UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

SmartSort: sistema di documentazione intelligente

AA 2024-25

Vito Stefano Birardi, 755782, v.birardi3@studenti.uniba.it

Simone Columpsi, 758299, s.columpsi@studenti.uniba.it

Repository GitHub: SmartSmort

Indice:

Introduzione

Il progetto si inserisce nel dominio dell'organizzazione intelligente dei dati, con l'obiettivo di creare un agente software in grado di organizzare autonomamente diverse tipologie di documenti testuali.

In un contesto in cui la quantità di informazioni digitali è in costante crescita, la necessità di sistemi automatici che possano classificare e strutturare i file in modo logico è diventata cruciale.

L'agente è progettato per operare su un insieme disomogeneo di file, che includono documenti scientifici, appunti universitari, codice sorgente, progetti personali e pagine web.

Sommario

Il sistema basato sulla conoscenza (KBS) sviluppato integra diversi moduli per affrontare il problema della classificazione documentale. L'architettura combina tecniche di machine learning per l'analisi del contenuto con un sistema di ragionamento basato su vincoli per l'organizzazione logica, il quale alla fine restituirà la categoria migliore da assegnare al file, assiema ad una percentuale di accuratezza.

Elenco argomenti di interesse

- **Apprendimento supervisionato e Incertezza [Capitolo 12]**: Utilizzo di un modello di classificazione single-label per assegnare ai documenti la categoria più pertinente con un relativo grado di confidenza.

- **Progettazione di Ontologie [Capitolo 16]**: Definizione di una struttura ontologica per distinguere in modo formale tra "Risorse" (i documenti) e "Posizioni" (le categorie), garantendo coerenza nella rappresentazione della conoscenza.

- **Constraint Satisfaction Problems (CSP) [Capitolo 4**]: Implementazione di vincoli gerarchici tramite `python-constraint` per garantire coerenza nelle predizioni multilivello tra L1, L2, e L3, con funzioni specializzate per setup e risoluzione di problemi CSP.

- **Knowledge Representation e Knowledge Base [Capitolo 15]:** Utilizzo del file Ontology.owx come knowledge base formale con namespace semantici, triple RDF e accesso dinamico tramite query SPARQL per la rappresentazione strutturata del dominio.

- **Feature Engineering e Rappresentazione [Capitolo 7]:** Creazione di feature avanzate combinando rappresentazioni TF-IDF con feature semantiche ontologiche attraverso `create_enhanced_features()` e matrici sparse per alta dimensionalità.

- **Ensemble Methods [Capitolo 7, Sezione 7.5]:** Architettura gerarchica con Logistic Regression per L1, Random Forest per L2, e ensemble SVM + Naive Bayes per L3.
  La combinazione delle probabilità avviene tramite media aritmetica semplice (probs_l3_svm + probs_l3_nb) / 2.0 per ottenere predizioni più robuste nel livello più specifico della gerarchia. `(dataset_create_gerarchy.py)`

- **Hierarchical Classification e Multi-level Learning [Capitolo 7]:** Classificazione su 3 livelli gerarchici con modelli specializzati, classe HierarchicalClassifier personalizzata e pipeline gerarchica con

vincoli progressivi. Questa classe è utilizzato nell'approccio gerarchico, mentre nell'approccio ricorsivo la gerarchia viene presa in considerazione solo durante la fase di predizione.

- **Local Search e Optimization [Capitolo 4, Sezione 4.6]:** Utilizzo di tecniche di ricerca locale per l'ottimizzazione dei parametri dei modelli, inclusi algoritmi SGD e hyperparameter tuning. Utilizzata esclusivamente per creare visualizzazioni della curva di loss, NON per l'addestramento principale.

- **Probabilistic Reasoning [Capitolo 11]:** Ragionamento probabilistico per gestire l'incertezza nelle classificazioni gerarchiche con predict_proba() e CSP con vincoli probabilistici.

- **Model Evaluation e Cross-Validation [Capitolo 7, Sezione 7.4.3]:** Metodologie rigorose di valutazione con metriche comprehensive (accuracy, precision, recall, F1), confusion matrix e split stratificati.

