Corso di Fondamenti di Intelligenza Artificiale
Università degli Studi di Salerno

BOOK CATEGORIZATION

Mancuso Maria Angela
Malfettone Ines
Santonicola Federico
Sessa Attilio

Link GitHub

January 27, 2024

Indice

1 Introduzione

1.1 Scopo del progetto

Il nostro progetto si pone come obiettivo lo studio e la sperimentazione di differenti tecniche di Machine Learning capaci di analizzare ed estrarre informazioni da dati sotto forma di linguaggio naturale. Nello specifico si è interessati alla categorizzazione di libri tramite una breve descrizione testuale ed un elenco di autori. La categorizzazione è stata elaborata tramite due tecniche di Machine Learning: Classificazione e Clustering, che, seppur facenti utilizzo di due approcci diversi (rispettivamente apprendimento supervisionato e apprendimento non supervisionato), si considerano in grado di riportare risultati similari e confrontabili. A tal proposito occorrerà addestrare più modelli differenti riguardanti entrambe le tecniche e verificare quali di questi permetterà di ottenere il miglior risultato.

2 Specifiche del progetto

2.1 Ambiente: PEAS

PEAS	
Performance	La misura di prestazione è la capacità di avvicinarsi quanto più possibile al corretto genere di un libro specifico. È necessario utilizzare misure di prestazione differenti con lo scopo di valutare gli algoritmi di Classificazione e Clustering. Nel caso della Classificazione si è usato: accuratezza e report di classificazione. Quest'ultimo comprende la misura di precisione, recall, f1-score (per ciascuna classe) e una matrice di confusione tramite la quale è possibile visualizzare l'andamento degli algoritmi. Per il clustering, invece, si è utilizzato il Silhouette Score.

Environment	I modelli presi in considerazione sono stati realizzati e operano nell'ambiente di sviluppo "PyCharm" avente le seguenti caratteristiche: • **completamente osservabile**: il modello ha la visione completa del dataset, nello specifico degli attributi associati a ciascun libro. • **deterministico**: una volta addestrato un modello, lo stato dell'ambiente non varia a fronte degli stessi input. • **episodico**: l'agente delibera a fronte di determinati episodi che consistono in nuove richieste di predizione. • **statico**: l'ambiente resta invariato mentre l'agente opera. • **discreto**: è fornito un insieme discreto di informazioni per ciascun libro. • **singolo**: l'ambiente permette di addestrare più modelli valutati singolarmente.
Actuators	Gli agenti mostrano i risultati attraverso due tipi di attuatori: • console dell'ambiente di sviluppo: durante l'addestramento e testing i modelli riportano informazioni di controllo e i risultati ottenuti. • grafici esplicativi: mostrano informazioni di vario tipo, tra cui analisi del dataset e risultati ottenuti dalle predizioni dei modelli.
Sensors	I modelli ricevono le informazioni necessarie per l'addestramento tramite un file in formato csv contenente il dataset. Inoltre, è possibile specificare, grazie all'utilizzo della console, nuovi dati su cui effetturare nuove predizioni.

Table 1: Tabella della specifica PEAS

2.2 Metodologie utilizzate

Come si evidenzia dallo scopo del progetto, ci si trova ad affrontare un'istanza di un problema di analisi testuale in linguaggio naturale. Tale classe di problemi risulta così complessa, ma anche così rilevante nell'ambito del machine learning, da essersi meritata una disciplina a sé nota come NLP (Natural Language Process). In genere, analizzare un testo in linguaggio naturale è molto complesso per differenti motivi, tra cui:

- Ambiguità del linguaggio;
- Uno stesso concetto può essere espresso in molti modi differenti;
- Errori di ortografia;
- Parole "rumorose" che non aggiungono significato;

Tuttavia, nel nostro contesto non è necessario concentrarsi sull'estrazione di semantica, in un testo in cui la semantica è nascosta, ma è molto più importante evidenziare la presenza di parole chiave che ci permettano di determinare la categoria di appartenenza di un dato libro. A titolo di esempio, si prenda in considerazione la seguente descrizione: *"Dei cavalieri affrontano ed uccidono un drago in un regno lontano"*. Ai fini del nostro scopo non è importante dedurre informazioni contestuali o semantiche (come il fatto che avvenga uno scontro) ma è sufficiente notare la presenza di determinate parole (come "*cavalieri*", "*drago*", "*regno*") per dedurre la categoria del libro (in questo caso "*Fantasy*"). Notato ciò, la nostra attenzione si pone su una specifica tecnica di NLP detta "Bag of Words", il cui funzionamento consiste nella creazione di un vocabolario di tutte le differenti parole presenti in un corpus di testi e ciascuno documento sarà rappresentato da un vettore di conteggi delle parole presenti. È evidente che, se una determinata descrizione contiene un gran numero di parole come "*cucina*" o "*ricetta*", è molto più probabile che si tratti di un libro di cucina, piuttosto che un romanzo. Nonostante ciò, risulta comunque opportuno effettuare delle operazioni di pulizia del testo che ci permettano di ridurre il "rumore" e rendere descrizioni e autori più uniformi. Ad esempio, sono moltissime le parole che non danno nessun tipo di contributo alla determinazione della categoria (come tutte le congiunzioni o parole ridondanti come "*book*" o "*new*") la cui rimozione permetterebbe di evidenziare maggiormente termini più significativi (si veda il paragrafo 3.2).

3 Analisi e preparazione dei dati

3.1 Scelta del dataset

Per lo scopo di tale progetto si è scelto un dataset già esistente, reperibile qui. La ricerca del dataset si è basata sulla necessità di trovarne uno contenente libri provveduti di descrizione e autori. Il dataset scelto fornisce informazioni relative a 103063 libri ed ha una dimensione di 69.75MB. Tutti i dati sono in formato testuale.

3.2 Data cleaning

Nella fase di Data Cleaning ci si occupa di "pulire" il dataset mantenendo solo i dati utili all'addestramento del modello. Da una prima osservazione dei dati è stata rilevata la presenza di due aspetti che possono ostacolare la corretta esecuzione degli algoritmi di classificazione e di clustering, che saranno implementati successivamente. La funzione "clearData" si occupa della pulizia dei dati, con le seguenti operazioni:

1) eliminazione dei campi nulli. In tal modo si è ottenuto un dataset senza dati mancanti.
2) eliminazione delle righe duplicate.
3) eliminazione delle righe non significative. Per identificare le frasi non significative è stato utilizzato il parametro "des_threshold" che indica un limite. Se le frasi contengono un numero di caratteri inferiore a "des_treshold" allora verranno considerate non significative.

La funzione "clearData" applica la prima e la seconda operazione su tutti i dati presenti all'interno del dataset. Mentre la terza operazione verrà eseguita solo sulla feature delle descrizioni, in quanto applicarla agli autori potrebbe risultare controproducente.

```
def cleanData(dataframe, des_threshold = 40):
    null_rows = len(dataframe[dataframe.isna().any(axis=1)])  # Numero di righe che hanno un qualsiasi campo nullo
    dataframe.dropna(inplace=True)   # Rimozione di righe con valori nulli
    print(f"Rimosse {null_rows} righe con valori nulli")

    identical_rows = dataframe.duplicated().sum()  # Numero di righe identiche
    dataframe.drop_duplicates(inplace=True)        # Rimozione di righe identiche
    print(f"Rimosse {identical_rows} righe duplicate")

    drop_indexes = dataframe[dataframe["Description"].str.len() < des_threshold].index   # Indici delle righe descrizion
    dataframe.drop(drop_indexes, inplace=True)  # Rimozione righe con descrizioni non significative
    print(f"Rimosse {len(drop_indexes)} righe con descrizioni non significative (meno di {des_threshold} caratteri)")
```

Dopo una successiva analisi del dataset e delle categorie presenti al suo interno, si è notata la presenza di alcune categorie similari. Per assicurarci una predizione ottimale del genere è necessario identificare le categorie nella loro unicità. A tale scopo viene utilizzata la funzione "renameCategories". Essa si occupa di rinominare le categorie utilizzando una lista di rinominazioni.

```
f renameCategories(dataframe, replacementList):
  for replacement in replacementList:
      dataframe["Category"] = dataframe["Category"].str.replace(pat=rf"\s*({replacement[0]})\s*", repl=replacement[1], regex=True)
```

Di lato è mostrata la lista passata alla funzione "renameCategories":

```
replacementList = [
    ("Religious", "Religion"),
    ("Christian Life", "Religion"),
    ("Historical", "History"),
    ("Family", "Family & Relationships"),
    ("Pets", "Animal")
]
```

Sono state selezionate sedici categorie differenti. Tale selezione è stata effettuata prendendo in considerazione la frequenza della ricorrenza di ciascuna categoria. Questo ha portato all' eliminazione di categorie aventi troppi pochi dati per permettere ai modelli di ottenere un buon riconoscimento della categoria stessa. Tale operazione viene effettuata tramite la funzione "extractCategories".

La seguente funzione si occupa di estrarre le categorie di interesse in modo da poter eliminare tutte quelle righe non appartenenti ad esse. Per ogni riga vengono svolte le seguenti azioni:

- estrarre tutte le categorie separate da ',';
- se nessuna delle categorie estratte appartiene a 'toExtract', la riga viene eliminata;
- tra le categorie che fanno parte di 'toExtact', viene assegnato al campo "Category" quella che ha meno esempi nel dataframe.

```python
def extractCategories(dataframe, toExtract):
    categories = dataframe["Category"].str.split(r"\s*,\s*", expand=True).stack()  # Ottengo la list
    categories = categories.str.strip()  # Rimuovo spazi vuoti superflui
    categories = categories.str.lower()  # Mi assicuro che tutte le lettere siano in minuscolo
    toExtract = [cat.strip().lower() for cat in toExtract]

    categories_counts = categories.value_counts()    # Ottengo il conteggio di ciascuna categoria
    drop_indexes = []  # Lista degli indici delle righe che saranno droppate dal dataframe

    for index, row in dataframe.iterrows():
        categories_splitted = [splitted.strip().lower() for splitted in row["Category"].split(",")]

        changed = False
        for cat in categories_splitted:   # Per ogni categoria della riga
            if (cat in toExtract) and (cat in categories_counts):          # Verifico se la categori
                if (not changed) or (categories_counts[cat] < categories_counts[row["Category"]]):
                    row["Category"] = cat
                    changed = True

        if changed:     # Se la riga ha subito un cambiamento allora sovrascivo il cambiamento avven
            dataframe["Category"].loc[index] = row["Category"]
        else:           # altrimenti considero la riga da eliminare, in quanto non appartiene a ness
            drop_indexes.append(index)

    dataframe.drop(drop_indexes, inplace=True)      # Elimino le righe considerate da scartare
```

Di seguito sono mostrate le categorie estratte:

```python
extractList = ["Religion", "Romance", "Cooking", "History",
               "Business & Economics", "Thrillers", "Mystery & Detective",
               "Health & Fitness", "Art",  "Sports & Recreation", "Travel",
               "Fantasy", "Science", "Animals", "Computers", "House & Home"]
```

Lavorare su dati testuali significa lavorare sul Natural Language. Quest'aspetto implica una sequenza di azioni di normalizzazione da applicare sulle descrizioni e sugli autori. Ma prima di normalizzare i dati risulta più efficiente eseguire un'operazione di bilanciamento del dataset (vedere paragrafo 3.4) nel quale vengono eliminate le righe di quelle categorie troppo frequenti. Le operazioni di normalizzazione sono svolte dalle funzioni "preprocessAuthors" e "preprocessDescription". Queste funzioni utizzano le stop words che vengono identificate prima della loro esecuzione. Tramite l'utilizzo delle librerie NLTK sono state individuate le stop words e sono state successivamente aggiunte ai file "authors_stopwords.txt" e "description_stopwords.txt" in modo da distinguere le stop words all'interno delle features "authors" e "descriptions".

