:
7: Toradora!, with distance of 0.1752054636541328:
8: Psycho-Pass, with distance of 0.17726396458904436:
9: Death Note, with distance of 0.18132604001891173:
10: Code Geass: Hangyaku no Lelouch, with distance of 0.18198153072037426:
Time to calculate it : 0.51 seconds
```

CONTENT BASED FILTERING

Il programma sfrutta il database per creare un sistema di raccomandazioni personalizzate per l'utente così da aiutarlo nelle sue scelte, partendo dagli elementi a cui reagisce positivamente. Mettendo a confronto il contenuto degli elementi e il profilo delle preferenze, il sistema di *content based filtering* suggerisce all'utente articoli che possono essere di suo interesse. L'approccio basato sul contenuto utilizza sistemi per creare delle stime probabili e affidabili.

Il gruppo, per scopi didattici, ha deciso di utilizzare algoritmi per le raccomandazioni performanti per lo scopo e non, al fine di capire se risultassero più precisi rispetto a quelli più performanti. Inoltre, per ogni algoritmo usato, è stata inserita una stima del tempo per capire quanto tempo impiega ogni algoritmo per restituire i risultati. L'obiettivo è il confronto delle tecniche usate nel content based filtering, considerando il tempo richiesto per l'elaborazione dei dati. Tale test sfortunatamente risulta non valido al 100%, in quanto richiede un uso superiore della ram a disposizione della macchina con cui tale programma è stato sviluppato. Per tale motivo il database o la tecnica usata in alcuni test sono stati limitati ma consentono comunque di ottenere un'idea della differenza di complessità richiesta fra le varie tecniche ed il loro relativo tempo per l'elaborazione.

Uno dei metodi utilizzati è il **tf-idf**, ovvero il più performante, ed il metodo prende in input il nome di un anime. Gli altri metodi invece prendono un elenco di generi su cui effettuare la raccomandazione. Ai fini di test, il programma suggerisce un anime famoso su cui utilizzare i diversi metodi ("attack of titan" a.k.a. "shikeshi no kioshi").

TDF-IDF

È una funzione utilizzata per misurare l'importanza di un termine rispetto ad un documento o ad una collezione di documenti. Tale funzione aumenta proporzionalmente al numero di volte che il termine è contenuto nel documento, ma cresce in maniera inversamente proporzionale con la frequenza del termine nella collezione. L'idea alla base di questo comportamento è di dare più importanza ai

termini che compaiono nel documento, ma che in generale sono poco frequenti. Nella versione implementata all'interno del codice (`TfIdfVectorizer`) della libreria scikit, permette di convertire una collezione di documenti non trattati in una matrice con caratteristiche Tf-Idf, ed usa la similarità di coseno (in questo caso, calcolata partenendo dalla matrice con la frequenza di nomi) per mostrare i 10 anime più vicini a quello inserito in input.

```
    tf = TfidfVectorizer(analyzer='word')
    tf_matrix = tf.fit_transform(genres_list)
    cosine_sim = cosine_similarity(tf_matrix, tf_matrix)

    idx = anime_index[title]
    sim_scores = list(enumerate(cosine_sim[idx]))

    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
sim_scores = sim_scores[1:11]
    # anime indices
    anime_indices = [i[0] for i in sim_scores]
    # print the top 10 most similar anime
    print('\nUsing Content based filtering, the Top 10 most similar anime are :\n')
    print(pd.DataFrame({'Anime name': anime_data['name'].iloc[anime_indices].val-
ues,
                                      'Rating': anime_data['rating'].iloc[anime_indi-
ces].values}))
    print('Time to calculate it : {:0.2f} seconds'.format(time.time()-time_sec))
    print('\n')
```

FIGURA 2: PARTE DI CODICE ALGORITMO TF-IDF CON SEGUENTE CASO D'USO

```
Do you want proceed with a name test : Shingeki no Kyojin a.k.a. Attack of titan ?
Enter y for a yes, something else for no : y

Using Content based filtering, the Top 10 most similar anime are :

