Sistema di Classificazione di Audiolibri Basato su Conoscenza e Machine Learning

Componenti.: Luca Canonico

MAT.: 708794

Email.: l.canonico@studenti.uniba.it

Ingegneria della Conoscenza A.A. 2023/2024

Link: https://github.com/Andriii14/iCon23-24

Indice

0) Introduzione

Il progetto si propone di applicare tecniche avanzate di apprendimento automatico per la classificazione e raccomandazione di audiolibri, con l'obiettivo di migliorare la scoperta e la selezione personalizzata per gli utenti. Attraverso uno script di web scraping automatizzato, il sistema raccoglie dati completi da Audible, estraendo informazioni dettagliate su audiolibri e podcast. Questi dati vengono poi strutturati in una base di conoscenza logica utilizzando Prolog, consentendo l'esecuzione di query avanzate per filtrare contenuti, generare raccomandazioni personalizzate e gestire le preferenze degli utenti. Il sistema integra modelli di apprendimento supervisionato per classificare gli audiolibri in categorie predefinite e modelli di apprendimento non supervisionato per ridurre la complessità del target attraverso il clustering. Affronta sfide comuni come la gestione dello squilibrio delle classi nei dataset e l'ottimizzazione degli iperparametri, assicurando lo sviluppo di modelli accurati e robusti. Il progetto rappresenta una solida base per l'espansione futura verso un sistema ancora più avanzato e complesso.

Requisiti Funzionali

Il progetto è stato realizzato in Python (3.12.3) in quanto offre un'ampia gamma di librerie specifiche per il machine learning che semplificano lo sviluppo e la manipolazione dei dati.

Installazione e Avvio

Aprire il file del progetto utilizzando l'ambiente di sviluppo preferito ed eseguire il programma attraverso il main. Questo file contiene il codice principale che si occupa di gestire il flusso completo di operazioni.

Note Importanti:

- **Dipendenze:** Prima di eseguire il programma, assicurarsi di aver installato tutte le librerie Python elencate nel file requirements.txt, è possibile sfruttare la funzione $installPackages()$.
- **Istruzioni commentate:** Alcune sezioni del codice potrebbero essere commentate, in quanto rappresentano operazioni che devono essere eseguite solo una volta o in determinate situazioni. Prestare attenzione a queste sezioni e scommentarle se necessario.

Librerie Utilizzate

- **pandas:** gestione e manipolazione di dataset in formato tabellare.
- **numPy:** operazioni matematiche e la gestione di array.
- **scikit-learn:** algoritmi di machine learning e strumenti di valutazione.
- **matplotlib**: creazione e visualizzazione di grafici.
- **selenium**: automazione del browser e il web scraping.
- **nltk:** elaborazione del linguaggio naturale, come stop words e stemming.
- **spaCy:** Analisi linguistica avanzata.
- **imblearn**: bilanciamento dei dataset con tecniche come l'oversampling.
- **pyswip**: interfacciamento con il motore Prolog.
- **re**: gestione di espressioni regolari e la manipolazione di stringhe.
- **logging:** gestione e registrazione dei messaggi di log.
- **joblib:** salvataggio e il caricamento di modelli e oggetti Python.
- **scipy**: funzioni statistiche e distribuzioni.
- **os**: permette di interagire con il file system.
- **time**: gestione delle pause temporali e i tempi di attesa.
- **csv**: per la lettura e scrittura di file CSV.
- **kneed**: identificazione del "gomito" nelle curve di clustering.
- **importlib**: utilizzata per l'importazione dinamica di moduli Python.
- **tqdm**: visualizzazione di barre di progresso durante l'esecuzione di operazioni iterabili.
- **sys**: permette di interagire con l'ambiente del sistema operativo.
- **subprocess**: consente l'esecuzione di comandi del sistema operativo e la gestione dei processi.

1) Creazione del Dataset

Le scelte adottate riguardo la raccolta dei dati sono state guidate dalla mancanza di dataset italiani disponibili sul tema, portando allo sviluppo di uno script di web scraping per l'estrazione automatica delle informazioni necessarie. Il dataset è stato quindi generato tramite un processo di raccolta automatizzata di dati dalle pagine web di Audible, utilizzando la libreria $Selenium$ di Python. Lo script automatizzato avvia un browser Chrome e simula le azioni di un utente reale, come l'accesso al sito con credenziali predefinite. Dopo il login, esplora le diverse categorie di audiolibri presenti sulla piattaforma, scorrendo dinamicamente le pagine e caricando i risultati. Per ogni categoria, lo script raccoglie informazioni chiave. È inoltre in grado di distinguere tra audiolibri e podcast, adattando l'estrazione in base al tipo di contenuto. Viene gestita anche la paginazione, consentendo allo script di navigare tra più pagine di risultati e raccogliere dati da ciascun audiolibro o podcast trovato. Un ulteriore aspetto rilevante è la capacità di estrarre recensioni degli utenti, arricchendo il dataset. Al termine del processo, i dati vengono organizzati e salvati, pronti per essere utilizzati nelle fasi successive. Lo script include meccanismi di gestione degli errori e delle eccezioni, garantendo la continuità dell'estrazione anche in caso di pagine web non standard o elementi mancanti. Inoltre, il processo di scraping è stato condotto in modo responsabile, rispettando i termini d'uso di Audible e prevenendo sovraccarichi ai server. Il dataset risultante ha fornito una solida base per l'addestramento dei modelli di classificazione.

	summary	category	subcategory	tags
00097	In una galassia percorsa da una forza vitale chiamata "corrente...	Adolescenti e Ragazzi	Fantascienza e fantasy	Distopica, Fantas...
00098	Il romanzo di formazione al femminile per eccellenza. Qualsiasi...	Adolescenti e Ragazzi	Letteratura e narrativa	Letteratura e nar...
00099	È l'alba della battaglia decisiva tra le rovine della città di ...	Adolescenti e Ragazzi	Fantascienza e fantasy	Fantasy
00100	Ascoltando... s'impara! Anche la letteratura, con il Prof! In q...	Adolescenti e Ragazzi	Letteratura e narrativa	Adolescenti e Rag...
00101	La vita di Paul Tantum è cambiata. Ha compiuto quattordici anni...	Adolescenti e Ragazzi	Fantascienza e fantasy	Paranormale, Lett...
00102	Cosa succede quando ti accorgi, per la prima volta, che ti piac...	Adolescenti e Ragazzi	Letteratura e narrativa	Letteratura e nar...
00103	Dall'autrice di bestseller numero uno, Morgan Rice, e autrice d...	Adolescenti e Ragazzi	Fantascienza e fantasy	Fantasy, Azione e...
00104	Il romanzo Il giardino segreto, pubblicato nel 1909, fu riscope...	Adolescenti e Ragazzi	Letteratura e narrativa	Situazioni diffic...
00105	C'è potere nella parola. Avere quindici anni non è mai facile, ...	Adolescenti e Ragazzi	Letteratura e narrativa	Situazioni diffic...
00106	1 giugno 1815. Emanuele non immagina che da quel giorno la sua ...	Adolescenti e Ragazzi	Letteratura e narrativa	Situazioni diffic...
00107	Esiste, in una dimensione parallela, un mondo chiamato Liiandri...	Adolescenti e Ragazzi	Fantascienza e fantasy	Fantasy
00108	Nel mondo di Erdas, quattro ragazzi partecipano a una cerimonia...	Adolescenti e Ragazzi	Fantascienza e fantasy	Fantasy, Paranorm...
00109	"Non sarà sempre così" dissi a me stessa. "Più tempo passerai d...	Adolescenti e Ragazzi	Fantascienza e fantasy	Fantasy
00110	La contessina Ines di Ventimiglia è a Panama per ritirare l'ere...	Adolescenti e Ragazzi	Letteratura e narrativa	Azione e avventur...
00111	"Sai perché mi scrivo sul braccio tutti i giorni quelle parole,...	Adolescenti e Ragazzi	Romanzo d'amore	Contemporaneo
00112	Una strega e un cacciatore di streghe legati nel sacro vincolo ...	Adolescenti e Ragazzi	Fantascienza e fantasy	Fantasy, Paranorm...
00113	Passione, tradimento, inganni mortali: l'esplosivo finale della...	Adolescenti e Ragazzi	Romanzo d'amore	Romanzo d'amore
00114	Norby è un robottino davvero unico. E non solo perché ha l'aspe...	Adolescenti e Ragazzi	Fantascienza e fantasy	Fantascienza
00115	Nikolai Lantsov, sovrano di Ravka, corsaro, soldato, secondogen...	Adolescenti e Ragazzi	Letteratura e narrativa	Letteratura e nar...