La funzione "preprocessDescrition" esegue le seguenti operazioni:

- i caratteri vengono resi tutti minuscoli in modo da rendere i dati testuali più uniformi e leggibili;
- eliminazione della punteggiatura per eliminare il rumore;
- eliminazione delle stop words;
- lemmatizzazione, una tecnica che permette la riduzione delle parole nella loro forma radice considerando sia la grammatica che il significato (esempio: *playing* *play*).

```
def preprocessDescription(dataframe, stop_words = stopwords.words('english')):
    newDescriptions = []
    for index, row in dataframe.iterrows():
        text = row["Description"]

        text = text.lower()  # Ogni lettere viene portata in minuscolo

        text = "".join([char for char in text if char not in punctuation])  # Rimozione della punteggiature

        words = word_tokenize(text)
        filtered_words = [word for word in words if word not in stop_words]  # Rimozione stopwords

        lemmatizer = WordNetLemmatizer()
        lemmed = [lemmatizer.lemmatize(word) for word in filtered_words]  # Lemming

        newDescriptions.append(" ".join(lemmed))

    dataframe["Description"] = newDescriptions
```

La funzione "preprocessAuthors" eseguirà le stesse prime tre operazioni di "preprocessDescription".

```
def preprocessAuthors(dataframe, stop_words = []):
    newAuthors = []
    if "by" not in stop_words:
        stop_words.append("by")

    editors = dataframe["Authors"].str.extract(r"\((\w+)\)", expand=False).dropna().unique().tolist()
    editors = [word.lower() for word in editors]
    for word in editors:
        if word not in stop_words:
            stop_words.append(word)

    for index, row in dataframe.iterrows():
        text = row["Authors"]

        text = text.lower()  # Ogni lettere viene portata in minuscolo

        text = "".join([char for char in text if char not in punctuation])  # Rimozione punteggiatura

        words = word_tokenize(text)
        filtered_words = [word for word in words if word not in stop_words and len(word) > 1]  # Rimoz

        newAuthors.append(" ".join(filtered_words))

    dataframe["Authors"] = newAuthors
```

Per illustrare la corretta esecuzione delle due funzioni mostrate precedentemente vengono utilizzate le WordCloud (vedere capitolo 3.5). Queste vengono utilizzate per osservare facilmente come i dati testuali siano cambiati e resi più semplci e funzionali. In conclusione, dopo la fase di Data Cleaning, avremmo un database completo e leggibile che garantisca sia la qualità che la quantità dei dati.

3.3 Feature Selection

Il dataset orginale presentava sette features diverse: Title, Authors, Description, Category, Publisher, Publish Date e Price. Per lo scopo del nostro progetto verrano prese in considerazioni solo le feature che contengono: il titolo, per una questione di chiarezza, la descrizione e gli autori, poiché sono le due caratteristiche che utilizzeremo per predirre il genere del libro, e il genere per organizzare i libri e valutare se la predizione ha dato esito positivo.

3.4 Bilanciamento dataset

Da una prima osservazione del grafico, che mostra i libri divisi nei 20 generi, si può notare lo sbilanciamento del dataset.

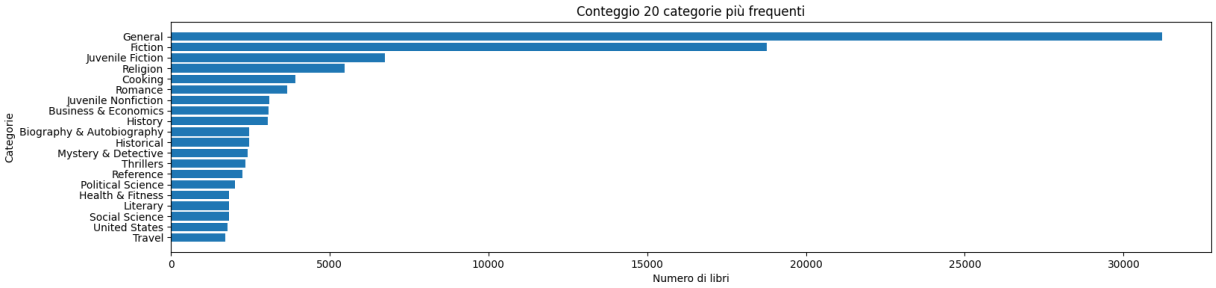


Figure 1: Database non bilanciato

Dunque, per ovviare a problemi di overfitting, si è ritenuto necessario bilanciarlo quanto più possibile. Si passa dunque al seguente dataset:

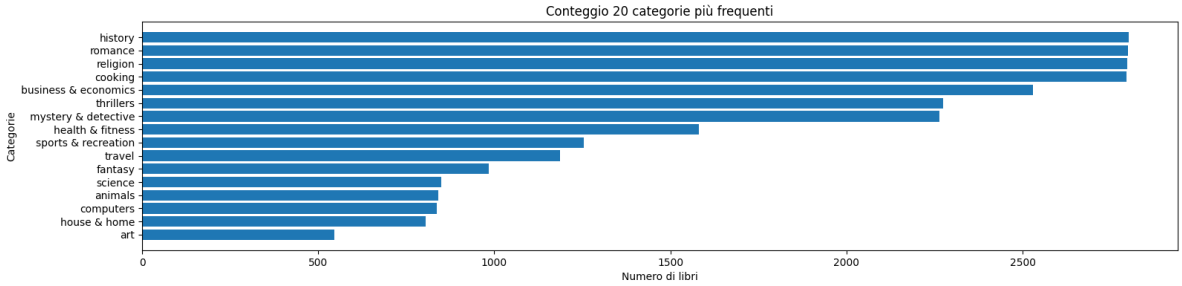


Figure 2: Database bilanciato

Come si nota dalla 'Figure 2' il dataset non risulta completamente bilanciato. Questo risultato è dovuto alla natura stessa del dataset. Per ottenere un dataset bilanciato si sarebbero dovute eliminare molte informazioni, non avendo così più a disposizione abbastanza dati per addestrare i nostri modelli.

Il bilanciamento del dataset è stato effettuato tramite la funzione "balanceCategories", la quale verifica il numero di righe per ogni categoria ed elimina casualmente delle righe di categorie con più di 'threshold' esempi.

```python
def balanceCategories(dataframe, threshold):
    drop_indexes = []

    for category in dataframe["Category"].unique():
        remove_n = len(dataframe[dataframe["Category"] == category]) - threshold

        if remove_n > 0:
            indexes = np.random.choice(dataframe[dataframe["Category"] == category].index, remove_n, replace=False)
            drop_indexes.extend(indexes)
            print(f"Rimosse {remove_n} righe dalla categoria {category}")

    dataframe.drop(drop_indexes, inplace=True)
```

Possiamo concludere che sono state effettuate le seguenti modifiche del dataset in questione, con la rimozione del seguente numero di righe:

```
-----------------------------------------------
Bilanciamento del dataset
Rimosse 1380 righe dalla categoria religion
Rimosse 910 righe dalla categoria history
Rimosse 84 righe dalla categoria cooking
Rimosse 595 righe dalla categoria romance
-----------------------------------------------
```

3.5 Visualizzazione Word Cloud

Word Cloud è una tecnica per la visualizzazione dei dati. È utilizzata per rappresentare dati di testo in cui la dimensione di ciascuna parola ne indica l'importanza. Grazie all'utilizzo delle WordCloud, nello specifico tramite l'utilizzo delle funzioni "createDescriptionWordCloud" e "createAuthorsWordCloud", è stato possibile illustrare le parole chiavi presenti all'interno delle descrizioni e degli autori. Verrano creati tanti WordCloud quante sono le categorie.

```python
def createDescriptionWordCloud(dataframe, save_file = "description_wordcloud"):
    n_categories = len(dataframe["Category"].unique())
    n_rows, n_cols = (int(n_categories/2), 2)

    fig, ax = plt.subplots(n_rows, n_cols, figsize=(10, n_rows*3))
    fig.suptitle('WordCloud delle descrizioni', fontsize=40, y=1)
    fig.tight_layout()

    contAx = 0
    for category in dataframe["Category"].unique():
        descriptions = dataframe[dataframe["Category"] == category]["Description"].tolist()
        text = ' '.join(descriptions)
        wordcloud = WordCloud(background_color='white').generate(text)

        ax[int(contAx/2), contAx%2].imshow(wordcloud)
        ax[int(contAx/2), contAx%2].axis('off')
        ax[int(contAx/2), contAx%2].set_title(category, fontsize=20, pad=10)

        contAx += 1

    plt.savefig(f"Plots/{save_file}.png")
```

La seguente immagine è un esempio di output della funzione (con lo scopo di farne comprendere il corretto funzionamento, vengono mostrate solo quattro categorie):

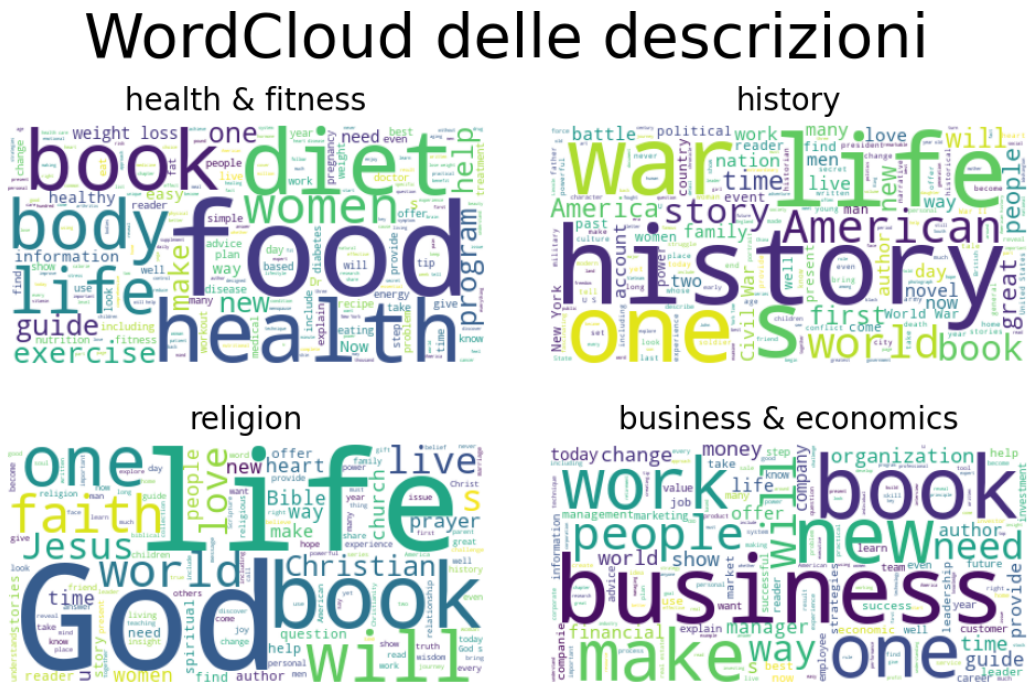


Figure 3: Word Cloud delle descrizioni non processate

Tale illustrazione ci permette di analizzare con più semplicità le parole ricorrenti ed individuare delle stop words aggiuntive, che fanno riferimento al dataset in questione, come ad esempio "*book*" e "*life*". Allo stesso ragionamento vengono sottoposti gli autori. Per le stop words aggiuntive, inerenti alla descrizione e agli autori, sono stati creati due nuovi file di testo, chiamati rispettivamente "description_stopwords.txt" e "authors_stopwords.txt". Di seguito sono mostrate le stop word individuate:

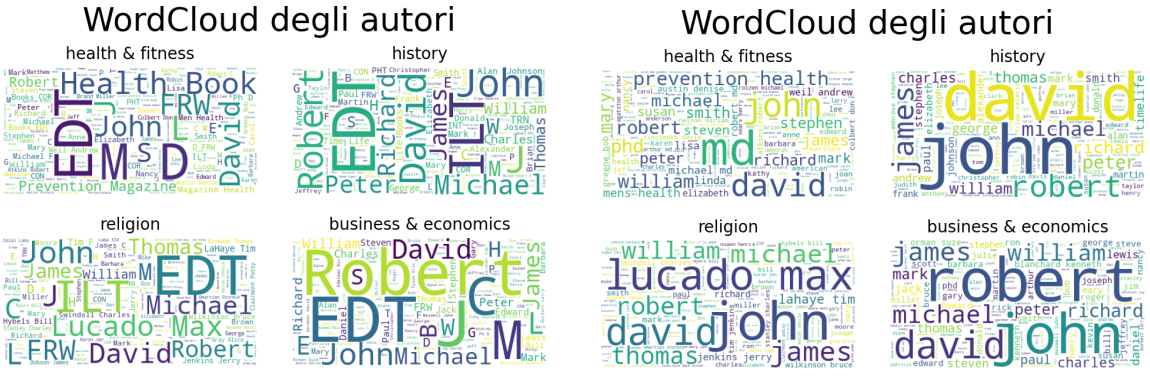
(a) Stop Word delle descrizioni

(b) Stop Word degli autori

Figure 4: Stop Word aggiunte

L'eliminazione delle stop words avviene tramite le funzioni "preprocessDescription", per le descrizioni, e "preprocessAuthors", per gli autori. Si ottiene il seguente word cloud riguardanti le descrizioni:

Figure 5: Word Cloud delle descrizioni processate



(a) Word Cloud degli autori non processati (b) Word Cloud degli autori processati

Figure 6: Word Cloud degli autori

3.6 Formattazione dei dati

Dopo le opportune operazioni di preparazione del dataset, è necessario rendere i dati testuali utilizzabili dai modelli di machine learning. Com'è già stato specificato precedentemente (si veda paragrafo 2.2), il testo verrà convertito in vettori di numeri, rappresentanti i conteggi delle singole parole. Affinché l'addestramento dei modelli avvenga adeguatamente, è necessario effettuare un'operazione di normalizzazione di tali conteggi. In questo modo si garantisce che testi riguardanti la stessa categoria, ma con un numero di parole chiave differenti, abbiano uno stesso peso in fase di addestramento (altrimenti si rischierebbe di sovrastimare testi lunghi e sottostimare testi corti). La tecnica di normalizzazione tipicamente utilizzata sui conteggi di "Bag of words" è detta TF-IDF (term frequency–inverse document frequency) la quale consente di riportare i conteggi sotto forma di frequenze, soppesate per un contributo direttamente proporzionale al numero di volte che la parola è contenuta nel documento e inversamente proporzionale al numero di documenti in cui questa è presente. Dunque viene data maggior importanza a parole molto frequenti in un documento ma abbastanza uniche e significative da non essere presenti in tanti documenti differenti.

4 Classificazione

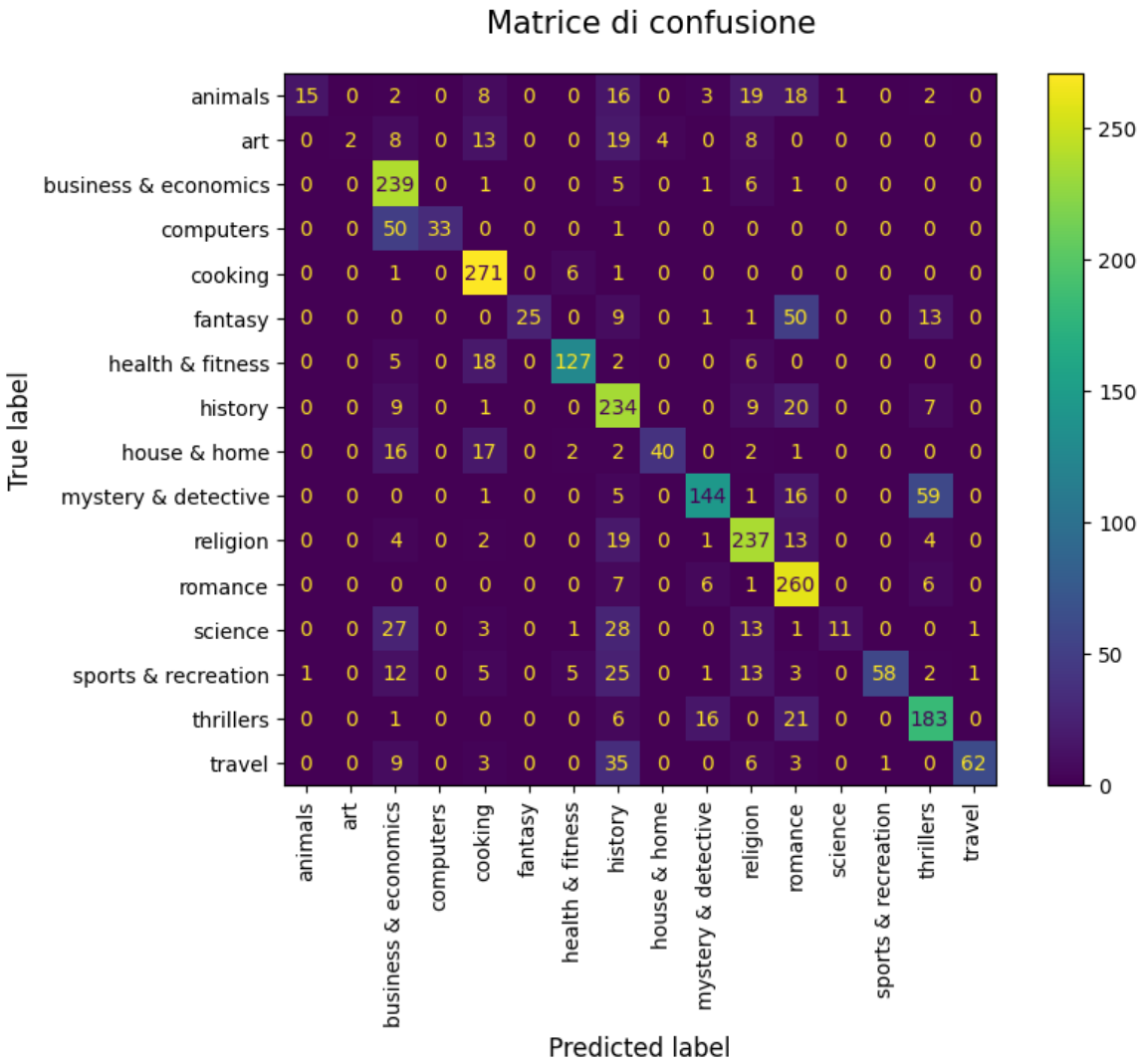
La classificazione fa uso di tecniche di apprendimento supervisionato al fine di predire la variabile target (Category) sulla base delle variabili indipendenti (Descriptions, Authors). Esistono svariati modelli di classificazione che operano sotto condizioni ed ipotesi differenti: alcuni di questi sono più adatti per dataset sbilanciati, altri per lavorare con matrici sparse (come quella ottenuta dall'elaborazione dei testi tramite TF-IDF), ecc. Nel nostro contesto, la classificazione basata su linguaggio naturale presenta molte problematiche per le quali non è possibile identificare un modello adatto alla loro risoluzione. A tal proposito, differenti modelli di classificazione sono stati addestrati e valutati, avendo cura di garantire un confronto equo con l'utilizzo degli stessi dati in tutti i modelli sia per l'addestramento che per il testing. Per ogni modello sono state calcolate e riportate le seguenti metriche:

- Accuracy;
- Tempo necessario per l'addestramento;
- Classification report: precisione, recall e f1-score per ciascuna classe predetta;
- Matrice di confusione delle predizioni effettuate.

4.1 Mulinomial Naive Bayes Classification

Il classificatore Multinomial Naive Bayes è una tipologia di classificatore probabilistico bayesiano che lavora secondo l'assunzione per cui le caratteristiche predittive siano organizzate secondo una distribuzione multinomiale. Risulta particolarmente adatto nei contesti in cui le caratteristiche degli elementi da predire rappresentano conteggi di occorrenze. Tuttavia, nel pratico anche i conteggi frazionari dovrebbero funzionare sufficientemente bene. Il suo addestramento ha impiegato all'incirca 5.14 secondi, ottenendo un punteggio di accuracy di 0.715. Di seguito ne si riporta il suo classification report e la matrice di confusione delle predizioni:

	precision	recall	f1-score	support
animals	0.94	0.18	0.30	84
art	1.00	0.04	0.07	54
business & economics	0.62	0.94	0.75	253
computers	1.00	0.39	0.56	84
cooking	0.79	0.97	0.87	279
fantasy	1.00	0.25	0.40	99
health & fitness	0.90	0.80	0.85	158
history	0.57	0.84	0.67	280
house & home	0.91	0.50	0.65	80
mystery & detective	0.83	0.64	0.72	226
religion	0.74	0.85	0.79	280
romance	0.64	0.93	0.76	280
science	0.92	0.13	0.23	85
sports & recreation	0.98	0.46	0.63	126
thrillers	0.66	0.81	0.73	227
travel	0.97	0.52	0.68	119