                                     Anime name  Rating
0                          Shingeki no Kyojin OVA    7.88
1       Shingeki no Kyojin Movie 2: Jiyuu no Tsubasa    7.75
2        Shingeki no Kyojin Movie 1: Guren no Yumiya    7.70
3                 Super Robot Taisen OG: Divine Wars    6.96
4                           Oni Chichi: Refresh♥    7.25
5              Saint Seiya: Meiou Hades Elysion-hen    7.82
6                                     One Piece    8.58
7  One Piece: Episode of Merry - Mou Hitori no Na...    8.29
8  One Piece: Episode of Nami - Koukaishi no Nami...    8.27
9  One Piece: Episode of Sabo - 3 Kyoudai no Kizu...    7.78
Time to calculate it : 3.44 seconds
```

RANDOM FOREST

Ogni albero in una *random forest* impara da un campione casuale di dati. I campioni vengono disegnati con la sostituzione, nota come *bootstrap*, il che significa che alcuni campioni verranno utilizzati più volte in un singolo albero.

L'idea è che addestrando ciascun albero su campioni diversi, sebbene ogni albero possa presentare una varianza elevata rispetto a una particolare serie di dati di addestramento, nel complesso l'intera foresta avrà una varianza inferiore, in modo da avere le predizioni finali vicine al risultato.

I vantaggi del Random Forest sono la versatilità, perché può essere utilizzato sia per problemi di regressione che di classificazione, funzionando bene con una combinazione di caratteristiche numeriche e categoriche, e l'avere gli iperparametri predefiniti ottimali, perché producono un buon

risultato di previsione, consentendo anche un notevole miglioramento di essi, e di conseguenza della previsione.

In generale, questi algoritmi sono veloci da addestrare, ma piuttosto lenti nel creare previsioni una volta che sono stati addestrati, poiché richiedono un alto numero di alberi per ottenere risultati accurati.

```
 time_sec =time.time()
    clf = RandomForestClassifier(n_jobs=10, random_state=42, max_depth=10)
    clf.fit(anime_select[features], y)
    print('Random Forest Classifier recommendation: ')
    print('Warning : the maximum depth of the tree is limited to 10 in this
particular test\n')
    print(pd.DataFrame({'Anime name' :
anime_select['name'].loc[anime_select['anime_id'] ==
clf.predict(test[features])[0]]}))
    print('Time to calculate it : {:0.2f} seconds'.format(time.time()-time_sec))
    print('\n')
    del clf
```

FIGURA 3: PARTE DI CODICE ALGORITMO RANDOM FOREST

SUPPORT VECTOR

L'algoritmo SVM può essere impiegato per problemi di classificazione multiclasse, utilizzando la metodologia **one-vs-one**.

Nello specifico, si creano $\frac{k(k-1)}{2}$ classificatori, dove k è il numero di classi che effettuano classificazione su coppie di classi, per poi assegnare come classe finale quella con più assegnazioni.

Nell'apprendimento automatico, le macchine vettoriali di supporto sono modelli di apprendimento supervisionato con algoritmi di apprendimento associati che analizzano i dati per la classificazione e l'analisi di regressione. L'SVM è basato sull'idea di trovare un iperpiano che divida al meglio un set di dati in due o più classi su x dimensioni spaziali dove x è il numero di classi. I punti dati più vicini all'iperpiano sono detti vettori di supporto e sono i vettori rappresentativi delle possibili classi di appartenenza.

Dato un insieme di esempi di addestramento, ciascuno contrassegnato come appartenente a una delle due categorie, un algoritmo di addestramento SVM costruisce un modello che assegna nuovi esempi a una categoria o all'altra. SVM mappa gli esempi di addestramento in punti nello spazio in modo da massimizzare l'ampiezza del divario tra le due categorie. I nuovi esempi vengono quindi mappati in quello stesso spazio e si prevede che appartengano a una categoria in base a quale lato del divario cadono.

Oltre a eseguire la classificazione lineare, le SVM possono eseguire in modo efficiente una classificazione non lineare utilizzando quello che viene chiamato il **trucco del kernel**, mappando implicitamente i loro input in spazi di funzionalità ad alta dimensione.

Nel nostro caso l'implementazione utilizzata usa il metodo del kernel tramite **rbf**, dove il primo parametro con valore base rende più uniforme l'iperpiano, mentre un valore più alto spinge a classificare meglio gli esempi; il secondo parametro viene usato per indicare il peso che ciascun elemento ha nell'insieme di training. Questo metodo permette maggiore versatilità, ma allo stesso

tempo, essendo un metodo non probabilistico, non esiste un'interpretazione probabilistica diretta per l'appartenenza al gruppo.