2) Preprocessing

Il preprocessing del dataset è stato suddiviso in due fasi distinte, ciascuna progettata per ottimizzare i dati in modo specifico. La prima fase, eseguita subito dopo l'estrazione dei dati attraverso la chiamata alla funzione $clean\_data()$, è focalizzata sulla pulizia e normalizzazione del dataset. Questa fase ha l'obiettivo di eliminare valori mancanti e duplicati, poiché la loro presenza può distorcere i risultati e influire negativamente sull'addestramento dei modelli. Inoltre, vengono filtrati gli elementi con poche informazioni rilevanti, poiché categorie scarsamente rappresentate non solo aggiungono rumore, ma possono anche introdurre squilibri nei modelli. Assicurarsi che i dati siano coerenti e correttamente formattati è cruciale per evitare errori nelle fasi successive. Alla fine di questa fase, il dataset viene salvato in un nuovo file $audible\_italiano\_cleaned.csv$.

La seconda fase prevede l'applicazione di tecniche avanzate di preprocessing del testo con l'obiettivo di migliorare l'accuratezza e le prestazioni degli algoritmi di apprendimento automatico. La chiamata avviene attraverso la funzione $preprocess\_data()$ che permette una certa personalizzazione attraverso i parametri di input.

```
Args:
    input_file (str): Nome del file di input (output di clean_data). Default: 'audible_italiano_cleaned.csv'
    output_file (str): Nome del file di output. Default: 'preprocessed_data.csv'
    columns_to_preprocess (list): Lista delle colonne a cui applicare rimozione stop words e stemming.
    columns_to_exclude (list): Lista delle colonne da escludere da qualsiasi preprocessing.
    save_output (bool): Se True, salva il file preprocessed_data.csv. Default: False.
"""
```

Durante questa fase, il testo viene prima normalizzato, rimuovendo punteggiatura, spazi bianchi extra e convertendo tutto in minuscolo, così da garantire uniformità tra i dati ed evitare che piccole differenze di formattazione possano influenzare i risultati. Successivamente, viene eseguita la rimozione delle **stop words**, eliminando le parole più comuni e poco informative che non apportano valore all'analisi semantica. Infine, viene applicato lo **stemming**, che riduce le parole alla loro radice, così da gestire le diverse forme grammaticali senza perdere il significato sottostante.

Durante il processo di sviluppo, sono state testate anche altre tecniche di preprocessing del testo, come la **lemmatizzazione**. Tuttavia, la combinazione di rimozione delle stop words e stemming si è dimostrata la più efficace in termini di rapporto efficienza-prestazioni. Questo equilibrio tra qualità dei dati e semplicità del processo ha garantito risultati migliori rispetto a tecniche più complesse, che aumentavano notevolmente i tempi di elaborazione senza un miglioramento significativo delle prestazioni dei modelli.

3) Prolog

Prolog è stato scelto per la sua capacità di gestire inferenze logiche. Il **ragionamento logico** si fonda sulla logica matematica, dove le conclusioni vengono derivate attraverso verità certe. Viene costruita una **Knowledge Base** (base di conoscenza) composta da assiomi, che sono affermazioni sempre vere all'interno del sistema. Gli assiomi si dividono in **fatti** e **regole**. I fatti rappresentano verità immutabili. Le regole, invece, definiscono relazioni tra i fatti e affermano che qualcosa è vero in base alla veridicità di altre condizioni.

Nel progetto, ho sviluppato una **base di conoscenza** per la gestione delle informazioni relative agli audiolibri, utilizzando Prolog come linguaggio di programmazione logica. La base di conoscenza è stata progettata per rappresentare in modo strutturato gli attributi principali degli audiolibri. Ogni informazione viene memorizzata come un fatto, permettendo una gestione organizzata dei dati e facilitando le interrogazioni logiche. Questo consente di filtrare attraverso interrogazioni su specifici attributi degli audiolibri. Nel mio caso, ho definito tre tipi di fatti e diverse regole logiche per implementare il comportamento logico desiderato.

- Fatto **audiobook**: contiene tutte le informazioni rilevanti relative a un audiolibro.

```
audiobook(
  'Notti bianche',
  ' ',
  ' ',
  'Fedor Dostoevskij',
  'Fabrizio Bentivoglio',
  136,
  4.5,
  1740,
  'Nell\'incanto evanescente delle notti pietroburghesi tra maggio e giugno,
  'Letteratura e narrativa',
  'Classici',
  ['Classici', 'Narrativa storica', 'Storico', 'Urbana', 'Vita in città'],
  'https://www.audible.it/pd/Notti-bianche-Audiolibri/B01BYlPYD6?qid=1720969
  'false',
  'Emons Edizioni'
).
```

- Fatto **utente**: memorizza il nome di un utente, consentendo di associare uno o più titoli preferiti a ciascun utente.
- Fatto **preferito**: associa a ogni utente un titolo di audiolibro presente nella base di conoscenza, registrando le preferenze personali.

A partire da questi fatti, ho implementato diverse regole. Le regole semplici permettono di filtrare gli audiolibri in base a criteri specifici. Inoltre, è stata sviluppata una regola più complessa che si basa sulle regole precedenti per mostra un comportamento logico. Questa regola consente di generare raccomandazioni basate su una serie di parametri comuni tra l'audiolibro di riferimento e quelli simili. Può essere adattata alle preferenze personali.

```
%% Regola per trovare audiolibri simili basata su più criteri
%% (modificare i criteri in base alle preferenze)
audiolibri_simili(Title, SimilarTitle) :-
    % Recupera i dettagli del titolo di riferimento
    written_by(Title, Author),
    belongs_to_category(Title, Category),
    belongs_to_subcategory(Title, Subcategory),
    %% narrated_by(Title, Narrator),
    published_by(Title, Publisher),
    has_podcast_type(Title, PodcastType),
    audiobook(Title, _, _, _, _, _, _, _, _, _, Tags, _, _, _),

    % Trova i dettagli per il titolo simile
    written_by(SimilarTitle, Author),
    belongs_to_category(SimilarTitle, Category),
    belongs_to_subcategory(SimilarTitle, Subcategory),
    %% narrated_by(SimilarTitle, Narrator),
    published_by(SimilarTitle, Publisher),
    has_podcast_type(SimilarTitle, PodcastType),
    audiobook_by_tag(SharedTag, SimilarTitle),
    memberchk(SharedTag, Tags),

    % Assicurati che non sia lo stesso titolo
    SimilarTitle \= Title,
```

È stata implementata una regola che permette di selezionare casualmente titoli di audiolibri, rendendo più rapido e intuitivo il processo di selezione dei preferiti da parte dell'utente. Questa funzionalità consente di suggerire automaticamente nuovi titoli da aggiungere alle preferenze, eliminando la necessità di una selezione manuale. Un aspetto chiave del sistema è la gestione delle preferenze degli utenti. Quando un utente seleziona un audiolibro come preferito, il titolo viene memorizzato come fatto all'interno della base di conoscenza. Una volta salvati, questi titoli vengono esclusi dalle raccomandazioni e dalle successive fasi di apprendimento.