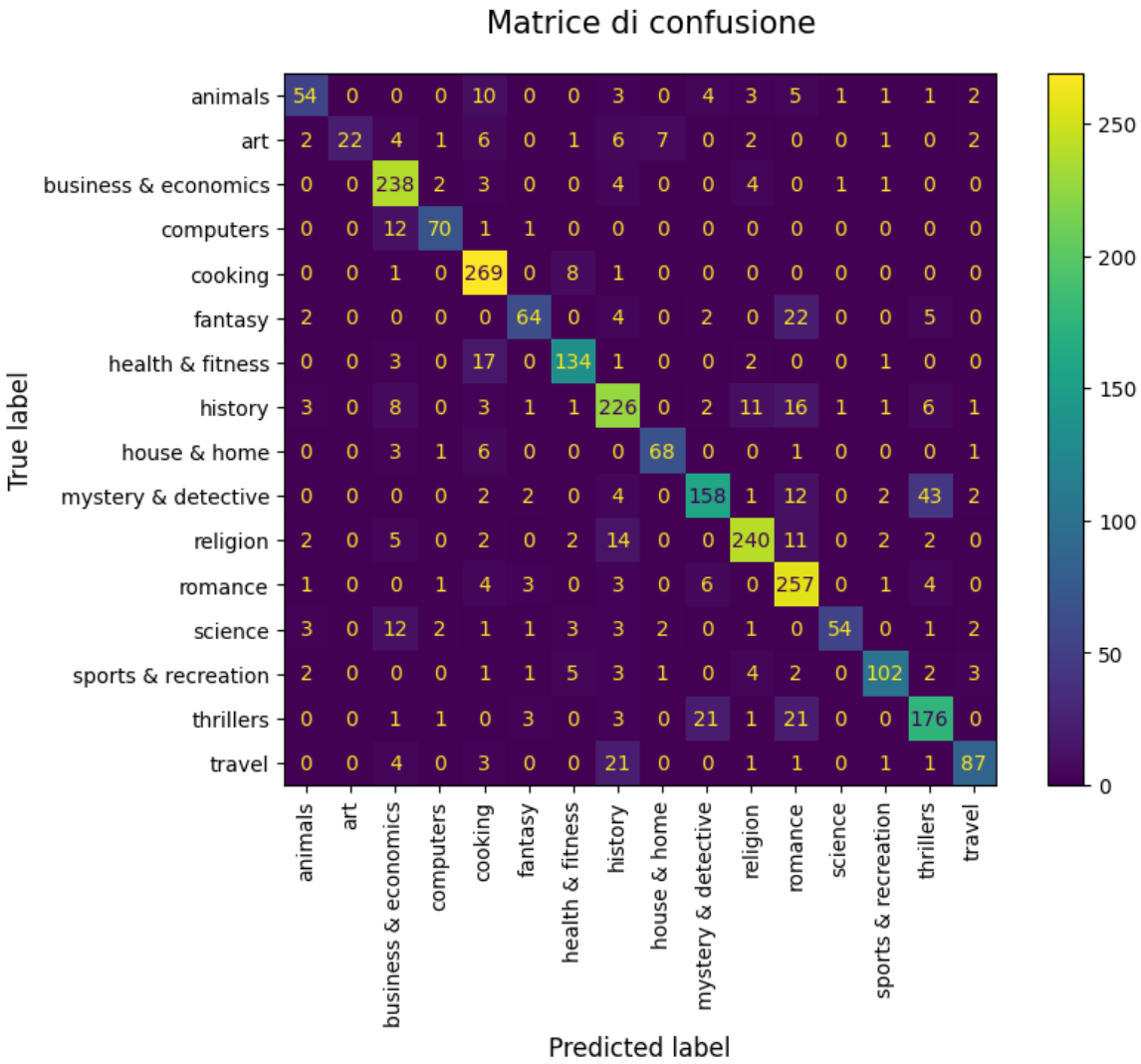


Tra tutti i classificatori provati, questo è risultato essere quello più rapido durante l'apprendimento, a discapito di un punteggio di accuracy e, in generale, di risultati nettamente inferiori rispetto agli altri. Si può notare, infatti, come il modello faccia confusione su diverse categorie, non riuscendo a distinguerle da altre (si notino le predizioni fatte per "*Animals*" e "*Art*"). Le categorie su cui presenta più difficoltà sono di fatto quelle con minor numero di esempi all'interno del dataset, rispecchiando quelle che sono le problematiche principali di un qualsiasi classificatore bayesiano.

4.2 Complement Naive Bayes Classification

Il classificatore Complement Naive Bayes è una variante del classificatore bayesiano multinomiale, progettato per affrontare situazioni in cui le classi del problema in esame risultino particolarmente sbilanciate. Nonostante i risultati mediocri riportati dal precedente classificatore, si è comunque deciso di provate un ulteriore classificatore bayesiano per verificare se l'approccio adottato dal Complement risulti maggiormente adatto a dataset sbilanciati rispetto al Multinomial. Il suo addestramento ha impiegato all'incirca 5.75 secondi, ottenendo un punteggio di accuracy di 0.818. Di seguito ne si riporta il suo classification report e la matrice di confusione delle predizioni:

	precision	recall	f1-score	support
animals	0.78	0.64	0.71	84
art	1.00	0.41	0.58	54
business & economics	0.82	0.94	0.87	253
computers	0.90	0.83	0.86	84
cooking	0.82	0.96	0.89	279
fantasy	0.84	0.65	0.73	99
health & fitness	0.87	0.85	0.86	158
history	0.76	0.81	0.78	280
house & home	0.87	0.85	0.86	80
mystery & detective	0.82	0.70	0.75	226
religion	0.89	0.86	0.87	280
romance	0.74	0.92	0.82	280
science	0.95	0.64	0.76	85
sports & recreation	0.90	0.81	0.85	126
thrillers	0.73	0.78	0.75	227
travel	0.87	0.73	0.79	119

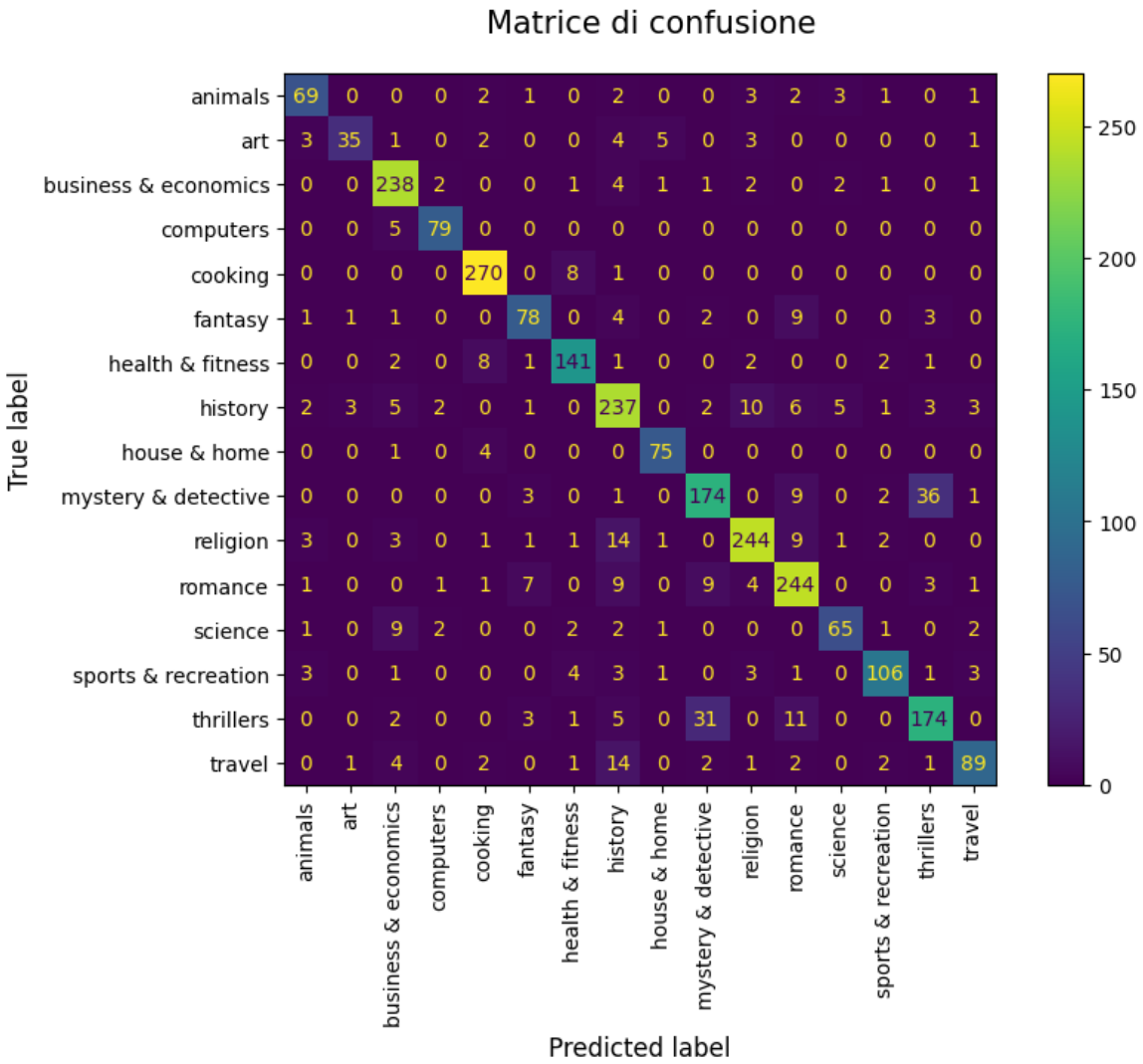


È possibile notare, fin da subito, risultati nettamente migliori rispetto al classificatore multinomiale, con un tempo di addestramento che, seppur leggermente superiore, è comunque paragonabile al precedente. Osservando la matrice di confusione, come ci si può aspettare da un classificatore bayesiano, si osservano ancora delle lacune nelle predizioni delle classi aventi meno esempi. Nonostante ciò, risulta evidente che l'approccio adottato dal classificatore Complement, per dataset sbilanciati, ha permesso al modello di adattarsi meglio ai dati disponibili e, seppur non ottimo per un problema di classificazione di testi, è comunque un modello valido e consigliabile in altri contesti.

4.3 Liner Support Vector Classification

Il classificatore Linear Support Vector è una tipologia di classificatore lineare binario. Il suo funzionamento è basato sulla rappresentazione degli esempi da predire, come punti in uno spazio multidimensionale, ed ha l'obiettivo di identificare il piano che meglio separa tali esempi in due categorie ben distinte. Essendo un classificatore binario, normalmente è in grado di classificare solo due categorie. Tuttavia, grazie all'integrazione della tecnica One-Vs-Rest, tale classificatore risulta in grado di risolvere anche problemi multiclasse. Secondo la documentazione riportata da Sklearn, il modello LinearSVC è adatto ad input sotto forma di matrici sparse e per tal motivo è stato sottoposto alle nostre valutazioni con l'obiettivo, inoltre, di osservare quanto un modello basato su un approccio lineare (dunque totalmente differente dagli approcci precedenti) potesse adattarsi bene ad un problema di classificazione di testi. Il suo addestramento ha impiegato all'incirca 7.36 secondi, ottenendo un punteggio di accuracy di 0.854. Di seguito ne si riporta il suo classification report e la matrice di confusione delle predizioni:

	precision	recall	f1-score	support
animals	0.83	0.82	0.83	84
art	0.88	0.65	0.74	54
business & economics	0.88	0.94	0.91	253
computers	0.92	0.94	0.93	84
cooking	0.93	0.97	0.95	279
fantasy	0.82	0.79	0.80	99
health & fitness	0.89	0.89	0.89	158
history	0.79	0.85	0.82	280
house & home	0.89	0.94	0.91	80
mystery & detective	0.79	0.77	0.78	226
religion	0.90	0.87	0.88	280
romance	0.83	0.87	0.85	280
science	0.86	0.76	0.81	85
sports & recreation	0.90	0.84	0.87	126
thrillers	0.78	0.77	0.78	227
travel	0.87	0.75	0.81	119