```
    time_sec =time.time()
    svc = SVC()
    print('Processing data...')
    svc.fit(anime_select[features], y)
    print('Support Vector Classifier recommendation: \n')
    print(pd.DataFrame({'Anime name' :
anime_select['name'].loc[anime_select['anime_id'] ==
svc.predict(test[features])[0]]}))
    print('Time to calculate it : {:0.2f} seconds'.format(time.time()-time_sec))
    print('\n')
    del svc
```

FIGURA 4: PARTE DI CODICE ALGORITMO SUPPORT VECTOR

BAGGING CLASSIFIER

Un classificatore Bagging è un **meta stimatore** d'insieme che adatta i classificatori di base ciascuno su sottoinsiemi casuali del set di dati originale e quindi aggrega le loro previsioni individuali (votando o calcolando la media) per formare una previsione finale. Un tale meta-stimatore può essere tipicamente utilizzato come un modo per ridurre la varianza di uno stimatore a scatola nera (ad esempio, un albero decisionale), introducendo la randomizzazione nella sua procedura di costruzione e quindi facendone un insieme. Il bagging si basa quindi sull'addestrare più modelli dello stesso tipo, ciascuno su sottoinsiemi casuali del dataset originale e quindi aggrega le loro previsioni individuali (mediante voto o media) per formare una previsione finale.

Il gruppo ha deciso di utilizzare due stimatori diversi per questo classificatore:

- K-nearest neighbur classifier: La classificazione è calcolata da un voto a maggioranza semplice dei vicini più vicini di ciascun punto: a un punto di interrogazione viene assegnata la classe di dati che ha il maggior numero di rappresentanti all'interno dei vicini più vicini del punto)
- Decision tree classifier: L'obiettivo è creare un modello che preveda il valore di una variabile target apprendendo semplici regole decisionali dedotte dalle caratteristiche dei dati.

Il motivo principale per l'uso di due stimatori è di comprendere come si comportano con i dati e di quanto discostano tra di loro, dato che il decision tree richiede minore preparazione sui dati, ma allo stesso tempo può diventare molto complesso, non permettendo di generalizzare bene i dati di training, cosa che invece è meno marcata con il knn.

```
time_sec =time.time()
    model=BaggingClassifier(base_estimator=KNeighborsClassifier(n_neighbors=3),rand
om_state=0,n_estimators=700)
    print('Processing data...')
    model.fit(anime_select[features],y)
    print('Bagging classifier using k-nearest neighbors vote recommendation: \n')
    print(pd.DataFrame({'Anime name' :
anime_select['name'].loc[anime_select['anime_id'] ==
model.predict(test[features])[0]]}))
    print('Time to calculate it : {:0.2f} seconds'.format(time.time()-time_sec))
    print('\n')
    del model
```

FIGURA 5: PARTE DI CODICE ALGORITMO BANGGING CLASSIFIER - KN CLASSIFIER

```
    time_sec =time.time()
    model=BaggingClassifier(base_estimator=DecisionTreeClassifier(max_depth=10),ran
dom_state=0,n_estimators=100)
    print('Processing data...')
    model.fit(anime_select[features],y)
    print('Bagging classifier using Decision Tree Classifier recommendation: ')
    print('Warning : the maximum depth of the tree is limited to 10 in this
particular test\n')
    print(pd.DataFrame({'Anime name' :
anime_select['name'].loc[anime_select['anime_id'] ==
model.predict(test[features])[0]]}))
    print('Time to calculate it : {:0.2f} seconds'.format(time.time()-time_sec))
    print('\n')
    del model
```

FIGURA 6: PARTE DI CODICE ALGORITMO BAGGING CLASSIFIER - DECISION TREE CLASSIFIER

```
Warning : to reduce the use of ram, the database size will be reduced
based on the type of anime such as tv movies etc ..

Enter 1 for seach with type Movie
Enter 2 for seach with type Music
Enter 3 for seach with type ONA
Enter 4 for seach with type OVA
Enter 5 for seach with type Special
Enter 6 for seach with type TV
Enter your choice : 6
Select one or more genres
 1 Action
 2 Adventure
 3 Cars
 4 Comedy
 5 Dementia
 6 Demons
 7 Drama
 8 Ecchi
 9 Fantasy
 10 Game
 11 Harem
 12 Hentai
 13 Historical
 14 Horror
 15 Josei
 16 Kids
 17 Magic
 18 Martial Arts
 19 Mecha
 20 Military
```

CASO D'USO – RACCOMANDAZIONE ANIME

```
 21 Music
 22 Mystery
 23 Parody
 24 Police
 25 Psychological
 26 Romance
 27 Samurai
 28 School
 29 Sci-Fi
 30 Seinen
 31 Shoujo
 32 Shoujo Ai
 33 Shounen
 34 Shounen Ai
 35 Slice of Life
 36 Space
 37 Sports
 38 Super Power
 39 Supernatural
 40 Thriller
 41 Vampire
 42 Yaoi
 43 Yuri
 44 test with genres of Shingeki no Kyojin a.k.a. Attack of titan
Enter 0 to stop selection, enter 45 if you do not want this type of recommendation
Enter your choice : 44
Processing data...
Random Forest Classifier recommendation:
Warning : the maximum depth of the tree is limited to 10 in this particular test