Strumenti utilizzati

Tale progetto si avvale dell'utilizzo di diverse librerie Python, opportunamente impiegate in base alle diverse fasi dalle quali l'intero progetto in questione è costituito. Come editor è stato utilizzato principalmente Visual Studio Code, mentre per l'ontologia è stato utilizzato Protégé.

La fase di preparazione dei dati e analisi del testo utilizza:

- **Pandas** per la gestione dei dati,
- **PyMuPDF (fitz)** e **PyPDF2** per l'estrazione del testo dai PDF
- **NLTK,spaCy e re** per la pulizia e la tokenizzazione del testo.
- **csv e os** per gestione file system e I/O

La componente di machine learning è implementata con:

- **Scikit-learn** per la vettorizzazione TF-IDF e l'addestramento di modelli come:
    - Logistic Regression,
    - Random Forest
    - Kernel SVM.
    - Multinomial Naive Bayes
- **Scipy** per matrici sparse e operazioni matematiche avanzate

La gestione della knowledge base e vincoli utilizza:

- **rdflib** per la gestione dell'ontologia OWL/RDF e query SPARQL
- **python-constraint** per l'implementazione di CSP e vincoli gerarchici

L'analisi esplorativa e visualizzazione sono supportate da:

- **Matplotlib e Seaborn** per grafici e visualizzazioni
- **NumPy** per operazioni numeriche
- **collections** per strutture dati specializzate come:
    - Defaultdict
    - Counter
- **pickle** per persistenza e serializzazione modelli

Capitolo 1: Estrazione e Pulizia del Testo

La creazione del dataset utile per l'addestramento del modello passa attraverso diverse fasi, attraverso le quali il dataset viene gradualmente raffinato.

Il motivo di tale scelta è da attribuire a una strategia metodologica che privilegia la **modularità**, il **controllo qualità** e l'**efficienza computazionale**. Invece di eseguire un unico processo monolitico, la pipeline di elaborazione dati è stata volutamente suddivisa in fasi distinte, ognuna delle quali produce un "artefatto" intermedio.

Questo approccio più granulare ha permesso non solo di rendere più rapido il debugging, ma anche l'implementazione di nuove feature ha subito una notevole semplficazione.

Descrizione delle funzioni principali

Creazione del file .csv

Il dataset di partenza viene costruito attraverso lo script `create_csv.py`, che esplora ricorsivamente **due gerarchie di directory distinte** alla ricerca di file (in particolare PDF) dai quali estrarre i metadati fondamentali.

Il sistema opera su due dataset separati per supportare una valutazione rigorosa dei modelli:

- **Cartella di Training**: `./training_data/` per l'addestramento dei modelli
- **Cartella di Test**: `./test_data/` per la valutazione delle performance

Per ciascun documento vengono acquisiti dati come:

- **Titolo**, estratto dai metadati
- **Nome del file**, che potrebbe differire dal titolo (viene usato nel caso in cui i metadati non forniscano un titolo sufficientemente adatto)
- **Autore del testo**
- **Anno di pubblicazione**
- **Categoria**, che rimane vuota per un utilizzo futuro
- **Estensione del file**
- **Tipo di file** (cartella o file)
- **Percorso relativo** del file a partire dalla rispettiva cartella base

L'estrazione dei metadati PDF avviene utilizzando la libreria **PyPDF2**, che permette di leggere direttamente le proprietà interne ai file PDF.

Lo script raccoglie queste informazioni in strutture dati separate che vengono scritte in file CSV distinti:

- **Training set**: `./training_result/output.csv`
- **Test set**: `./test_result/testoutput.csv`

Questi file costituiscono la base informativa per le fasi successive del progetto e consentono di mantenere la separazione tra dati di training e test.