4) Apprendimento non supervisionato

L'apprendimento non supervisionato è un tipo di apprendimento automatico in cui il modello cerca schemi o strutture nei dati senza etichette predefinite, ovvero senza conoscere a priori la risposta corretta.

Si è cercato di affrontare sia un problema generale di complessità dei target, sia uno specifico legato alla ripetizione delle sottocategorie nel dataset. Durante l'analisi dei dati, è emerso che molte sottocategorie presentavano nomi ripetuti, ma gestivano audiolibri dello stesso tipo. Questo problema causava una frammentazione ingiustificata dei target, complicando il processo di classificazione. Ad esempio, la sottocategoria "Fantascienza" poteva apparire sotto diverse categorie oppure con nomi lievemente diversi come “Fantasy” e “Fantasy e Fantascienza”, generando più target distinti che, nella pratica, rappresentavano contenuti simili o uguali. Questa ridondanza aumentava la complessità e influiva negativamente sulle prestazioni dei modelli di apprendimento supervisionato.

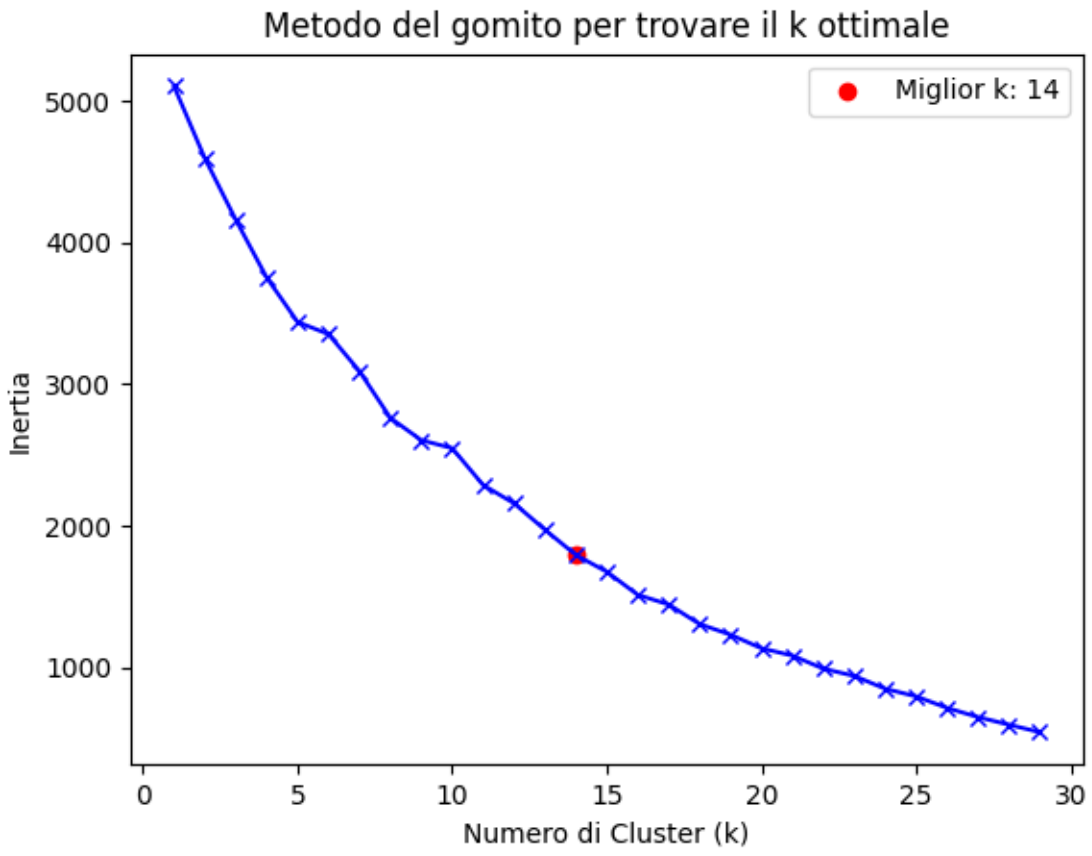
Per risolvere questo problema, è stato adottato un approccio di **clustering** tramite l'algoritmo **K-means**, con l'obiettivo di ridurre la complessità e migliorare l'efficacia dei target. Il clustering è stato utilizzato per raggruppare i target simili, combinando categorie e sottocategorie sovrapposte in un unico cluster. Questo ha ridotto la frammentazione, consentendo di lavorare con meno target, ma più distinti e significativi.

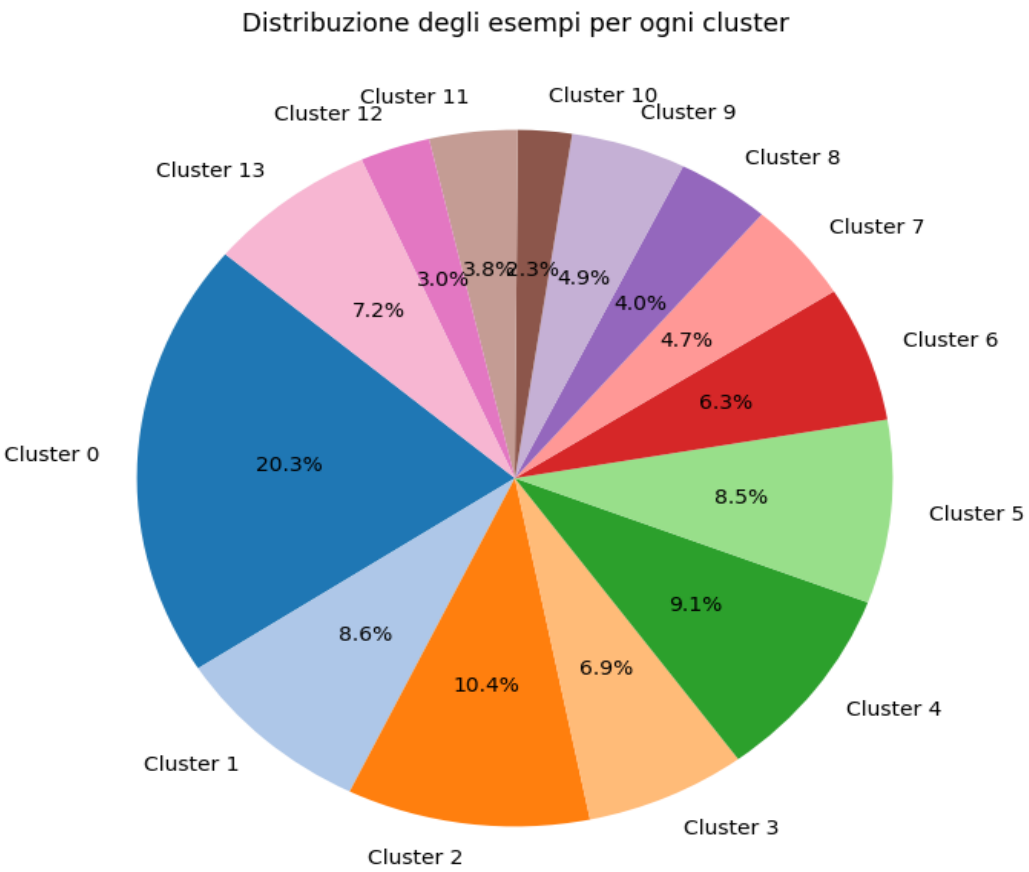
Il clustering è una tecnica di apprendimento non supervisionato che mira a raggruppare i dati in insiemi o cluster, basati su caratteristiche simili. Nel contesto del clustering, esistono due approcci: il **soft clustering** e l'**hard clustering**. Nel nostro caso, è stato adottato l'hard clustering, che assegna ogni audiolibro a un singolo cluster esclusivo, evitando ambiguità o sovrapposizioni tra i target. Questo approccio è particolarmente utile per semplificare il successivo processo di apprendimento, poiché ciascun audiolibro è chiaramente assegnato a una categoria specifica, senza appartenenze multiple con gradi diversi che potrebbero introdurre maggiore complessità.