           Anime name
86  Shingeki no Kyojin
Time to calculate it : 2.07 seconds


Processing data...
Support Vector Classifier recommendation:

           Anime name
86  Shingeki no Kyojin
Time to calculate it : 39.15 seconds


Processing data...
Bagging classifier using k-nearest neighbors vote recommendation:

           Anime name
86  Shingeki no Kyojin
Time to calculate it : 10.36 seconds


Processing data...
Bagging classifier using Decision Tree Classifier recommendation:
Warning : the maximum depth of the tree is limited to 10 in this particular test

     Anime name
2167    Tactics
Time to calculate it : 5.36 seconds
```

SUCCESS PREDICTION

Il sistema, oltre a fornire alcune raccomandazioni, è in grado di prevedere tramite l'uso di diversi algoritmi se un determinato anime inserito ha avuto successo. Allo scopo vengono utilizzati diversi algoritmi predittivi, tra cui:

- Sistemi di apprendimento supervisionato tramite l'uso di classificatori
- Sistemi di apprendimento non supervisionato tramite l'uso di operazioni di clustering

CLASSIFICATORI

Classificazione e predizione sono processi che consistono nel creare modelli che possono essere usati per descrivere degli insiemi di dati o per fare predizioni future.

Il processo di classificazione è divisibile in tre fasi tra le quali: addestramento, stima dell'accuratezza e utilizzo del modello.

La classificazione nel caso di studio è stata utilizzata con lo scopo di predire, tramite addestramento sul dataset precedentemente ottenuto, di conoscere il successo di una serie e di raccomandare un anime in base ad un genere preferito o a precedenti serie già viste.

Il dataset utilizzato è abbastanza complesso e ampio, pertanto è stata necessaria una prima fase dedita ad una ricerca accurata del classificatore più adatto, ossia quello in grado di gestire il dataset in questione. Infatti, basandosi sul teorema del "No free Lunch", si capisce che non tutti gli algoritmi possono andare bene per questo genere di task, motivo per cui sono stati usati diversi algoritmi per un singolo scopo. Questo non solo per comprenderne l'uso a livello accademico, ma per capire come questi si comportano con i dati e trovare l'alternativa più adatta in base al modello di predizione più accurato, in base alla velocità o alle performance del modello stesso.

Il metodo utilizzato per attuare questa ricerca è stato quello di considerare i classificatori le cui caratteristiche sembravano ottimali per il caso di studio, per poi valutarne le performance tramite *cross validation* ed in seguito effettuare un confronto tra tutti.

GAUSSIAN NAIVE CLASSIFIER

I classificatori "naive Bayes" sono una famiglia di semplici classificatori probabilistici basati sull'applicazione del teorema di Bayes con una forte assunzione (ingenua) di indipedenza tra le feature. Questo avviene con tutti i classificatori Naive Bayes, i quali presuppongono che il valore di una particolare caratteristica sia indipendente dal valore di qualsiasi altra feature, data la variabile di classe. Con un modello di eventi multinomiali, gli esempi (vettori di feature) rappresentano le frequenze con cui certi eventi sono stati generati da una distribuzione polinomiale $(p_1, \dots, p_n)$ dove p_i è la probabilità che l'evento i si verifichi. Il modello è espresso nella formula

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Dove i prametri σ_y e μ_y sono stimati usando la probabilità maggiore.

Il classificatore è usato in questo modo per prevedere, dopo aver effettuato una fase di training, per predire il successo di un titolo inserito all'interno del dataset.

```
        print('For Gaussian Naive Bayes Classifier : ')
        model = GaussianNB()
        model.fit(x_train,y_train)
        pred = model.predict(test)
        pred = np.array(pred).astype(int)
        if pred[0] == 1:
            print('Yes')
        else:
            print('No')
```

FIGURA 7: GAUSSIAN NAIVE CLASSIFIER – TEST SINGOLO

```
    print('For Gaussian Naive Bayes Classifier : ')
    model = GaussianNB()
    model.fit(x_train,y_train)
    pred = model.predict(x_test)
    print(metrics.classification_report(y_test,pred))
    print('')
```

FIGURA 8- GAUSSIAN NAIVE CLASSIFIER - TEST COMPLESSIVO

LOGISTIC REGRESSION

È un modello di regressione statisico usato negli algoritmi di classificazione ML (supervised) in cui vengono determinati i pesi di una funzione lineare appiattita dal sigmoide – che converte i valori reali in un valore compreso tra 0 ed 1 - minimizzando l'errore sull'insieme di esempi. Il modello viene utilizzato come uno delle diverse possibilità per determinare il successo di un titolo anime.