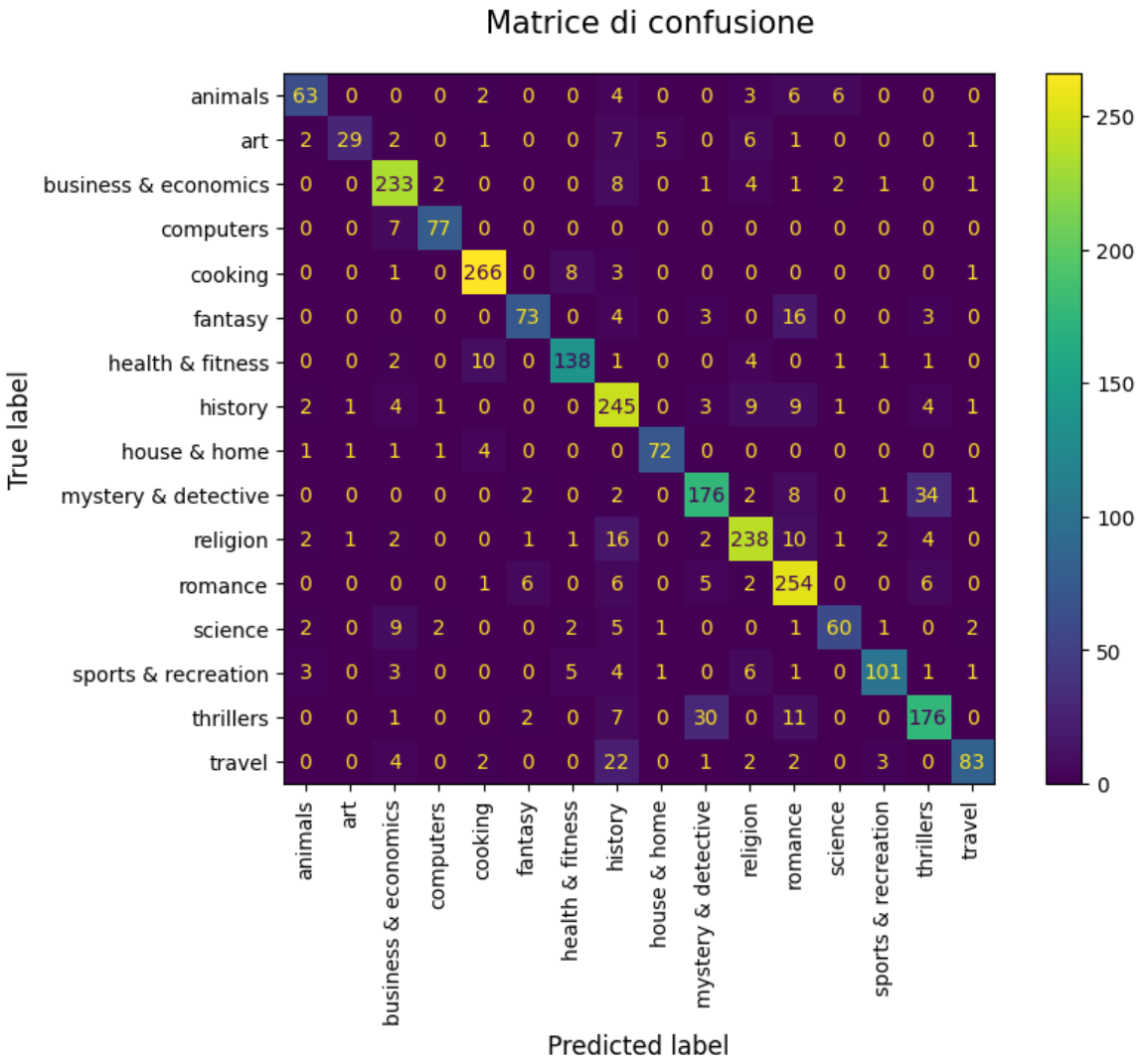


Si notano risultati decisamente migliori rispetto ai modelli precedenti. Il tempo per l'apprendimento è ancora sufficientemente basso e le predizioni sono state molto più accurate e con meno lacune rispetto le categorie meno frequenti. Dal classification report si nota come la categoria "Art" sia quella con lo score minore (f1-score di 0.74), come ci si poteva aspettare dalla classe con meno esempi nel dataset. In generale, le predizioni effettuate sono abbastanza soddisfacenti. Grazie alla matrice di confusione è possibile notare una tendenza, più evidente rispetto ai modelli precedenti, nel confondere alcune categorie simili. Di fatto un buon numero di libri appartenenti alla categoria "Mistery & Detective" sono stati confusi come "Thriller" (e viceversa). Osservando i wordcloud delle categorie in questione sono presenti parole chiave comuni (ad esempio: "*murder*", "*killer*"): risulta dunque evidente la difficoltà nella suddivisione netta delle due tipologie di libri (è una tendenza che ci si aspetta di osservare anche dai successivi classificatori).

4.4 Logistic Regression Classification

Il classificatore Logistico, anche detto Regressore Logistico, è una tipologia di classificatore lineare binario. Il suo obiettivo è l'identificazione di una retta (o un piano, nel caso in cui le feature fossero superiori a due) in grado di stimare la probabilità che un esempio faccia parte di una determinata classe piuttosto che di un'altra. Dalla probabilità stimata è poi facile ricondurre la categoria da predire, passando dunque da un problema risolto con un approccio regressivo ad uno di classificazione. Tale algoritmo supporta l'integrazione della tecnica "One-Vs-Rest" per effettuare predizioni multiclasse e risulta essere adatto ad input sotto forma di matrici sparse. Considerando i risultati ottenuti dal classificatore precedente, si è deciso di valutare un ulteriore classificatore lineare, con una tipologia di approccio misto regressione – classificazione. Il suo addestramento ha impiegato all'incirca 114.64 secondi, ottenendo un punteggio di accuracy di 0.842. Di seguito ne si riporta il suo classification report e la matrice di confusione delle predizioni:

	precision	recall	f1-score	support
animals	0.84	0.75	0.79	84
art	0.91	0.54	0.67	54
business & economics	0.87	0.92	0.89	253
computers	0.93	0.92	0.92	84
cooking	0.93	0.95	0.94	279
fantasy	0.87	0.74	0.80	99
health & fitness	0.90	0.87	0.88	158
history	0.73	0.88	0.80	280
house & home	0.91	0.90	0.91	80
mystery & detective	0.80	0.78	0.79	226
religion	0.86	0.85	0.86	280
romance	0.79	0.91	0.85	280
science	0.85	0.71	0.77	85
sports & recreation	0.92	0.80	0.86	126
thrillers	0.77	0.78	0.77	227
travel	0.91	0.70	0.79	119

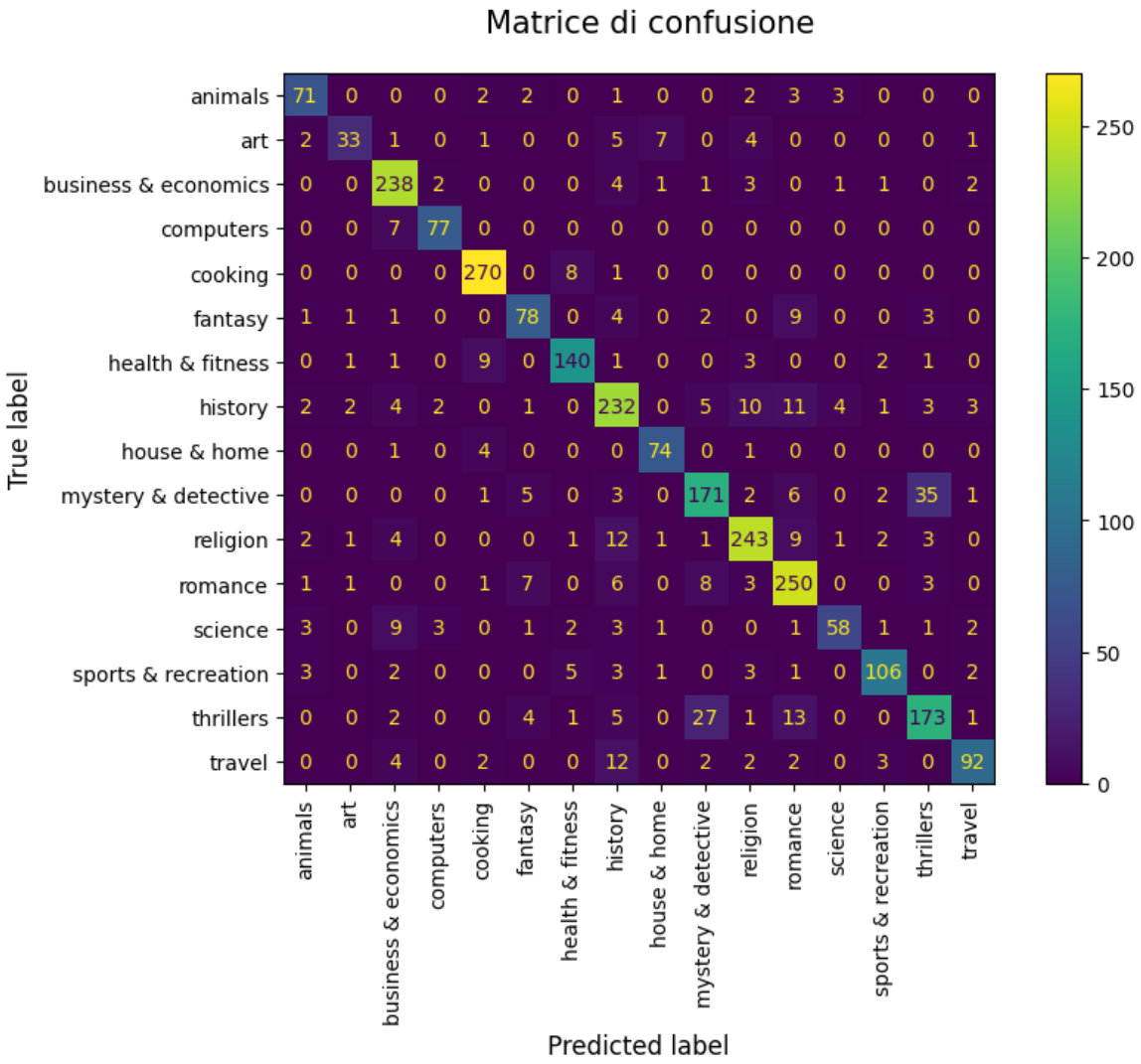


Tra tutti i classificatori provati, questo è il classificatore più lento in fase di addestramento. Com'è noto, gli algoritmi di apprendimento, per problemi di regressione lineare, impiegano molto tempo per apprendere le funzioni lineari che meglio si adattano ai dati, soprattutto se si lavora con dati in larga scala (di numero di righe e numero di feature). Durante i vari test condotti sul regressore logistico è stato molto frequente l'evento in cui l'algoritmo non è stato in grado di convergere ad una soluzione riportando, ovvero un errore. Indipendentemente da ciò, i risultati ottenuti sono di poco al di sotto del classificatore precedente e, in generale, abbastanza soddisfacenti. Si può notare una maggior confusione, rispetto a quanto visto nel Linear Support Vector, verso ulteriori categorie simili, come per esempio "Religion" - "History" e "Fantasy" - "Romance". Di conseguenza tutti i punteggi f1-score, riportati per le varie categorie, risultano essere più bassi.

4.5 Stochastic Gradient Descent Classification

Il classificatore Stochastic Gradient Descent non fa parte di una specifica famiglia di modelli di machine learning, ma è una tecnica di ottimizzazione che permette di addestrare un modello di classificazione in maniera efficiente. Secondo la documentazione fornita da Sklearn, a seconda della configurazione degli iperparametri che gli vengono forniti, tale classificatore utilizza specifiche tecniche adottate già da altri modelli con l'obiettivo di ottimizzarle. Tali tecniche si basano su approcci lineari e, visti gli ottimi risultati ottenuti dai precedenti classificatori lineari, sembra ragionevole considerare anche quest'ulteriore classificatore. È un modello adatto per apprendimenti su dati in larga scala, sia in termini di numeri di esempi che di feature per ciascuno di essi. Di conseguenza, dovrebbe risultare ottimo nei problemi di classificazione di testo e, in generale, nel natural language process. Il suo addestramento ha impiegato all'incirca 7.23 secondi, ottenendo un punteggio di accuracy di 0.85. Di seguito ne si riporta il suo classification report e la matrice di confusione delle predizioni:

	precision	recall	f1-score	support
animals	0.84	0.85	0.84	84
art	0.85	0.61	0.71	54
business & economics	0.87	0.94	0.90	253
computers	0.92	0.92	0.92	84
cooking	0.93	0.97	0.95	279
fantasy	0.80	0.79	0.79	99
health & fitness	0.89	0.89	0.89	158
history	0.79	0.83	0.81	280
house & home	0.87	0.93	0.90	80
mystery & detective	0.79	0.76	0.77	226
religion	0.88	0.87	0.87	280
romance	0.82	0.89	0.85	280
science	0.87	0.68	0.76	85
sports & recreation	0.90	0.84	0.87	126
thrillers	0.78	0.76	0.77	227
travel	0.88	0.77	0.83	119



I risultati ottenuti sono molto simili a quelli ricavati dal Linear Support Vector. Tempi di addestramento e punteggi sono del tutto analoghi. Inoltre, durante i vari test condotti sui modelli, si è potuto notare come in effetti nessuno dei due modelli prevalga in assoluto sull'altro.
Per alcuni testi il Linear Support Vector è risultato migliore, per altri, invece, lo è stato SGD. Tale modello rappresenta un'ulteriore prova di come gli approcci lineari risultino ottimi per problemi di classificazione di testo.