Poiché il numero iniziale di classi target era superiore a 50, ho deciso di eseguire l'algoritmo K-means con un range di cluster compreso tra 1 e 30. Le impostazioni utilizzate includono $n\_init = 10$, che esegue K-means 10 volte con inizializzazioni diverse dei centroidi, selezionando la soluzione con la minore inerzia, e $init = k - means + +$, una tecnica che ottimizza la scelta iniziale dei centroidi, migliorando la convergenza. Il parametro $random\_state = 42$ è stato impostato per rendere i risultati riproducibili.

Per la clusterizzazione, i dati della colonna sono stati preprocessati usando la tecnica **TF-IDF** con le seguenti impostazioni: $max\_df = 0.7$ per ignorare i termini che appaiono in più del 70% dei documenti, eliminando quelli troppo comuni, e $min\_df = 2$ per escludere termini troppo rari. In quanto la seconda fase di preprocessing non era ancora stata applicata, la variabile stop_words è stata utilizzata per escludere parole comuni. Infine, la chiamata alla funzione $raggruppa\_e\_bilancia\_target()$ principale permette di scegliere se visualizzare o salvare i grafici generati durante l'esecuzione.

Un altro strumento importante utilizzato è la **regola del gomito**, che ha permesso di determinare il numero ottimale di cluster. La regola del gomito monitora come l'inerzia (la somma delle distanze tra i punti e i loro centroidi) varia con l'aumento del numero di cluster. L'algoritmo K-means riduce l'inerzia man mano che il numero di cluster aumenta, ma a un certo punto la riduzione diventa trascurabile. Il "gomito" in questo grafico indica il punto ottimale in cui fermarsi, bilanciando la riduzione dell'inerzia con la complessità dei dati. Dal grafico del gomito, si osserva che il punto ottimale per il nostro dataset è rappresentato da 14 cluster. Questo numero di cluster garantisce un buon compromesso tra granularità e riduzione della complessità.





Il grafico a torta che rappresenta la distribuzione degli esempi per ogni cluster mostra come i dati siano stati distribuiti nei 14 cluster identificati. La distribuzione non è completamente uniforme: alcuni cluster, come il Cluster 0, contengono una percentuale elevata di esempi (20.3%), mentre altri, come il Cluster 10 o il Cluster 11, ne contengono solo una piccola frazione (rispettivamente 2.3% e 3%). Questa variabilità può indicare che ci sono alcune combinazioni di categorie e sottocategorie molto più frequenti rispetto ad altre. Va precisato che, sebbene la distribuzione non sia perfettamente bilanciata tra i cluster, il bilanciamento è stato comunque migliorato rispetto alla struttura originaria.

5) Apprendimento supervisionato

L'apprendimento supervisionato è una delle principali tecniche dell'apprendimento automatico, in cui un agente o un modello viene addestrato su un insieme di dati etichettati. Ogni esempio di addestramento è composto da un input e dal corrispondente output o target, e l'obiettivo del modello è imparare la relazione che collega gli input agli output. Il compito più comune dell'apprendimento supervisionato è la **classificazione** (quando l'output è discreto) o la **regressione** (quando l'output è continuo). Altri tipi includono la **previsione strutturata**, dove l'output è una struttura di dati complessa. L'obiettivo finale dell'apprendimento supervisionato è quello di creare un modello in grado di generalizzare le relazioni tra input e output, in modo da effettuare previsioni accurate su nuovi dati mai visti prima. Il processo di apprendimento supervisionato si articola in diverse fasi:

- **Scelta degli iperparametri**: Gli iperparametri sono parametri che controllano il processo di apprendimento di un modello, ma che non vengono appresi direttamente dai dati. Gli iperparametri devono essere impostati manualmente o tramite tecniche di ottimizzazione prima dell'addestramento.
- **Fase di addestramento**: Durante questa fase, il modello viene addestrato sui dati etichettati, cercando di minimizzare l'errore tra le previsioni del modello e gli output reali.
- **Fase di test**: Il modello addestrato viene valutato su un insieme di dati separato (non utilizzato durante l'addestramento) per misurare le sue prestazioni su dati nuovi e non visti.
- **Valutazione delle prestazioni**: Le prestazioni del modello vengono misurate utilizzando metriche appropriate, come accuracy, precision, recall o errore quadratico medio.

In questo progetto, l'attenzione è stata rivolta alla classificazione. L'obiettivo principale era addestrare modelli apprendimento supervisionato per prevedere la classe corretta, utilizzando i dati etichettati, ottenuti in precedenza tramite le tecniche di clustering. Per l'addestramento dei modelli, sono stati utilizzate molte tra le tecniche più note, vediamole nel dettaglio con le rispettive impostazioni di iperparametri.

1. **Naive Bayes**: Algoritmo di classificazione basato sul Teorema di Bayes, che assume l'indipendenza condizionale tra le caratteristiche (feature). Stima la probabilità che un'osservazione appartenga a una determinata classe, basandosi

sulla probabilità condizionata di ciascuna caratteristica. Nonostante l'assunzione di indipendenza, risulta particolarmente efficace nell'elaborazione del testo.

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

- **clf__alpha**: L'iperparametro **alpha** nella Naive Bayes serve a regolarizzare le stime delle probabilità, aggiungendo uno smoothing (di solito Laplace smoothing). Questo evita problemi di probabilità zero, che possono verificarsi quando una categoria o un valore non sono presenti nei dati di addestramento.

```
naive_bayes_param_distributions = {**tfidf_params, 'clf__alpha': uniform(0.1, 2.0)}
```

```
• Migliori iperparametri: {'clf__alpha': 0.21247153170062907,
  'features__summary__tfidf__transformer__max_features': None,
  'features__tags__tfidf__transformer__max_features': 2000,
  'features__title__tfidf__transformer__max_features': None}
```

2. **Random Forest**: Algoritmo basato su un insieme di **alberi decisionali**. Ogni albero viene addestrato su un sottoinsieme casuale dei dati, e la previsione finale è ottenuta tramite la maggioranza delle previsioni (nel caso della classificazione) o la media (nel caso della regressione). Questo approccio riduce il rischio di sovradattamento (overfitting). Alcuni iperparametri chiave utilizzati includono:
   - **clf__n_estimators**: Questo iperparametro definisce il numero di alberi nella foresta. Un numero maggiore di alberi di solito aumenta la stabilità e l'accuratezza del modello, ma incrementa anche i tempi di calcolo.
   - **clf__max_depth**: La profondità massima di ciascun albero controlla quanto dettagliatamente ogni albero può dividere i dati. Profondità maggiori permettono alberi più complessi, ma rischiano di sovradattare i dati.
   - **clf__min_samples_split**: Numero minimo di campioni richiesti per effettuare una suddivisione di un nodo. Un numero più alto previene la creazione di nodi che si adattano troppo ai dati.
   - **clf__min_samples_leaf**: Numero minimo di campioni richiesti per essere una foglia dell'albero. Valori più alti riducono il rischio di sovradattamento.
   - **clf__criterion**: Criterio utilizzato per la misurazione della qualità di una divisione. Le opzioni includono 'gini', 'entropy', o 'log_loss', ognuna delle quali ottimizza l'albero in modi diversi.

```
random_forest_param_distributions = {**tfidf_params,
                                     'clf__n_estimators': randint(50, 301),
                                     'clf__max_depth': [None, 5, 10, 20],
                                     'clf__min_samples_split': randint(2, 11),
                                     'clf__min_samples_leaf': randint(1, 5),
                                     'clf__criterion': ['gini', 'entropy', 'log_loss']}
```

```
• Migliori iperparametri: {'clf__criterion': 'gini', 'clf__max_depth':
  None, 'clf__min_samples_leaf': 2, 'clf__min_samples_split': 2,
  'clf__n_estimators': 211,
  'features__summary__tfidf__transformer__max_features': 2000,
  'features__tags__tfidf__transformer__max_features': 2000,
  'features__title__tfidf__transformer__max_features': 2000}
```