```
        print('For Logistic Regression : ')
        model = LogisticRegression()
        model.fit(x_train,y_train)
        pred = model.predict(test)
        pred = np.array(pred).astype(int)
        if pred[0] == 1:
            print('Yes')
        else:
            print('No')
```

FIGURA 9: LOGISTIC REGRESSION – TEST SINGOLO

```
    print('For Logistic Regression : ')
    model = LogisticRegression()
    model.fit(x_train,y_train)
    pred = model.predict(x_test)
    print(metrics.classification_report(y_test,pred))
    print('')
```

FIGURA 10: LOGISTIC REGRESSION – TEST COMPLESSIVO

K-NN CLASSIFIER

Il classificatore kNN è usato per classificare gli oggetti basandosi sulle caratteristiche di oggetti vicini a quello considerato. In questo classificatore un oggetto è ordinato da un voto di pluralità con l'oggetto assegnato alla classe più comune tra i suoi k vicini più vicini (k è un numero intero positivo, tipicamente piccolo). Il tutto viene implementato attraverso l'algoritmo k-nearest neighbors (k-NN) il cui input è costituito dagli esempi di addestramento, mentre l'output è il risultato in valore numerico della possibilità di successo del titolo.

```
        print('For KNeighbors Classifier : ')
        model = KNeighborsClassifier()
        model.fit(x_train,y_train)
        pred = model.predict(test)
        pred = np.array(pred).astype(int)
        if pred[0] == 1:
            print('Yes')
        else:
            print('No')
```

FIGURA 11-12: K-NN CLASSIFIER –TEST SINGOLO

```
    print('For KNeighbors Classifier : ')
    model = KNeighborsClassifier()
    model.fit(x_train,y_train)
    pred = model.predict(x_test)
    print(metrics.classification_report(y_test,pred))
    print('')
```

FIGURA 12: K-NN CLASSIFIER – TEST COMPLESSIVO

DECISION TREE CLASSIFIER

È un metodo molto semplice ed efficace per realizzare un classificatore e l'addestramento degli alberi di decisione è una delle tecniche attuali di maggior successo. Un albero di decisione è un albero di classificatori (Decision Stump) dove ogni nodo interno è associato ad una particolare "domanda" su una caratteristica (feature). Da questo nodo dipartono tanti archi quanti sono i possibili valori che la caratteristica può assumere, fino a raggiungere le foglie che indicano la categoria associata alla decisione. Esso viene costruito utilizzando tecniche di apprendimento a partire dall'insieme dei dati iniziali (data set), il quale viene diviso in due sottoinsiemi: il training set sulla base del quale si crea la struttura dell'albero e il test set che viene utilizzato per testare l'accuratezza del modello predittivo così creato.

```
print('For Decision Tree Classifier : ')
model = DecisionTreeClassifier()
model.fit(x_train,y_train)
pred = model.predict(test)
pred = np.array(pred).astype(int)
if pred[0] == 1:
    print('Yes')
else:
    print('No')
```

FIGURA 13: DECISION TREE CLASSIFIER – TEST SINGOLO

```
print('For Decision Tree Classifier : ')
model = DecisionTreeClassifier()
model.fit(x_train,y_train)
pred = model.predict(x_test)
print(metrics.classification_report(y_test,pred))
print('')
```

FIGURA 14: DECISION TREE CLASSIFIER – TEST COMPLESSIVO

MLP CLASSIFIER

È un classificatore che appartiene alle reti neuronali, che allena il modello tramite *back-propagation* ed agisce a più strati. Con questo modello viene ottimizzata la funzione di perdita logaritmica utilizzando l'algoritmo L-BFGS oppure la discesa del gradiente stocastico. Quest'ultimo è stato scelto dal gruppo in quanto è relativamente più veloce durate la fase di allenamento.