Il processo è stato evoluto per gestire entrambi i dataset in modo sistematico:

- Processing Training Set: La funzione `process_folder_to_csv('./trainingdata', './trainingresult/output.csv')` esplora ricorsivamente la cartella di training tramite `explore_folder_recursive`
- Processing Test Set: La funzione `process_folder_to_csv('./testdata', './testresult/testoutput.csv')` processa separatamente la cartella di test
- Estrazione metadati: Ogni PDF viene processato tramite `extract_metadata_from_pdf()` per estrarre le informazioni
- Scrittura CSV: I dati vengono scritti nei rispettivi file CSV tramite `write_to_csv()`
- Riepilogo statistico: Il sistema fornisce un report completo con il totale dei file processati da entrambe le cartelle

Estrazione e Pulizia del Testo

Il modulo `text_extract.py` si occupa di acquisire il contenuto testuale dei documenti PDF utilizzando la libreria **PyMuPDF**, scelta per la sua efficacia nel gestire testi complessi e numerosi formati di pagina.

Il processo è stato adattato per operare su entrambi i dataset mantenendo la separazione:

- **Training data**: Processing da `./trainingdata/` → `output ./trainingresult/outputwithtext.csv`
- **`Test data`**: `Processing da ./testdata/ → output ./testresult/testdatawithtext.csv`

L'estrazione avviene pagina per pagina, unendo il testo in una stringa complessiva per singolo documento.

Il testo estratto viene quindi sottoposto a una pipeline di pulizia e preprocessing che elimina caratteri speciali, numeri e punteggiatura, tramite regular expression

Di seguito, un esempio di espressione regolare utilizzata nel caso di studio:

```
def clean_text(self, text):
        if not text:
            return ""
        text = re.sub(r'[^a-zA-Z\s]', '', text)
        text = text.lower()
        text = re.sub(r'\s+', ' ', text).strip()
        return text
```

Successivamente, con l'uso combinato di NLTK e spaCy, il testo viene tokenizzato (suddiviso in parole), le stop words (parole non rilevanti come articoli o preposizioni) vengono rimosse, e le parole rimanenti vengono riportate alla loro forma base attraverso la lemmatizzazione.

Di seguito, si riporta il codice utilizzato per la tokenizzazione:

```
def tokenize_text(self, text):
    tokens = word_tokenize(text)
    tokens = [self.lemmatizer.lemmatize(token) for token in tokens if token not in self.stop_words and len(token) > 2]
    return tokens
```

Decisioni di progetto

La pulizia del testo consente di ottenere testi coerenti e ridotti a una forma standardizzata, essenziale per l'applicazione di modelli NLP e feature statistiche.

La decisione di mantenere due dataset separati fin dall'inizio garantisce:

- **Valutazione rigorosa**: Nessun data leakage tra training e test
- **Riproducibilità**: Risultati consistenti e verificabili
- **Scalabilità**: Possibilità di aggiungere nuovi documenti mantenendo la separazione

Sono state inoltre calcolate feature descrittive del testo, come:

- **Numero di parole totali**
- **Diversità lessicale** (rapporto tra parole uniche e totali)
- **Lunghezza delle parole e delle frasi**

La matrice TF-IDF viene generata **esclusivamente** sui dati di training tramite `generate_tfidf_matrix()`, evitando bias nella valutazione.

Librerie utilizzate

Per la realizzazione dello script `create_csv.py e text_extract.py`, sono state utilizzate le seguenti librerie, ciascuna con un ruolo specifico:

- **os**: È stata essenziale per la navigazione ricorsiva delle cartelle (`os.walk`), la manipolazione dei percorsi dei file (`os.path.join, os.path.basename`) e l'estrazione dei nomi dei file e delle loro estensioni (`os.path.splitext`).
- **PyPDF2**: È stata utilizzata per aprire i documenti, leggere la loro struttura interna e accedere in modo programmatico ai metadati (come Titolo, Autore e Data di Creazione) attraverso la classe `PdfReader`.
- **csv**: È stato impiegato per creare il file `output.csv`, scrivere la riga di intestazione e popolare il file con tutti i dati raccolti, utilizzando `csv.DictWriter` per garantire una scrittura strutturata e robusta dei dati.
- **PyMuPDF** (**fitz**): Per l'estrazione efficiente del testo dai PDF con gestione avanzata dei formati di pagina.
- **pickle**: Per il salvataggio della matrice TF-IDF e altri oggetti per uso nelle fasi successive.