3. **Decision Tree**: Modello che segmenta i dati basandosi sui valori delle caratteristiche, formando una struttura ad albero. Ogni nodo rappresenta una condizione su una caratteristica, e ogni foglia rappresenta una classe. Gli alberi decisionali tendono a sovradattarsi ai dati, motivo per cui vengono spesso utilizzati all'interno di modelli ensemble come la random forest.
    - **clf__criterion**: Indica il criterio di valutazione per le suddivisioni, che può essere 'gini', 'entropy', o 'log_loss'. Ogni criterio misura in modo diverso la purezza di una suddivisione.
    - **clf__max_depth**: Imposta la profondità massima dell'albero. Limiti di profondità più bassi riducono la complessità e il rischio di sovradattamento.
    - **clf__min_samples_split**: Numero minimo di campioni necessari per suddividere un nodo.
    - **clf__min_samples_leaf**: Numero minimo di campioni per formare una foglia.
    - **clf__splitter**: Metodo utilizzato per scegliere le suddivisioni nei nodi dell'albero. Le opzioni includono 'best', che seleziona la migliore suddivisione, o 'random', che seleziona casualmente un sottoinsieme di suddivisioni.

```
decision_tree_param_distributions = {**tfidf_params,
                                     'clf__criterion': ['gini', 'entropy', 'log_loss'],
                                     'clf__max_depth': [None, 5, 10],
                                     'clf__min_samples_split': randint(2, 21),
                                     'clf__min_samples_leaf': randint(1, 21),
                                     'clf__splitter': ['best', 'random']}
```

```
• Migliori iperparametri: {'clf__criterion': 'gini', 'clf__max_depth':
  None, 'clf__min_samples_leaf': 4, 'clf__min_samples_split': 6,
  'clf__splitter': 'random',
  'features__summary__tfidf__transformer__max_features': 3000,
  'features__tags__tfidf__transformer__max_features': 2000,
  'features__title__tfidf__transformer__max_features': 2000}
```

4. **Support Vector Machine (SVM)**: Algoritmo che cerca di identificare un iperpiano che separa le classi nel modo più ottimale, massimizzando il margine tra le classi. Se i dati non sono linearmente separabili, viene utilizzata la tecnica del **kernel** per proiettare i dati in spazi dimensionali più elevati. Gli iperparametri principali sono:
    - **clf__C**: Il parametro **C** controlla la regolarizzazione, bilanciando tra il margine massimo e la classificazione accurata. Valori più bassi di **C** comportano un margine più ampio ma con più errori di classificazione nei dati di addestramento.
    - **clf__kernel**: Il kernel definisce la trasformazione dei dati in uno spazio di dimensioni superiori per trovare un confine di separazione lineare. I kernel comuni sono 'linear', 'rbf' (radial basis function), 'poly' (polinomiale) e 'sigmoid'.
    - **clf__gamma**: Gamma controlla l'influenza di un singolo campione sul confine decisionale. Valori bassi di gamma indicano che ogni punto di dati ha un'influenza maggiore, mentre valori più alti riducono la portata di tale influenza. Opzioni come 'scale' e 'auto' sono approcci automatici per impostare gamma in base ai dati.

```
svm_param_distributions = {**tfidf_params,
                           'clf__C': uniform(0.1, 100.0),
                           'clf__kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
                           'clf__gamma': ['scale', 'auto'] + list(np.linspace(0.001, 1.0, 3))}
```

```
• Migliori iperparametri: {'clf__C': 8.800191491250908, 'clf__gamma':
   'scale', 'clf__kernel': 'linear',
   'features__summary__tfidf__transformer__max_features': 3000,
   'features__tags__tfidf__transformer__max_features': None,
   'features__title__tfidf__transformer__max_features': None}
```

5. **Logistic Regression**: Modello di classificazione lineare che viene utilizzato per stimare la probabilità che un campione appartenga a una certa classe. Anche se si chiama "regressione", è usata principalmente per problemi di classificazione binaria. Utilizza la funzione **sigmoide** per stimare la probabilità che un campione appartenga a una certa classe. Il modello ottimizza la **log-verosimiglianza** per trovare i pesi che massimizzano la probabilità. I principali iperparametri includono:
    - **clf__C**: Parametro di regolarizzazione che controlla il trade-off tra adattamento ai dati di addestramento e generalizzazione. Valori più alti riducono la regolarizzazione.

- **clf__penalty**: Determina il tipo di penalizzazione applicata, in questo caso viene utilizzata la penalizzazione **L2**, che preferisce soluzioni con parametri più piccoli.
- **clf__solver**: Algoritmo di ottimizzazione utilizzato per trovare i pesi del modello. Opzioni comuni includono 'liblinear', 'lbfgs', 'sag', e 'saga', ciascuno con diversi vantaggi per tipi specifici di dataset.
- **clf__max_iter**: Numero massimo di iterazioni per raggiungere la convergenza. Valori più alti permettono al modello di continuare a migliorare le sue previsioni fino a raggiungere un livello accettabile di ottimizzazione.

```
logistic_regression_param_distributions = {**tfidf_params,
                                           'clf__C': uniform(0.001, 100.0),
                                           'clf__penalty': ['l2'],
                                           'clf__solver': ['liblinear', 'lbfgs', 'sag', 'saga'],
                                           'clf__max_iter': randint(100, 1001)}
```

```
• Migliori iperparametri: {'clf__C': 58.974168514283754, 'clf__max_iter':
  696, 'clf__penalty': 'l2', 'clf__solver': 'lbfgs',
  'features__summary__tfidf__transformer__max_features': None,
  'features__tags__tfidf__transformer__max_features': None,
  'features__title__tfidf__transformer__max_features': 1000}
```

Il cuore del codice dedicato all'addestramento dei modelli è la funzione $train\_models()$, che offre un alto grado di personalizzazione. L'utente può specificare i modelli da utilizzare in input per l'addestramento e scegliere le feature su cui basare l'apprendimento. Ogni feature viene pesata in modo decrescente, dando maggiore rilevanza alle prime. Per ciascuna feature testuale, viene creata una pipeline che include la trasformazione $TF - IDF$ pesata, e queste pipeline sono combinate in una $FeatureUnion$, che permette al modello di considerare tutte le feature contemporaneamente.

In Python una pipeline automatizza la sequenza di passaggi, rendendo il processo più efficiente e riproducibile. Nel nostro caso è stata utilizzata per automatizzare l'addestramento del modello, riducendo il rischio di errori o ridondanze.

Il $TF - IDF$ è una tecnica che converte i testi in vettori numerici. La **Term Frequency** calcola quanto spesso una parola appare in un documento, mentre la **Inverse Document Frequency** misura quanto una parola sia rara nel corpus. Viene attribuito un peso maggiore alle parole frequenti in un documento ma rare nel corpus, riducendo l'importanza di termini comuni che non aiutano a distinguere tra i documenti.