```
print('For Neural Network : ')
model = MLPClassifier()
model.fit(x_train,y_train)
pred = model.predict(test)
pred = np.array(pred).astype(int)
if pred[0] == 1:
    print('Yes')
else:
    print('No')
```

FIGURA 15: MLP CLASSIFIER – TEST SINGOLO

```
print('For Neural Network : ')
model = MLPClassifier()
model.fit(x_train,y_train)
pred = model.predict(x_test)
print(metrics.classification_report(y_test,pred))
print('')
```

FIGURA 16: MLP CLASSIFIER - TEST COMPLESSIVO

RANDOM FOREST CLASSIFIER

È un modello composito costituito da molti alberi di decisione, ognuno dei quali fornisce una predizione. Esse vengono poi combinate allo scopo di ottenere una previsione complessiva della foresta per un dato esempio. Le feature su cui vengono fatte le predizioni sono permutate a random.

```
        print('For Random Forest Classifier : ')
        model = RandomForestClassifier()
        model.fit(x_train,y_train)
        pred = model.predict(test)
        pred = np.array(pred).astype(int)
        if pred[0] == 1:
            print('Yes')
        else:
            print('No')
```

FIGURA 17: RANDOM FOREST CLASSIFIER – TEST SINGOLO

```
    print('For Random Forest Classifier : ')
    model = RandomForestClassifier()
    model.fit(x_train,y_train)
    pred = model.predict(x_test)
    print(metrics.classification_report(y_test,pred))
    print('')
```

FIGURA 18: RANDOM FOREST CLASSIFIER – TEST COMPLESSIVO

GRADIENT BOOSTING CLASSIFIER

È una tecnica di apprendimento automatico utilizzata, in questo caso, in attività classificazione supervisionata. Fornisce un modello di previsione sotto forma di un insieme di modelli di previsione deboli, che sono tipicamente alberi decisionali. Con questo algoritmo di classificazione viene restituita una predizione che viene migliorata ad ogni iterata, in modo tale da poter minimizzarne l'errore ed avere un risultato più preciso sulla possibilità di successo.

```
        print('For Gradient Boosting Classifier : ')
        model = GradientBoostingClassifier()
        model.fit(x_train,y_train)
        pred = model.predict(test)
        pred = np.array(pred).astype(int)
        if pred[0] == 1:
            print('Yes')
        else:
            print('No')
```

FIGURA 19: GRADIENT BOOSTING CLASSIFIER – TEST SINGOLO

```
    print('For Gradient Boosting Classifier : ')
    model = GradientBoostingClassifier()
    model.fit(x_train,y_train)
    pred = model.predict(x_test)
    print(metrics.classification_report(y_test,pred))
    print('')
```

FIGURA 20: GRADIENT BOOSTING CLASSIFIER – TEST COMPLESSIVO

CLUSTERING

Il clustering è un insieme di tecniche di analisi dei dati volte alla selezione e raggruppamento di elementi omogenei in un insieme di dati. Le tecniche di clustering si basano su misure relative alla somiglianza tra gli elementi. In molti approcci questa similarità (o meglio, dissimilarità) è concepita in termini di distanza in uno spazio multidimensionale ed i suoi algoritmi sviluppati su questo concept raggruppano gli elementi sulla base della loro distanza reciproca. Di conseguenza, l'appartenenza o meno a un insieme dipende da quanto l'elemento preso in esame è distante dall'insieme stesso.

All'interno del programma un algoritmo che utilizza una forma di clustering detta **clustering partizionale** è il K-Means

KMEANS

L'algoritmo K-means è un algoritmo apprendimento non supervisionato di hard-clustering partizionale che permette di suddividere un insieme di oggetti in K gruppi sulla base dei loro attributi. Si assume che gli attributi degli oggetti possano essere rappresentati come vettori, e che quindi formino uno spazio vettoriale. Ogni cluster viene identificato mediante un centroide in cui i dati vengono divisi in gruppi aventi caratteristiche simili. L'algoritmo segue una procedura iterativa. Inizialmente crea K partizioni e assegna ad ogni partizione i punti d'ingresso o casualmente o usando alcune informazioni euristiche. Quindi calcola il centroide di ogni gruppo. Costruisce quindi una nuova partizione associando ogni punto d'ingresso al cluster il cui centroide è più vicino ad esso. Quindi vengono ricalcolati i centroidi per i nuovi cluster e così via, finché l'algoritmo non converge. Questo algoritmo mira a scegliere i centroidi che minimizzano **l'inerzia**, o il criterio della **somma dei quadrati all'interno del cluster**, che si esprime come:

$$\sum_{i=0}^{n} \min_{\mu_j \in C} \left(\left|\left| x_i - \mu_j \right|\right|^2 \right)$$

Con inerzia si intende una misura della coerenza interna dei diversi cluster.