Capitolo 2: Categorizzazione automatica dei documenti

Il sistema di categorizzazione del progetto SmartSort utilizza un approccio basato su keyword semantiche associate a categorie ontologiche, implementato attraverso lo script `categorize_files.py`.

Tramite questo codice, viene categorizzato sia il dataset di training che quello di test.
Lo script utilizza la **knowledge base ontologica** (`Ontology.owx`).

Metodologia di Base

L'obiettivo principale è assegnare a ogni documento una sola categoria semantica, da inserire nel campo "category" del file .csv creato nella fase precedente, scegliendo la più specifica possibile all'interno di una gerarchia definita di categorie, organizzate per livello di specificità basate su un sistema di parole chiave (keyword) semantiche.

Gerarchia delle categorie:

- **Categorie molto specifiche** (foglie ontologiche):
    - Ambiente
    - Chimica
    - Ecologia
    - Energia
    - Spazio
    - AI_ML
    - Comunicazione
    - Data_analysis
    - Database
    - Security
    - System_programming
    - Web_development
    - Alimentazione
    - Cardiologia
    - Oncologia
    - Archeologia
    - Culturale
    - Animale
    - Botanica
    - Umana

    - Antica
    - Contemporanea
    - Filosofia
    - Moderna
    - Preistoria
- **Categorie specifiche** (nodi intermedi):
    - Biologia
    - Fisica
    - Informatca
    - Medicina
    - Antropologia
    - Paleontologia
    - Storia
- **Categorie generali** (rami principali):
    - Scienza
    - Studi umanistici
- **Categoria fallback**: Altro

Caratteristiche dell'Implementazione

Il sistema organizza automaticamente i risultati prodotti nella fase precedente in:

- **Training set**: `./training_result/training_set_categorized.csv`
- **Test set**: `./test_result/test_set_categorized.csv`

Il sistema fornisce report dettagliati sulla distribuzione delle categorie per monitorare la qualità della categorizzazione.

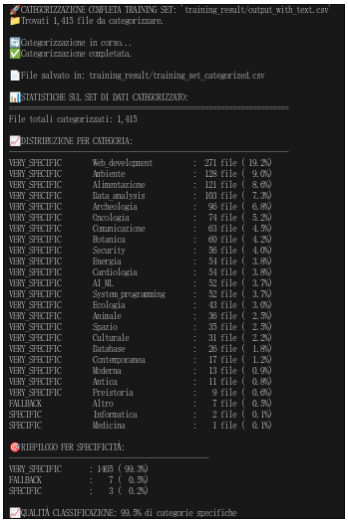
- Per valutare la categoria più appropriata per ogni documento, la funzione somma i punteggi associati al numero di occorrenze di keyword, pesate in base alla lunghezza della parola chiave (keyword più dettagliate hanno pesi maggiori). Si cerca quindi di identificare la categoria con il punteggio più alto, dando priorità alle categorie più specifiche (foglie ontologiche) rispetto a quelle più generali.
- Se nessuna delle keyword è rilevante, si assegna la categoria sulla base dell'estensione del file, associando estensioni comuni a categorie di esempio (es. .py → AI_ML, .cpp → System_programming).

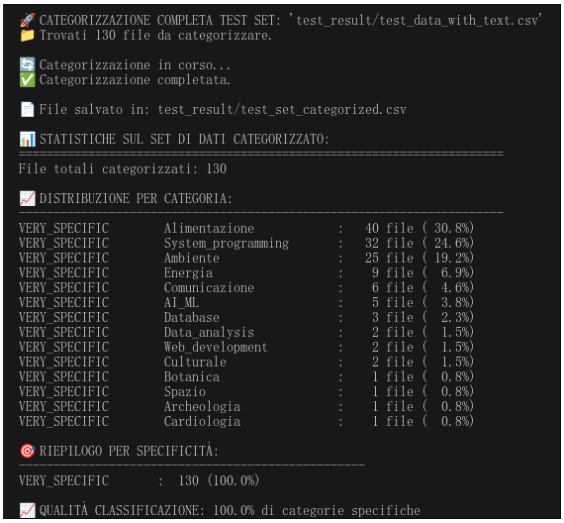
```
extension_categories = {
".html": ["Web_development"],
".css": ["Web_development"],
".js": ["Web_development"],
".php": ["Web_development"],
".py": ["AI_ML"],
".java": ["System_programming"],
".cpp": ["System_programming"],
".c": ["System_programming"],
".sql": ["Database"],
".csv": ["Data_analysis"],
".json": ["Data_analysis"],
".unknown": ["Altro"]
}
```