I dizionari degli iperparametri vengono definiti per ogni modello disponibile, e se l'utente non specifica quali modelli addestrare, vengono utilizzati tutti i modelli. Per bilanciare le classi, SVM utilizza un bilanciamento interno delle classi, mentre per gli altri modelli viene applicato $RandomOverSampler$, che replica casualmente gli esempi della classe minoritaria fino a ottenere una distribuzione bilanciata. Si è scelto $RandomOverSampler$ al posto di tecniche più complesse come $SMOTE$, per mantenere la semplicità e ridurre il carico di lavoro.

Per l'ottimizzazione degli iperparametri, il codice utilizza $RandomizedSearchCV$ al posto di $GridSearchCV$. Anche questa scelta è stata fatta per ragioni pratiche legate all'efficienza in termini di tempo e risorse. $RandomizedSearchCV$ esegue un numero di iterazioni casuali (ad esempio, $n\_iter = 50$), riducendo il tempo di esecuzione rispetto all'approccio esaustivo di $GridSearchCV$. La scelta di questa tecnica permette di esplorare lo spazio degli iperparametri rispettando i limiti di risorse disponibili. Per ogni combinazione casuale di iperparametri, viene eseguita una **cross-validation** su 5 sottogruppi, ripetuta 3 volte, utilizzando il metodo $RepeatedStratifiedKFold$.

$RepeatedStratifiedKFold$ è una tecnica di cross-validation che combina la stratificazione dei dati con la ripetizione del processo di suddivisione. La **stratificazione** garantisce che ciascun fold mantenga la proporzione delle classi del dataset, mentre la **ripetizione** permette di eseguire più cicli di suddivisione con diverse partizioni dei dati, migliorando l'affidabilità delle valutazioni del modello e riducendo la variabilità nelle stime delle prestazioni.

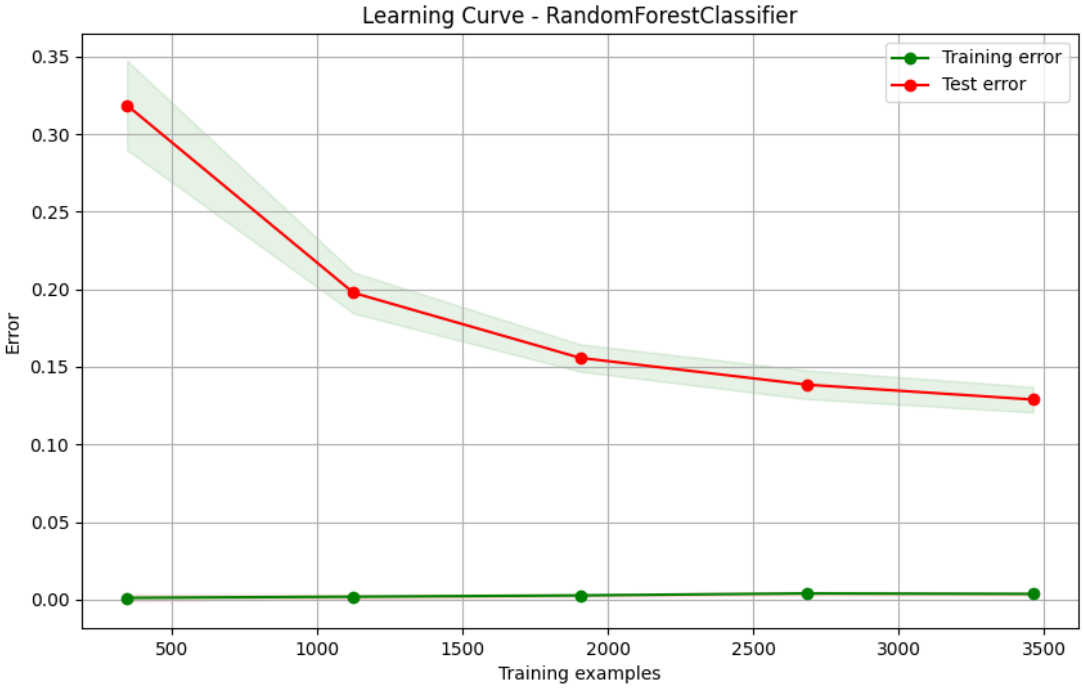
Ricordiamo che la cross-validation è una tecnica utilizzata per valutare le prestazioni di un modello di machine learning in modo più affidabile. Il dataset viene suddiviso in più **fold** (sottogruppi). In ogni iterazione, uno dei fold viene usato come set di test, mentre i restanti fold vengono usati per l'addestramento del modello. Questo processo viene ripetuto per ciascun fold e le prestazioni medie vengono calcolate.

La cross-validation fornisce una stima più affidabile delle prestazioni del modello su dati non visti, riducendo il rischio di overfitting e di dipendenza dalla suddivisione iniziale dei dati.

Tuttavia, in alcuni casi, nonostante l'uso di varie tecniche, l'overfitting non è stato completamente eliminato, indicando che la complessità dei dati o il disequilibrio tra classi potrebbe richiedere ulteriori interventi.

Passiamo ora a rivedere i risultati e le prestazioni di ogni modello, con un'analisi mirata sui potenziali segnali di overfitting:

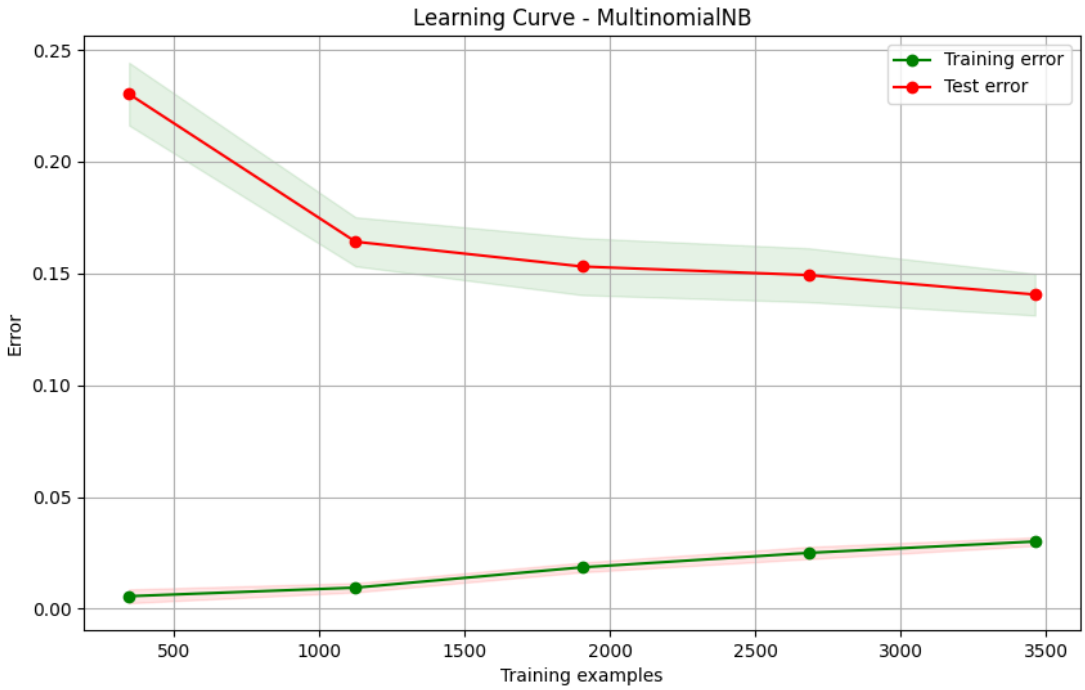
Random Forest:



- **Errore di training molto basso** e **errore di test** leggermente più alto: Anche qui, il modello sembra essere abbastanza accurato nei dati di addestramento, ma il comportamento nel test set è abbastanza stabile. Questo potrebbe essere un lieve overfitting, ma generalmente accettabile.