```
test = anime_data.loc[anime_data['name'] == animename]
        train = anime_data.drop(['name', 'type',
'episodes','rating','members','success'], axis=1)
        test = test.drop(['name', 'type', 'episodes','rating','members','success'],
axis=1)

        model = KMeans(n_clusters=1000, random_state=0).fit(train)
        pred = model.predict(test)

        print('Similar anime using unsupervised learning : ')
        for i in range(len(anime_data)):
            if model.labels_[i] == pred[0]:
                print(anime_data['name'].iloc[i])
        print('')
    else:
        print('There is no anime with that name in the dataset\n')
```

```
Enter anime name as example Gintama : Naruto
For Gaussian Naive Bayes Classifier :
Yes
For Logistic Regression :
Yes
For KNeighbors Classifier :
Yes
For Decision Tree Classifier :
Yes
For Neural Network :
No
For Random Forest Classifier :
Yes
For Gradient Boosting Classifier :
Yes
```

CASO D'USO – SUCCESSO DELL'ANIME

```
1 mean Success
0 means Flop

For Gaussian Naive Bayes Classifier :
              precision    recall  f1-score   support

           0       0.81      0.85      0.83      4015
           1       0.67      0.61      0.64      1994

    accuracy                           0.77      6009
   macro avg       0.74      0.73      0.73      6009
weighted avg       0.76      0.77      0.77      6009


For Logistic Regression :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4015
           1       0.99      0.99      0.99      1994

    accuracy                           0.99      6009
   macro avg       0.99      0.99      0.99      6009
weighted avg       0.99      0.99      0.99      6009


For KNeighbors Classifier :
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      4015
           1       0.78      0.64      0.70      1994

    accuracy                           0.82      6009
   macro avg       0.81      0.77      0.79      6009
weighted avg       0.82      0.82      0.81      6009


For Decision Tree Classifier :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4015
           1       1.00      1.00      1.00      1994

    accuracy                           1.00      6009
   macro avg       1.00      1.00      1.00      6009
weighted avg       1.00      1.00      1.00      6009
```

```
For Decision Tree Classifier :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4015
           1       1.00      1.00      1.00      1994

    accuracy                           1.00      6009
   macro avg       1.00      1.00      1.00      6009
weighted avg       1.00      1.00      1.00      6009


For Neural Network :
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      4015
           1       0.97      0.99      0.98      1994

    accuracy                           0.99      6009
   macro avg       0.98      0.99      0.98      6009
weighted avg       0.99      0.99      0.99      6009


For Random Forest Classifier :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4015
           1       1.00      1.00      1.00      1994

    accuracy                           1.00      6009
   macro avg       1.00      1.00      1.00      6009
weighted avg       1.00      1.00      1.00      6009


For Gradient Boosting Classifier :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4015
           1       1.00      1.00      1.00      1994

    accuracy                           1.00      6009
   macro avg       1.00      1.00      1.00      6009
weighted avg       1.00      1.00      1.00      6009
```

CASO D'USO – SUCCESSO DELL'ANIME CON MODELLO DI TEST

KNOWLEDGE BASE

Una base di conoscenza è una banca dati, grazie alle cui informazioni e, quindi, alle conoscenze che sono presenti al suo interno, riesce a fornire un supporto all'utente fornendogli risposte a delle domande che vengono effettuate senza la necessità di generare i possibili mondi. Questa base di conoscenza o KB è definibile come un insieme di assiomi, cioè delle proposizioni che possono essere asserite essere vere.

La base di conoscenza viene utilizzata nel caso di studio al fine di consentire all'utente e al sistema uno scambio di domande e risposte inerenti il dominio degli anime e dei loro generi attuando uno scambio di informazioni.

Nello specifico, l'utente può avanzare richieste come:

- Leggere i generi che appartengono all'anime

```
#Function that print an anime details
def checkanimegenre(anime_data):
    animename = input('Enter anime name : ')

    if animename in anime_data['name'].values:
        features = anime_data.columns[6:-1]
        print('Genre : ')
        for i in range(len(features.to_list())):
            if anime_data.at[ anime_data[anime_data['name'] == animename].index[0]
,features.to_list()[i]] == 1:
                print(features.to_list()[i])
        print('')
    else:
        print('There is no anime with that name in the dataset')
```

```
Enter anime name : Naruto
Genre :
Action
Comedy
Martial Arts
Shounen
Super Power
```

- Leggere i dettagli dell'anime (il punteggio, se è una serie o un film) partendo dal titolo come elemento di ricerca