- Se ancora non è possibile assegnare una categoria, si assegna la categoria generica di “Altro”.

Il risultato finale è un file CSV aggiornato che sovrascrive la colonna category creata in precedenza, in cui ogni documento ha la sua singola categoria assegnata secondo questa logica gerarchica e semantica.

Lo script riporta infine statistiche riassuntive sulla distribuzione delle categorie nel dataset.





Questa metodologia consente una classificazione granulare e automatica, compatibile con la strutturazione di un’ontologia semantica e funzionale a supportare moduli successivi di ragionamento automatico e apprendimento supervisionato nel progetto.

Descrizione delle funzioni principali

Funzione principale: categorize_entire_file(input_csv, output_folder, output_file)

Input:

- `input_csv`: File CSV contenente i documenti con testo estratto

- `output_folder:` Directory di destinazione (`./test_result/` e `./training_result`)
- `output_file:` Nome del file di output

Operazioni principali:

1. Carica il dataset completo in un DataFrame pandas
2. Filtra i file validi (con titolo non nullo)
3. Processa il 100% dei documenti validi
4. Per ogni documento, utilizza `find_most_specific_category(row)` per assegnare la categoria più specifica

```
def find_most_specific_category(row):

titolo = str(row.get('titolo', '')).lower()
filename = str(row.get('filename', '')).lower()
extension = str(row.get('extension', '')).lower()
abstract = str(row.get('abstract', '')).lower() if 'abstract' in row else ''
clean_text = str(row.get('clean_text', '')).lower() if 'clean_text' in row else ''
full_text = f"{titolo} {filename} {abstract} {clean_text}"

category_scores = defaultdict(int)
for category, keywords in category_keywords.items():
        if keywords:
        for keyword in keywords:
            pattern = r'\b' + re.escape(keyword.lower()) + r'\b'
            matches = len(re.findall(pattern, full_text.lower()))
            if matches > 0:
                weight = len(keyword.split()) * 2
                category_scores[category] += weight * matches

best_category = None
best_score = 0

for level in ['very_specific', 'specific']:
        if best_category is None:
            for category in categories_by_specificity[level]:
            score = category_scores.get(category, 0)
            if score > best_score:
                best_score = score
                best_category = category

              if best_category is None and extension in extension_categories:
                ext_cats = extension_categories[extension]
                if ext_cats and ext_cats[0] != 'Altro':
                    best_category = ext_cats[0]
```

```
if best_category is None:
        for category in categories_by_specificity['general']:
            if category_scores.get(category, 0) > 3:
            best_category = category
            break
if best_category is None:
        best_category = "Altro"
        return best_category
```

5. Salva i risultati in file separati per training e test set
6. Genera statistiche dettagliate sulla distribuzione delle categorie

Estrazione dinamica delle keyword dall'ontologia OWL tramite query SPARQL

Algoritmo per l'estrazione autoamtica della categoria dall'ontologia per caiscun file del training set e test set:

```
def estrai_category_keywords_da_ontologia(ontology_path):
    g = Graph()
    g.parse(ontology_path, format="xml")
    ns = Namespace("http://www.semanticweb.org/vsb/ontologies/2025/8/untitled-ontology-11")
    hasKeyword = ns.hasKeyword

    category_keywords = defaultdict(list)
    for s, p, o in g.triples((None, hasKeyword, None)):
        cat_name = str(s).split("#")[-1]
        category_keywords[cat_name].append(str(o))

    return dict(category_keywords)
```

Tale funzione ha il compito di interrogare dinamicamente il file dell'ontologia (`Ontology.owx`) per costruire un dizionario di parole chiave. Inizializza un grafo rdflib e vi carica il file dell'ontologia.