	precision	recall	f1-score	support
0	0.79	0.91	0.84	223
1	0.93	0.87	0.90	91
2	0.93	0.79	0.86	97
3	0.95	0.80	0.87	75
4	0.89	0.88	0.89	106
5	0.94	0.88	0.91	93
6	0.84	0.83	0.83	75
7	0.83	0.75	0.79	52
8	0.84	0.92	0.88	51
9	0.84	0.94	0.89	50
10	0.90	0.82	0.86	22
11	0.82	1.00	0.90	41
12	0.74	0.96	0.84	27
13	0.92	0.75	0.83	80
accuracy			0.86	1083
macro avg	0.87	0.86	0.86	1083
weighted avg	0.87	0.86	0.86	1083

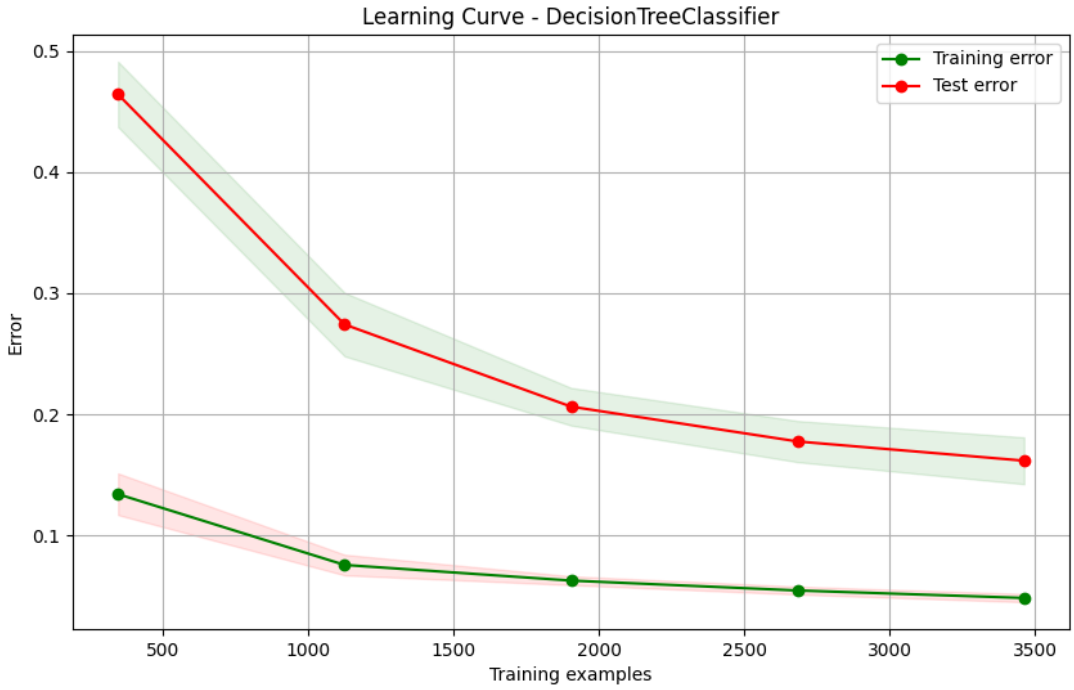
MultinomialNB:



- **Errore di training** e **errore di test** abbastanza vicini: Questo modello sembra comportarsi bene senza segni evidenti di overfitting. I due errori convergono in modo accettabile.

	precision	recall	f1-score	support
0	0.90	0.80	0.85	223
1	0.94	0.90	0.92	91
2	0.90	0.78	0.84	97
3	0.87	0.89	0.88	75
4	0.83	0.91	0.87	106
5	0.90	0.81	0.85	93
6	0.81	0.91	0.86	75
7	0.60	0.77	0.67	52
8	0.84	1.00	0.91	51
9	0.86	0.76	0.81	50
10	0.58	0.86	0.69	22
11	0.77	0.83	0.80	41
12	0.79	0.85	0.82	27
13	0.96	0.90	0.93	80
accuracy			0.85	1083
macro avg	0.83	0.86	0.84	1083
weighted avg	0.86	0.85	0.85	1083

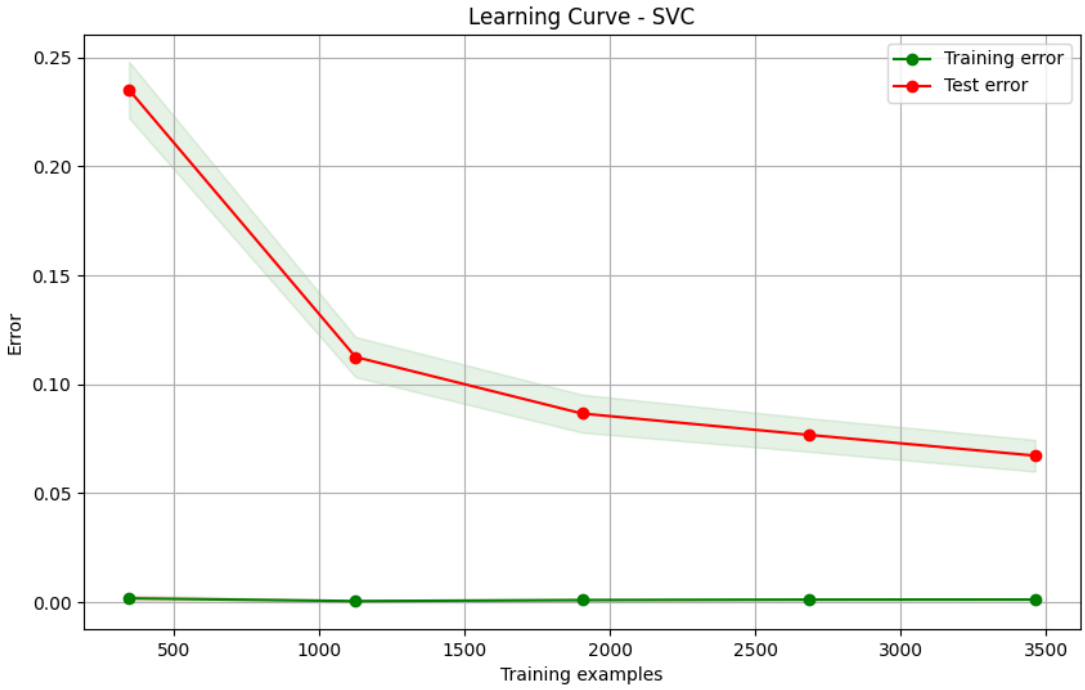
Decision Tree:



- **Errore di addestramento molto basso** e **errore di test più alto**: Qui si vede chiaramente un overfitting. L'errore di training è quasi nullo, ma l'errore di test rimane significativamente più alto, indicando che l'albero decisionale si è adattato troppo ai dati di addestramento.

	precision	recall	f1-score	support
0	0.80	0.84	0.82	223
1	0.88	0.89	0.89	91
2	0.86	0.77	0.82	97
3	0.82	0.91	0.86	75
4	0.82	0.79	0.80	106
5	0.93	0.75	0.83	93
6	0.78	0.79	0.78	75
7	0.84	0.79	0.81	52
8	0.82	0.90	0.86	51
9	0.90	0.94	0.92	50
10	0.86	0.82	0.84	22
11	0.80	0.88	0.84	41
12	0.87	0.96	0.91	27
13	0.74	0.72	0.73	80
accuracy			0.83	1083
macro avg	0.84	0.84	0.84	1083
weighted avg	0.83	0.83	0.83	1083

SVM (Support Vector Machine):



- **Errore di training praticamente nullo** questo è un chiaro segno che il modello sta **memorizzando i dati di addestramento**, un segnale di **overfitting**.
- **Errore di test** più alto e non così vicino all'errore di training: Questo suggerisce che il modello ha difficoltà a generalizzare sui dati di test, supportando l'ipotesi di overfitting.