```
#Function that print an anime details
def animedetails(anime_data):
    animename = input('Enter anime name as example Gintama : ')
    if animename in anime_data['name'].values:
        print(pd.DataFrame({'Anime name': anime_data['name'].loc[anime_data['name']
== animename].values,
                            'Rating': anime_data['rating'].loc[anime_data['name']
== animename].values,
                            'Type': anime_data['type'].loc[anime_data['name'] ==
animename].values}))

    else:
        print('There is no anime with that name in the dataset\n')
```

```
Enter anime name as example Gintama : Naruto
  Anime name  Rating Type
0     Naruto    7.81   TV
```

- Verificare che una serie abbia un determinato genere con una risposta affermativa o negativa

```
#Function that check one anime genre
def checkanime1genre(anime_data):
    animename = input('Enter anime name : ')
    if animename in anime_data['name'].values:
        features = anime_data.columns[6:-1]
        while True:
            print('Select one genres')
            print(' 1 Action\n 2 Adventure\n 3 Cars\n 4 Comedy\n 5 Dementia')
            print(' 6 Demons\n 7 Drama\n 8 Ecchi\n 9 Fantasy')
            print(' 10 Game\n 11 Harem\n 12 Hentai\n 13 Historical\n 14 Horror')
            print(' 15 Josei\n 16 Kids\n 17 Magic\n 18 Martial Arts')
            print(' 19 Mecha\n 20 Military\n 21 Music\n 22 Mystery\n 23 Parody')
            print(' 24 Police\n 25 Psychological\n 26 Romance\n 27 Samurai')
            print(' 28 School\n 29 Sci-Fi\n 30 Seinen\n 31 Shoujo\n 32 Shoujo Ai')
            print(' 33 Shounen\n 34 Shounen Ai\n 35 Slice of Life')
            print(' 36 Space\n 37 Sports\n 38 Super Power\n 39 Supernatural')
            print(' 40 Thriller\n 41 Vampire\n 42 Yaoi\n 43 Yuri')
            print(' if you enter a number that does not match a listed gender, the
question will be repeated')
            choice = input('Enter your choice : ')
            if int(choice) > 0 and int(choice) < 44:
                break
        if anime_data.at[ anime_data[anime_data['name'] == animename].index[0]
,features.to_list()[int(choice)]] == 1:
            print('Yes\n')
        else:
            print('No\n')
    else:
        print('There is no anime with that name in the dataset\n')
```

```
Enter anime name : Naruto
Select one genres
 1 Action
 2 Adventure
 3 Cars
 4 Comedy
 5 Dementia
 6 Demons
 7 Drama
 8 Ecchi
 9 Fantasy
 10 Game
 11 Harem
 12 Hentai
 13 Historical
 14 Horror
 15 Josei
 16 Kids
 17 Magic
 18 Martial Arts
 19 Mecha
 20 Military
 21 Music
 22 Mystery
 23 Parody
 24 Police
 25 Psychological
 26 Romance
 27 Samurai
 28 School
 29 Sci-Fi
 30 Seinen
 31 Shoujo
 32 Shoujo Ai
 33 Shounen
 34 Shounen Ai
 35 Slice of Life
 36 Space
 37 Sports
 38 Super Power
 39 Supernatural
 40 Thriller
 41 Vampire
 42 Yaoi
 43 Yuri
 if you enter a number that does not match a listed gender, the question will be repeated
Enter your choice : 1
No
```

- Dato il nome di due anime, restituisce risposta affermativa se esiste almeno un genere in comune tra loro

```
#Function that print yes if 2 anime have common genres (at least one)
def comparisonanime(anime_data):
    features = anime_data.columns[6:-1]
    exit = False
    animename1 = input('Enter first anime name : ')
    if animename1 in anime_data['name'].values:
        animename2 = input('Enter second anime name : ')
        if animename2 in anime_data['name'].values:
            for i in range(len(features.to_list())):
                for j in range(len(features.to_list())):
                    if anime_data.at[ anime_data[anime_data['name'] ==
animename1].index[0] ,features.to_list()[i]] == 1:
                        if anime_data.at[ anime_data[anime_data['name'] ==
animename1].index[0] ,features.to_list()[i]] == anime_data.at[
anime_data[anime_data['name'] == animename2].index[0] ,features.to_list()[j]]:
                            print('Yes\n')
                            exit = True
                            break
                if exit == True:
                    break
        else:
            print('There is no anime with that name in the dataset\n')
    else:
        print('There is no anime with that name in the dataset\n')

    if exit == False:
        print('No\n')
```

```
Enter first anime name : Naruto
Enter second anime name : Death Note
Yes
```