4.6 Valutazioni finali sulla Classificazione

Dai risultati ottenuti si evidenzia come i classificatori bayesiani siano del tutto sconsigliabili per problemi di classificazione testuale: seppur abbastanza rapidi nella fase di addestramento, non sembrano adattarsi bene ad uno spazio di esempi con numerose feature e rischiano facilmente di andare in underfitting.
La classificazione tramite approccio regressivo adottata dal regressore lineare sembra essere sufficientemente adeguata a fornire buoni risultati. Tuttavia, la sua scarsa scalabilità a dati di grandi dimensioni lo rende un algoritmo poco efficiente e facilmente impraticabile. Nel nostro caso sono stati utilizzati all'incirca 27000 esempi per l'addestramento. In casi reali, per produrre una soluzione di machine learning sufficientemente addestrata, il numero di esempi dovrebbe essere molto più grande, nonostante il conseguente aumento di complessità per un modello di regressione lineare di arrivare a convergenza.
In conclusione, i modelli lineari (Linear Support Vector e SGD) sono risultati assolutamente i migliori: non solo i punteggi ottenuti sono di gran lunga superiori agli altri tipi di modelli, ma anche i tempi di addestramento, seppur non i più bassi, sono stati sufficientemente contenuti. Inoltre, sono stati in grado di adattarsi in modo ottimale ad un dataset piuttosto sbilanciato e fornire predizioni abbastanza accurate per ogni categoria. La scalabilità verso un quantitativo di dati maggiore, dalle documentazioni riportate dalla libreria Sklearn, l'algoritmo SGD dovrebbe essere progettato e ottimizzato appositamente per lavorare con grandi mole di dati e, per tale motivo, è preferibile rispetto al Linear Support Vector.
Si tenga presente che tutti i modelli hanno avuto chiare difficoltà nel distinguere accuratamente categorie con elementi simili, come per "Mystery & Detective" – "Thriller" o "Religion" – "History". Osservando i relativi wordcloud ci si accorge facilmente di parole chiave comuni che potrebbero indurre in confusione. Al fine di rendere più netta la divisione tra le categorie, e dunque ottenere dai modelli risultati migliori, si potrebbe pensare di accorpare categorie con elementi simili (com'è già stato fatto per altre categorie precedentemente presenti nel dataset). Tuttavia, si rischierebbe di avere un set di categorie eccessivamente generiche e, di conseguenza, poco utili nella maggior parte dei contesti pratici.

5 Clustering

Nel contesto dell'analisi e dell'organizzazione di un vasto insieme di libri provenienti da un dataset contenente descrizioni e informazioni sugli autori, il clustering, basato sulle regole dell'apprendimento non supervisionato, si presenta, in linea teorica, come un buon approccio alternativo alla classificazione. Questa metodologia ci offre la possibilità di organizzare i libri in gruppi sfruttando contemporaneamente le somiglianze semantiche nelle descrizioni e le ricorrenze degli autori per ogni cluster, così da creare connessioni significative tra opere affini. Conoscendo a priori il numero di categorie differenti presenti nel dataset (16 categorie), si è adottato un approccio che coinvolge algoritmi di clustering in cui il numero di gruppi risultanti sia preimpostato come iperparametro. Questa scelta ci ha permesso di effettuare anche un confronto più accurato con gli algoritmi di classificazione. Per ogni modello selezionato sono state calcolate e riportare le seguenti metriche:

- Silhouette score
- Tempo necessario per l'addestramento
- Top 10 keywords più frequenti contenute nelle descrizioni degli elementi di ogni cluster (vedi immagine sotto)
- Top 10 autori più frequenti contenuti negli elementi di ogni cluster (vedi immagine sotto)

```python
def get_top_keywords(X, transformer, clusters, n_terms):
    """
    Questa funzione restituisce le keyword per ogni centroide del KMeans

     Parametri:
        - X, colonna del dataframe da cui estrarre le keywords
        - transformer, TfidfVectorizer addestrato sulla colonna X
        - clusters, etichette identificative dei clusters
        - n_terms, numero delle stringhe principali da visualizzare per ogni cluster
    """

    X_tran = transformer.transform(X)
    data = pd.DataFrame(X_tran.todense()).groupby(clusters).mean()  # raggruppa il vettore TF-IDF per gruppo
    terms = transformer.get_feature_names_out()  # accedi ai termini del tf idf

    for i, r in data.iterrows():
        print('\nCluster {}'.format(i))
        print(','.join([terms[t] for t in np.argsort(r)[-n_terms:]]))  # per ogni riga del dataframe, trova gli n termini che hanno il punteggio più alto
```

5.1 Algoritmo K-Means

Il K-Means è uno degli algoritmi di clustering più famoso. Il suo obiettivo è quello di raggruppare i dati allo scopo di separare i campioni in K gruppi di uguale varianza, minimizzando un criterio noto come "somma dei quadrati" all'interno del cluster. Possiamo descrivere questo procedimento con i seguenti step:

1) Scegliere in modo casuale K punti nel dataset come centroidi iniziali;
2) Generare un partizionamento assegnando ogni campione al centroide più vicino;
3) Calcolare i nuovi centroidi del cluster considerando la media dei valori del cluster generato al punto 2;
4) Ripetere i passi 2 e 3 fino a quando i centroidi non cambino.

Il suo addestramento ha impiegato all'incirca 17.99 secondi, ottenendo un Silhouette score di 0.006. Di seguito ne si riportano le top 10 keywords più frequenti contenute nelle descrizioni e i top 10 autori più frequenti.

```
Cluster 0
fat,fitness,eating,body,program,exercise,food,weight,health,diet

Cluster 1
internet,window,system,booknewscom,copyright,web,news,inc,portland,annotation

Cluster 2
meat,soup,dish,vegetable,describes,provides,appetizer,instruction,discus,recipe

Cluster 3
history,killer,case,lindsay,war,story,alex,cross,murder,james

Cluster 4
novel,original,secret,story,young,murder,family,man,woman,love
```

Figure 7: Top 10 keywords più frequenti contenute nelle descrizioni degli elementi di ogni cluster

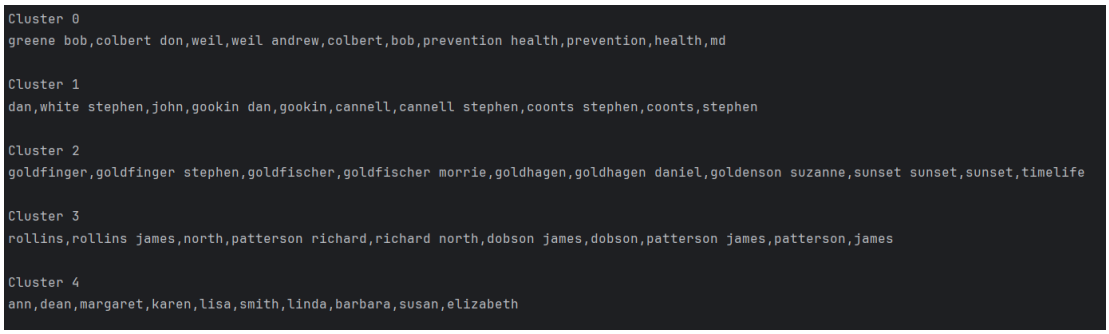


Figure 8: Top 10 autori più frequenti contenuti negli elementi di ogni cluster

A questo punto possiamo osservare che, avendo ottenuto un Silhouette score molto vicino allo 0, i cluster ricavati al termine sono sovrapposti. Infatti, anche andando ad analizzare le keywords ottenute, non è molto chiaro per tutti i cluster che genere questi rappresentino. Questi risultati poco incoraggianti ci lasciano come scelta solo quella di analizzare come si comporta un modello di Mini-Batch K-Means in termini di tempo di addestramento (poiché non ci potrà permettere di ottenere risultati migliori) e successivamente affidarci all'abilità di gestire strutture complesse e non lineari nei dati dello Spectral Clustering.

5.2 Algoritmo MiniBatchKMeans

Il Mini-Batch K-Means è una variante dell'algoritmo K-Means standard che utilizza minibatch di campioni, ovvero sottoinsiemi di dati di input campionati casualmente in ogni iterazione di training per aggiornare i centroidi, invece di utilizzare l'intero set di dati. Questo approccio rende Mini-Batch K-Means più efficiente dal punto di vista computazionale permettendo di lavorare su dataset più grandi rispetto al K-Means tradizionale. A differenza di altri algoritmi che riducono il tempo di convergenza, il Mini-Batch K-Means produce risultati che generalmente sono solo leggermente peggiori rispetto all'algoritmo standard. Possiamo descrivere questo algoritmo con i seguenti step:

1) Scegliere in modo casuale K punti nel dataset come centroidi iniziali;
2) Ad ogni iterazione, selezionare casualmente un minibatch di dimensione prefissata dal dataset;
3) Generare un partizionamento assegnando ogni campione del minibatch al centroide più vicino;
4) Aggiornare i centroidi usando la media dei punti nel minibatch corrente;
5) Ripetere i passi da 2 a 4 fino a quando i centroidi non cambiano.

Il suo addestramento ha impiegato all'incirca 3.17 secondi, ottenendo un Silhouette score di 0.005. Di seguito ne si riportano le top 10 keywords più frequenti contenute nelle descrizioni e i top 10 autori più frequenti.

```
Cluster 0
reader,people,author,year,first,american,war,history,story,god

Cluster 1
web,portland,annotation,inc,management,edition,offer,information,business,guide

Cluster 2
meal,kitchen,instruction,cookbook,home,tip,wine,cooking,food,recipe

Cluster 3
bread,pasta,meat,appetizer,vegetable,salad,dessert,soup,dish,recipe

Cluster 4
find,original,novel,secret,young,murder,family,man,woman,love
```

Figure 9: Top 10 keywords più frequenti contenute nelle descrizioni degli elementi di ogni cluster

```
Cluster 0
george,bill,charles,paul,thomas,william,peter,richard,michael,david

Cluster 1
mark,inc,stephen,thomas,paul,peter,william,richard,michael,david

Cluster 2
living,digest association,mary,martha,of,readers,readers digest,digest,association,sunset

Cluster 3
good,phyllis,chuck,williams chuck,williams,gourmet,sunset,crocker betty,betty,crocker

Cluster 4
kellerman,smith,janet,steel,steel danielle,lisa,danielle,linda,susan,elizabeth
```

Figure 10: Top 10 autori più frequenti contenuti negli elementi di ogni cluster

È possibile notare come questo modello sia nettamente più rapido nell'addestramento rispetto alla versione standard del K-Means (quasi 15 secondi di differenza) ma, come già previsto dalla teoria, il Silhouette score non si abbassa drasticamente.