	precision	recall	f1-score	support
0	0.90	0.96	0.93	223
1	1.00	0.95	0.97	91
2	0.97	0.92	0.94	97
3	0.96	0.96	0.96	75
4	0.93	0.93	0.93	106
5	0.92	0.90	0.91	93
6	0.94	0.88	0.91	75
7	0.87	0.90	0.89	52
8	0.91	0.96	0.93	51
9	0.89	0.96	0.92	50
10	0.95	0.95	0.95	22
11	1.00	0.93	0.96	41
12	0.90	0.96	0.93	27
13	0.97	0.91	0.94	80
accuracy			0.93	1083
macro avg	0.94	0.93	0.94	1083
weighted avg	0.94	0.93	0.93	1083

Logistic Regression:

- **Errore di addestramento vicino a 0**: Anche in questo caso, un errore di training così basso indica un potenziale overfitting.
- **Errore di test più alto ma stabile**: Sebbene l'errore di test non sia estremamente elevato, il contrasto con l'errore di addestramento è significativo, suggerendo che il modello si adatta troppo ai dati di training e non riesce a generalizzare altrettanto bene.

	precision	recall	f1-score	support
0	0.92	0.96	0.94	223
1	1.00	0.93	0.97	91
2	0.97	0.90	0.93	97
3	0.96	0.97	0.97	75
4	0.93	0.90	0.91	106
5	0.97	0.91	0.94	93
6	0.92	0.88	0.90	75
7	0.84	0.94	0.89	52
8	0.89	0.96	0.92	51
9	0.87	0.96	0.91	50
10	0.95	0.95	0.95	22
11	0.98	1.00	0.99	41
12	0.93	0.96	0.95	27
13	0.96	0.94	0.95	80
accuracy			0.94	1083
macro avg	0.94	0.94	0.94	1083
weighted avg	0.94	0.94	0.94	1083

Per concludere, riportiamo le matrici di confusione di ciascun modello e un grafico che confronta le prestazioni basate sulle seguenti metriche:

1. **Accuracy**: misura la percentuale di predizioni corrette rispetto al totale delle predizioni effettuate. È calcolata come il rapporto tra le predizioni corrette (sia vere positive che vere negative) e il totale delle predizioni. Tuttavia, l'accuracy può essere fuorviante se il dataset è sbilanciato, poiché il modello potrebbe semplicemente predire sempre la classe dominante per ottenere un'accuracy elevata.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2. **Precision**: misura la percentuale di predizioni corrette per la classe positiva rispetto al totale delle predizioni fatte per quella classe. È particolarmente utile quando i falsi positivi hanno un costo elevato e vogliamo ridurre al minimo i falsi allarmi. Un'alta precisione indica che pochi esempi classificati come positivi sono in realtà falsi positivi.

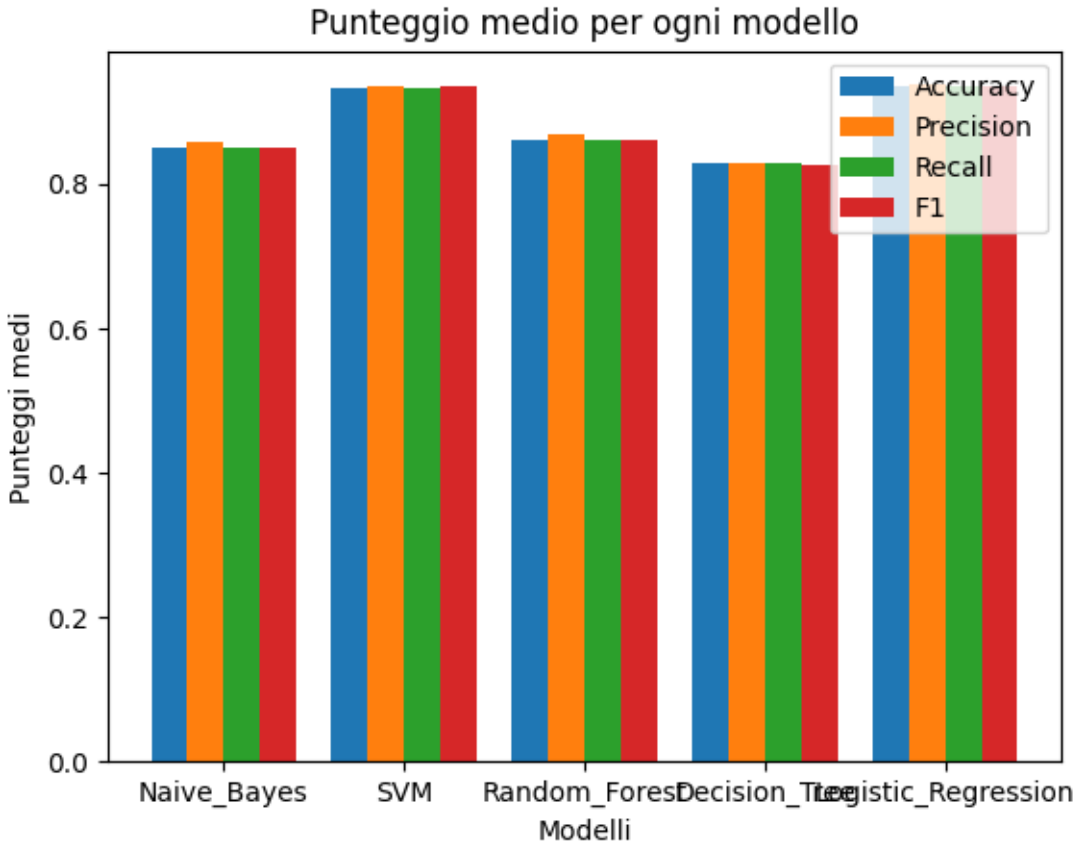
$$Precision = \frac{TP}{TP + FP}$$

3. **Recall**: (noto anche come sensibilità o tasso di veri positivi) misura la capacità del modello di identificare correttamente i casi positivi. Indica la percentuale di veri positivi rispetto al totale degli esempi che avrebbero dovuto essere classificati come positivi. È utile quando l'obiettivo è minimizzare i falsi negativi, ad esempio in scenari come la diagnosi medica.

$$Recall = \frac{TP}{TP + FN}$$

4. **F1-Score**: è la media armonica di precision e recall. Si usa quando è necessario trovare un equilibrio tra precision e recall, specialmente quando il dataset è sbilanciato. Un alto F1 indica che sia precision che recall sono elevate. Viene preferito all'accuracy in caso di sbilanciamento delle classi, perché prende in considerazione sia i falsi positivi che i falsi negativi.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$



6) Sviluppi futuri

Come anticipato nell'introduzione, il progetto rappresenta una base solida per futuri sviluppi su scala più ampia. Alcune possibili direzioni includono:

- **Integrazione con reti neurali**: L'uso di **reti neurali** potrebbe rappresentare un significativo miglioramento per l'analisi dei dati testuali, permettendo di captare relazioni più complesse tra le feature. Queste reti, note per la loro capacità di apprendere rappresentazioni gerarchiche, potrebbero potenziare l'accuratezza delle classificazioni e delle raccomandazioni, migliorando la comprensione e il processamento dei testi, come riassunti e recensioni.
- **Analisi avanzata delle recensioni**: Le recensioni non sono state considerate. Un possibile sviluppo futuro potrebbe essere l'uso di tecniche di **sentiment analysis** o l'estrazione di opinioni specifiche per identificare trend e sentimenti predominanti, migliorando la qualità delle raccomandazioni basate sul feedback degli utenti.
- **Espansione del Dataset**: Aumentare la quantità e la diversità dei dati di addestramento considerando audiolibri in altre lingue, potrebbe mitigare la problematica dell’overfitting.