Definisce quindi il *namespace* (lo schema URI) e la proprietà specifica da cercare, in questo caso hasKeyword. Il suo ciclo principale itera su tutti i *triples* (soggetto, predicato, oggetto) presenti nel grafo, cercando specificamente quelli in cui il predicato (la relazione) è hasKeyword. Per ogni tripla trovata, estrae il nome della categoria dal soggetto (es. "AI_ML" dall'URI completo) e la parola chiave letterale dall'oggetto (es. "machine learning"), aggiungendo la **parola** chiave a una lista associata a quella categoria in un dizionario.

Infine, restituisce questo dizionario completo che mappa ogni categoria alla propria lista di keyword.

Funzione di supporto: `print_statistics(df)`

Fornisce analisi dettagliate sulla distribuzione delle categorie assegnate, incluse:

- Conteggio per categoria
- Percentuali di distribuzione
- Livelli di specificità raggiunti

Compatibilità con entrambi gli approcci di addestramento

Il sistema di categorizzazione unificato supporta perfettamente entrambi gli approcci successivi:

- Approccio Gerarchico (`dataset_create_gerarchy.py`): Utilizza i file dalla directory `./training_result/training_set_categorized.csv` e `./test_result/test_set_categorized.csv`
- Approccio Ricorsivo (`dataset_ricorsivo.py`): Accede agli stessi dati categorizzati, garantendo coerenza

Decisioni di progetto

La decisione di utilizzare un unico script di categorizzazione porta diversi benefici:

- **Coerenza:** Stessa logica di categorizzazione per entrambi gli approcci
- **Manutenibilità:** Un solo script da mantenere e aggiornare
- **Efficienza:** Eliminazione della duplicazione di codice
- **Riproducibilità**: Risultati consistenti per tutti gli esperimenti

È stato deciso di assegnare una sola categoria per ciascun documento per motivi di chiarezza, gestione e interpretabilità della classificazione automatica.

La scelta di assegnare la categoria più specifica possibile (la categoria "foglia" nell'ontologia) massimizza la precisione semantica della classificazione, assegnando l'etichetta più dettagliata che i dati permettono di determinare.

Il sistema processa il 100% dei documenti disponibili, garantendo:

- **Copertura totale:** Nessun documento escluso dalla categorizzazione
- **Riproducibilità:** Risultati consistenti e deterministici
- **Robustezza:** Dataset completi per l'addestramento dei modelli

Librerie utilizzate

Lo script `categorize_files.py` utilizza le seguenti librerie:

- **pandas:** per la manipolazione e gestione dei dati in DataFrame
- **re (regular expressions):** per la ricerca e il matching di parole chiave nel testo
- **rdflib:** per l'interazione con l'ontologia OWL e l'estrazione dinamica delle keyword tramite SPARQL
- **collections** (defaultdict, Counter): per contare occorrenze e gestire dizionari di categorie e parole chiave
- **os:** per la gestione dei percorsi e la creazione di directory

Capitolo 3: Addestramento dei Modelli

Il processo di classificazione documentale utilizza l'**Ontologia** per definire una gerarchia di categorie su tre livelli (**L1 - Generale**, **L2 - Specifico**, **L3 - Molto Specifico/Foglia**). La classificazione avviene attraverso un approccio ibrido che combina Machine Learning e ragionamento basato su vincoli (CSP).

Da questo punto, sono stati intrapresi due approcci differenti. Sono stati infatti sviluppati i codici di `dataset_ricorsivo.py e dataset_create_gerarchy.py.`

Sono state intraprese strade differenti, per valutarne pregi e difetti di ognuna, e per avere una valutazione più accurata delle varie modifiche apportate.

Approccio Ricorsivo (Logica CSP Post-Predizione)

Il file `dataset_ricorsivo.py` è concepito come una pipeline robusta, il cui obiettivo primario è generare predizioni affidabili e persistenti.
La sua strategia di addestramento è indipendente: i modelli per i livelli L1, L2 e L3 vengono addestrati separatamente, ciascuno sull'intero set dei dati di training, senza che il livello più generale influenzi quello più specifico, almeno in questa fase.