Il fine dell'implementazione di questo algoritmo è più sperimentale che di utilità. È da considerare che in una possibile futura analisi su un dataset, con le stesse features ma con molti più dati, l'algoritmo Mini-Batch K-Means potrebbe quasi del tutto sostituire K-Means e, talvolta, essere preferibile ad esso.

5.3 Algoritmo SpectralClustering

Lo Spectral Clustering è un algoritmo di clustering che si basa sulla teoria spettrale, ovvero una branca matematica legata allo studio delle proprietà degli spettri degli operatori lineari. Quest'approccio offre un modo potente per affrontare strutture complesse e non lineari nei dati, le quali non possono essere gestite adeguatamente da metodi più tradizionali come K-Means. Il processo di funzionamento può essere suddiviso in diverse fasi:

1) Utilizzando la similarità tra i punti del dataset (calcolata ad esempio utilizzando la distanza euclidea tra i punti), viene costruito un grafo;
2) Sfruttando la teoria spettrale, il grafo viene trasformato in uno spazio di dimensioni inferiori attraverso la procedura degli autovettori e degli autovalori;
3) Una volta ottenuto lo spazio trasformato, viene applicato un algoritmo di clustering, come ad esempio K-Means, per raggruppare i punti in cluster;
4) I risultati dell'algoritmo di clustering vengono utilizzati per assegnare ciascun punto del dataset a un cluster specifico. La coerenza e la separazione dei cluster dipendono dalla capacità della trasformazione spettrale di rivelare pattern significativi nei dati.

Il suo addestramento ha impiegato all'incirca 273.08 secondi, ottenendo un Silhouette score di 0.005.

Di seguito ne si riportano le top 10 keywords più frequenti contenute nelle descrizioni e i top 10 autori più frequenti.

```
Cluster 0
love,year,make,author,offer,story,first,woman,guide,recipe

Cluster 1
bible,reading,answer,scripture,prayer,journaling,jesus,max,lucado,god

Cluster 2
and,suspense,terrifying,mystery,novel,man,night,odd,dean,koontz

Cluster 3
look,history,provides,technique,dish,examines,culture,instruction,discus,recipe

Cluster 4
earl,catherine,lacey,dillon,coulter,fbi,sherbrooke,agent,sherlock,savich
```

Figure 11: Top 10 keywords più frequenti contenute nelle descrizioni degli elementi di ogni cluster

```
Cluster 0
elizabeth,peter,william,susan,richard,michael,robert,james,david,john

Cluster 1
goldfinger,goldfinger stephen,goldfischer,goldfischer morrie,goldhagen,goldenson suzanne,zyrianova julia,max,lucado max,lucado

Cluster 2
goldenson suzanne,goldfine,goldfine allison,goldfinger,goldfinger stephen,goldfischer,goldhagen,dean,koontz,koontz dean

Cluster 3
goldfine allison,goldfinger,goldfinger stephen,goldfischer,goldfischer morrie,goldhagen,goldhagen daniel,golden christopher,zyrianova julia,timelife

Cluster 4
goldfischer,goldfischer morrie,goldhagen,goldhagen daniel,goldhammer,goldfine,zyrianova julia,catherine,coulter,coulter catherine
```

Figure 12: Top 10 autori più frequenti contenuti negli elementi di ogni cluster

Tra tutti gli algoritmi di clustering provati, questo è stato decisamente il più lento, anche con un margine piuttosto evidente. C'era da aspettarselo poiché, oltre che eseguire il K-Means, lo Spectral Clustering richiede ulteriori operazioni preliminari di costruzione di un grafo e trasformazione di esso in uno spazio di dimensioni inferiori. Nonostante il tempo molto disteso per l'addestramento, il Silhouette score ottenuto risulta essere addirittura peggiore di quello ricavato dal K-Means (seppur di poco). Questa valutazione ci permette di capire che l'algoritmo non è stato in grado di rivelare dei pattern significativi nei dati, difficoltà, evidentemente, introdotta dai vari problemi previsti nella gestione del linguaggio naturale.

5.4 Valutazioni finali sul clustering

Dallo studio condotto sui vari modelli di clustering si nota come per ognuno di essi i risultati ottenuti non li rendono effettivamente utilizzabili nella pratica. La media dei Silhouette score ottenuti si aggira intorno allo zero e le keywords estratte dalle singole analisi non sono funzionali al riconoscimento della categoria di appartenenza del libro. Nonostante ciò, l'algoritmo che ha portato a risultati migliori, seppur di poco, è stato il K-Means, mentre il più veloce nell'addestramento è stato il Mini-Batch K-Means, il che lo rende anche il più propenso a essere adattato a dataset con un numero molto maggiore di dati. Lo Spectral Clustering, invece, non ci ha permesso di giungere a risultati di qualità superiore a causa della rappresentazione stessa dei dati e perchè si tratta di una tecnica tipicamente non utilizzata per dati testuali, ma per dati numerici o geometrici, come quelli provenienti da immagini.

6 Implementazione script per l'addestramento

Com'è già stato specificato precedentemente, al fine di poter addestrare un modello di machine learnig su dati testuali è stato necessario far ricorso a conteggi in frequenze per convertire le parole in sequenze di valori numerici utilizzabili da algoritmi di apprendimento. Per far ciò è stato necessario ricorrere alla classe 'TfidfVectorizer', la quale è autonomamente in grado di estrarre i token costituenti i periodi testuali, contarli e normalizzare i conteggi secondo la tecnica tf-idf (term frequency–inverse document frequency). L'output fornito da tale processo è una matrice sparsa di valori numerici costituita da tante colonne quante sono le differenti parole identificate e tante righe quanti sono gli esempi. Per le descrizioni si è deciso di considerare unicamente token costituiti da parole singole, mentre, per gli autori, si è pensato potesse tornare utile considerare anche token costituiti da due parole (quelle che dovrebbero rappresentare nome e cognome dell'autore).

```
description_vect = TfidfVectorizer()
authors_vect = TfidfVectorizer(ngram_range=(1, 2))
colTransformer = ColumnTransformer([
    ('des_transformer', description_vect, "Description"),
    ('aut_transformer', authors_vect, "Authors")
])
```

Successivamente, lo script costruisce una pipeline contenente i “vettorizzatori” (istanze di TfidfVectorizer’) per le descrizioni e per gli autori e il modello il cui nome viene specificato da uno dei parametri in input.

```
match model:    # A seconda del modello scelto, instanzio l'opportuna pipeline
    case "LogisticRegression":
        modelPipe = Pipeline([('transformer', colTransformer), ('clf', LogisticRegression(max_iter=2000))])

    case "SGDClassifier":
        modelPipe = Pipeline([('transformer', colTransformer), ('clf', SGDClassifier(max_iter=2000))])

    case "MultinomialNB":
        modelPipe = Pipeline([('transformer', colTransformer), ('clf', MultinomialNB())])

    case "LinearSVC":
        modelPipe = Pipeline([('transformer', colTransformer), ('clf', LinearSVC(dual="auto", max_iter=2000))])

    case "ComplementNB":
        modelPipe = Pipeline([('transformer', colTransformer), ('clf', ComplementNB())])

    case _:
        raise ValueError("Modello non riconosciuto o implementato")
```

Figure 13: Pipeline dei classificatori

```
if model == "KMeans":
    modelPipe = Pipeline([('transformer', colTransformer), ('clt', KMeans(n_clusters = K_clusters, random_state = 42, n_init = 5))])
elif model == "MiniBatchKMeans":
    modelPipe = Pipeline([('transformer', colTransformer), ('clt', MiniBatchKMeans(n_clusters=K_clusters, random_state=42, n_init=5))])
elif model == "SpectralClustering":
    modelPipe = Pipeline([('transformer', colTransformer), ('clt', SpectralClustering(n_clusters=K_clusters, random_state=42, n_init=5))])
else:
    raise ValueError("Inserisci il nome di un modello valido, a scelta tra: KMeans, MiniBatchKMeans e SpectralClustering")
```

Figure 14: Pipeline dei cluster

In seguito, si passa all’addestramento del modello. Se specificato da uno dei parametri in input, si effettua una ricerca dei migliori parametri di configurazione per il modello (vedere paragrafo 7), portando ad un addestramento molto più lungo, altrimenti si addestra il modello con una configurazione di default. Infine, se specificato, il modello viene serializzato e memorizzato in un file binario per poter essere, successivamente, recuperato e utilizzato per effettuare predizioni.

```python
if findBestEstimator:   #Se findBestEstimator = True, ricerco il "miglior" modello
    params = getModelParams(model)
    rs = RandomizedSearchCV(modelPipe, params, cv=5, n_jobs=-1)

    start_time = time()
    rs.fit(X_train, Y_train)
    fitTime = time() - start_time

    trainedModel = rs.best_estimator_
else:       # Altrimenti addesstro il modello con la configurazione di default
    startTime = time()
    modelPipe.fit(X_train, Y_train)
    fitTime = time() - startTime

    trainedModel = modelPipe

if saveEstimator:       # Se saveEstimator = True, serializzo il classificatore all'interno di un apposito file
    dump(trainedModel, filename: f"Models/{saveEstimator}.joblib")
    print(f"Classificatore {model} serializzato all'interno del seguente file: /Models/{saveEstimator}.joblib")
```

Figure 15: Addestramento Classificazione

```python
if findBestEstimator:  # Se findBestEstimator = True, ricerco il "miglior" modello
    params = getModelParams(model)
    rs = RandomizedSearchCV(modelPipe, params, scoring=cv_silhouette_scorer, cv=[(slice(None), slice(None))], n_jobs=-1)
    start_time = time()
    rs.fit(X)
    fitTime = time() - start_time

    trainedModel = rs.best_estimator_
else:  # Altrimenti addesstro il modello con la configurazione di default
    startTime = time()
    modelPipe.fit(X)
    fitTime = time() - startTime

    trainedModel = modelPipe

if saveEstimator:  # Se saveEstimator = True, serializzo il classificatore all'interno di un apposito file
    dump(trainedModel, filename: f"Models/{saveEstimator}.joblib")
    print(f"Clusterer {model} serializzato all'interno del seguente file: /Models/{saveEstimator}.joblib")
```

Figure 16: Addestramento Clustering

7 Ottimizzazione modelli

Com'è noto, ogni tipo di modello, che sia di classificazione o clustering, possiede una serie di parametri. Questi parametri sono noti come iperparametri e, se opportunamente configurati, permettono di addestrare il modello in maniera ottimale e di adattarsi meglio ai dati a disposizione. Ogni modello possiede parametri specifici e la regolazione di essi risulta particolarmente complessa e richiederebbe conoscenze specifiche sul funzionamento dell'algoritmo di addestramento di cui fa uso. Anche assumendo di essere a conoscenza del funzionamento di ciascun algoritmo, risulta comunque complesso e dispendioso dal punto di vista temporale determinare "a mano" i valori per ciascuno dei parametri a disposizione. Per tale motivo, tipicamente si sceglie di addestrare e valutare lo stesso modello, sugli stessi dati, ma con differenti configurazioni dei parametri, determinando poi quale di queste ha conseguito il risultato migliore. La libreria Sklearn mette a disposizione degli strumenti per automatizzare tale processo: GridSearchCV e RandomizedSearchCV. Entrambe le classi operano nello stesso modo: si indica il modello che si intende addestrare e si specifica una lista dei parametri interessati al "tuning" e dei possibili valori da assegnargli e automaticamente essi addestreranno e valuteranno il modello indicato con differenti permutazioni dei valori specificati per i vari parametri. Nello specifico, ciascun modello viene valutato con un approccio Cross – Validation, per aumentare l'affidabilità della valutazione stessa. La differenza sostanziale tra GridSearch e RandomizedSearch è che il primo addestra e valuta tutti i modelli con tutte le possibili permutazioni di configurazioni specificate mentre il secondo valuta casualmente soltanto un numero predefinito di queste. Ovviamente il primo approccio è esaustivo e permette di identificare con certezza il miglior modello possibile, a discapito di un'enorme quantità di tempo necessaria per la ricerca. Il secondo, invece, è una ricerca meno esaustiva che punta sulla "fortuna della casualità", ma permette di ottenere dei risultati mediamente migliori di una configurazione dei parametri di default e in tempi molto più brevi. Ai fini del nostro progetto, e per le risorse limitate a noi a disposizione, abbiamo comunque deciso di provare ad ottimizzare i nostri modelli ma tramite l'ausilio della RandomizedSearch. C'è da specificare che l'identificazione del miglior modello deve essere fatta sulla base di un punteggio che permetta di determinare quando un modello è migliore di un altro. Nello specifico, per i modelli di classificazione ci si è basati sull'Accuracy, mentre per i modelli di clustering si è usato il Silhouette score. Si tenga conto che un approccio di valutazione Cross Validation, per modelli di clustering, non ha molta utilità e per tale motivo questo tipo di valutazione è stata usata solo per modelli di classificazione.

```
case "LogisticRegression":
    params ={"transformer__aut_transformer__ngram_range": [(1, 1), (1, 2), (2, 2)],
             "clf__solver": ["sag", "saga", "lbfgs", "newton-cg"],
             "clf__C": [1, 10, 100, 1000],
             "clf__tol": [0.001, 0.0001, 0.00001]
             }

case "SGDClassifier":
    params ={"transformer__aut_transformer__ngram_range": [(1, 1), (1, 2), (2, 2)],
             "clf__loss": ["hinge", "log_loss", "squared_hinge"],
             "clf__penalty": ["l2", "l1", "elasticnet"],
             "clf__alpha": [0.001, 0.0001, 0.00001],
             "clf__tol": [0.001, 0.0001, 0.00001]
             }

case "MultinomialNB":
    params ={"transformer__aut_transformer__ngram_range": [(1, 1), (1, 2), (2, 2)]
             }
```

Figure 17: Parametri e valori considerati per l'ottimizzazione di alcuni dei modelli di classificazione

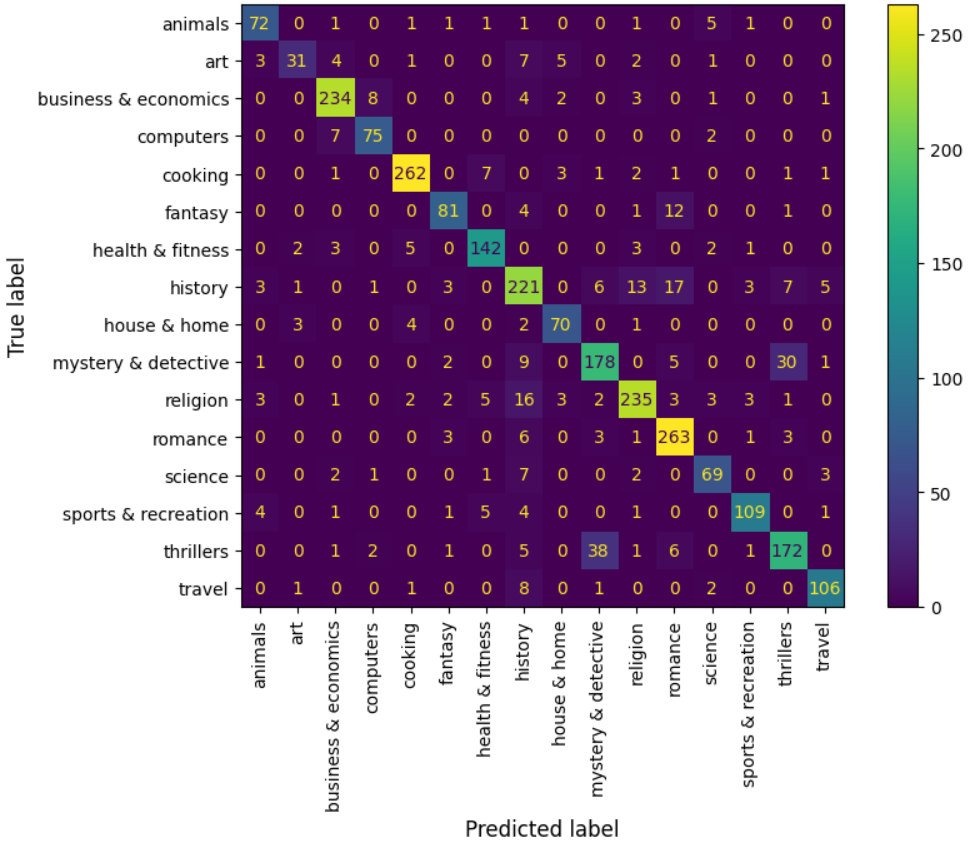
```
case "KMeans":
    params = {"transformer__aut_transformer__ngram_range": [(1, 1), (1, 2), (2, 2)],
              "clt__init": ["k-means++", "random"],
              "clt__tol": [0.001, 0.0001, 0.00001],
              "clt__algorithm": ["lloyd", "elkan"]
              }

case "MiniBatchKMeans":
    params = {"transformer__aut_transformer__ngram_range": [(1, 1), (1, 2), (2, 2)],
              "clt__init": ["k-means++", "random"],
              "clt__batch_size": [256, 512, 1024, 2048],
              "clt__tol": [0.001, 0.0001, 0.00001],
              "clt__max_no_improvement": [10, 20, 30],
              "clt__init_size": [3072, 3500, 4000],
              "clt__reassignment_ratio": [0.01, 0.05, 0.1]
              }
```

Figure 18: Parametri e valori considerati per l'ottimizzazione di alcuni dei modelli di clustering

Si riportano i risultati ottenuti dall'ottimizzazione di un modello di classificazione e uno di clustering. Per la classificazione, si è tentata l'ottimizzazione del modello Linear Support Vector: la ricerca ha impiegato all'incirca 1130.68 secondi ottenendo un punteggio di accuracy di 0.855. Di seguito ne si riporta il suo classification report e la matrice di confusione delle predizioni:

	precision	recall	f1-score	support
animals	0.84	0.86	0.85	84
art	0.82	0.57	0.67	54
business & economics	0.92	0.92	0.92	253
computers	0.86	0.89	0.88	84
cooking	0.95	0.94	0.94	279
fantasy	0.86	0.82	0.84	99
health & fitness	0.88	0.90	0.89	158
history	0.75	0.79	0.77	280
house & home	0.84	0.88	0.86	80
mystery & detective	0.78	0.79	0.78	226
religion	0.88	0.84	0.86	279
romance	0.86	0.94	0.90	280
science	0.81	0.81	0.81	85
sports & recreation	0.92	0.87	0.89	126
thrillers	0.80	0.76	0.78	227
travel	0.90	0.89	0.89	119

Matrice di confusione



Per il clustering, si è tentata l'ottimizzazione del modello K-Means: la ricerca ha impiegato all'incirca 236.51 secondi ottenendo un Silhouette score di 0.011.
Di seguito ne si riportano le top 10 keywords più frequenti contenute nelle descrizioni e i top 10 autori più frequenti.

```
Cluster 0
original,find,young,novel,secret,murder,family,man,love,woman

Cluster 1
cookbook,meal,vegetable,salad,soup,dessert,cooking,food,dish,recipe

Cluster 2
help,american,war,author,america,first,history,web,guide,david

Cluster 3
story,american,author,game,war,guide,god,first,history,john

Cluster 4
first,work,year,history,offer,american,business,war,spenser,robert
```

Figure 19: Top 10 keywords più frequenti contenute nelle descrizioni degli elementi di ogni cluster

```
Cluster 0
janet,anne,smith,sandra,lisa,catherine,barbara,brown,susan,elizabeth

Cluster 1
anne,gourmet,stewart,chuck,living,williams,martha,sunset,betty,crocker

Cluster 2
liss,drake,halberstam,gardner,flanagan,morrell,eddings,pogue,baldacci,david

Cluster 3
clifford,mortimer,le,carre,maxwell,grisham,lescroart,feinstein,sandford,john

Cluster 4
harold,schuller,irwin,md,tanenbaum,atkins,barnard,ludlum,parker,robert
```

Figure 20: Top 10 autori più frequenti contenuti negli elementi di ogni cluster

Confrontando i risultati ottenuti dai modelli "ottimizzati" con le rispettive versioni "non ottimizzate" si può notare come, in effetti, essi non siano tanto migliori e, anzi, hanno raggiunto punteggi simili.
L'esito non stupisce considerando che:

- La ricerca RandomizedSearch è un processo casuale che non garantisce l'ottimalità;
- Non avendo conoscenze dettagliate sul funzionamento tecnico degli algoritmi in questione, probabilmente la lista dei parametri indicati e i rispettivi valori specificati potrebbero non essere adatti, o comunque, potrebbero essere specificati valori maggiormente sensati;
- Non è da escludere la possibilità per cui le configurazioni predefinite, usate per le versioni "non ottimizzate" dei modelli, siano già efficaci a ottenere il miglior risultato possibile utilizzando i dati disponibili.

8 Conclusioni

Dopo aver condotto addestramenti approfonditi su ciascun modello di classificazione e clustering, accompagnati da dettagliati report analitici e successiva ottimizzazione del modello più promettente, emergono come rappresentativi della rispettiva categoria il Linear Regression e l'algoritmo K-Means. Nel confronto tra i due modelli, si evidenzia chiaramente la superiorità del Linear Regression rispetto alle prestazioni più limitate del K-Means. Mentre i cluster identificati da K-Means risultano poco significativi e poco coesi rendendo quest'algoritmo inutilizzabile per la nostra applicazione pratica, le analisi condotte sul Linear Regression indicano che questo modello è capace, nella maggior parte dei casi, di assegnare con precisione la categoria di un libro. Un vantaggio notevole è dato anche dal tempo di addestramento, significativamente inferiore rispetto a quello della sua controparte di apprendimento non supervisionato.

Questi risultati sottolineano chiaramente che, nel nostro contesto di categorizzazione dei libri, la metodologia più adatta è la classificazione. Quest'approccio, infatti, offre modelli specificamente progettati per lavorare su dati testuali garantendo una maggiore precisione e pertinenza rispetto al clustering, in cui si utilizzano approcci più generici e meno specifici per la gestione di dati in linguaggio naturale.

9 Contributi

Per definire l'obiettivo del progetto, la scelta del dataset e le operazioni di Data Preparation sono stati svolti incontri di gruppo. Successivamente ogni componente del gruppo ha dato un contributo specifico in determinati ambiti.
Di seguito la tabella dei contributi:

Mancuso Maria Angela	Sviluppo della documentazione
Malfettone Ines	Sviluppo della documentazione
Santonicola Federico	Addestramento e valutazione Classificatori
Sessa Attilio Marco	Addestramento e valutazione Clusterers

Mancuso Maria Angela 0612705832
Malfettone Ines 0612705311
Santonicola Federico 0612706449
Sessa Attilio Marco 0612705403