Tutta la logica gerarchica è demandata al momento della predizione, che impiega un meccanismo di risoluzione dei vincoli (CSP) a due stadi: il "Piano A" tenta di trovare la combinazione L1 -> L2 -> L3 gerarchicamente valida con la probabilità più alta; se fallisce, ad esempio a causa di previsioni contrastanti, entra in gioco il "Piano B", ossia un fallback che prende la predizione L3 più probabile e ricostruisce la gerarchia all'indietro (L3 -> L2 -> L1) consultando l'ontologia.

Questo approccio garantisce che l'output sia sempre coerente, eliminando gli errori.
Lo script è costruito per la persistenza, salvando i modelli (.pkl) e i dati processati per evitare un nuovo addestramento. Inoltre, implementa il filtraggio automatico delle classi con meno di 5 campioni per prevenire l'instabilità del modello e gli errori di Stratified K-Fold.

Approccio Gerarchico (Logica Gerarchica in Addestramento)

Al contrario, **dataset_create_gerarchy.py** funziona come un **laboratorio sperimentale**, progettato per valutare una specifica strategia di addestramento alternativa.

La sua filosofia è quella di **far apprendere la gerarchia** ai modelli durante l'addestramento stesso, tramite un **approccio a cascata**: il modello L2 viene addestrato solo sui campioni che il modello L1 ha predetto correttamente in modo coerente, e lo stesso vale per L3, che apprende solo dai campioni coerenti con le previsioni di L2. L'obiettivo di questa tecnica è migliorare l'accuratezza finale.

Viene applicato inoltre un **Ensemble Method** dove si utilizza la **media aritmetica semplice** delle probabilità tra un modello **SVM** e un modello **Naive Bayes** per la predizione sul livello L3, con l'obiettivo di ottenere una classificazione più robusta in questo livello di massima specificità.

Il suo meccanismo di predizione utilizza un **CSP di base**. Se i modelli, anche se addestrati gerarchicamente, producono probabilità in conflitto tali da rendere impossibile la risoluzione ottimale, il CSP garantisce un **risultato coerente** assegnando la categoria **'Altro'**. Questo meccanismo assicura che il sistema fornisca sempre una tripla di categorie gerarchicamente valida.

Lo scopo primario di questo script non è la produzione, ma l'**analisi**, come dimostrano le sue funzioni per generare grafici K-Fold e curve di loss, valutando l'efficacia di questa specifica tecnica di training.

Descrizione delle funzioni principali

Lo sviluppo si basa su un set di funzioni **comuni** per la gestione del dataset e la *feature engineering*:

- **`Load_keywords_from_ontology()`**: Estrae le keyword per ogni categoria direttamente dall'ontologia OWL (Ontology.owx) tramite la libreria rdflib.
- **Create_enhanced_features()**: Conta le occorrenze delle keyword ontologiche nel testo di ogni documento, creando **feature semantiche**. Queste vengono concatenate (*stack*) con le feature **TF-IDF** (ottenute con TfidfVectorizer) per formare il set di feature finale.
- **get_parents()**: Consulta l'Ontologia per determinare la relazione `rdfs: subClassOf` (genitore/figlio) tra le categorie, essenziale per i vincoli CSP.
- **setup_csp_problem():** Crea un problema CSP in cui le variabili sono i livelli di classificazione (L1, L2, L3) e i vincoli assicurano la **coerenza gerarchica** (L3 deve essere figlio di L2, e L2 figlio di L1).
- **find_best_csp_solution():** Trova la combinazione di categorie coerenti che **massimizza la probabilità congiunta**, utilizzando le probabilità calcolate dai classificatori.

Le funzioni più caratteristiche che differenziano chiaramente i due approcci, l'**Approccio Ricorsivo** (`dataset_ricorsivo.py`) e l'**Approccio Gerarchico** (`dataset_create_gerarchy.py`), sono quelle legate al *training* con vincoli e alla risoluzione delle predizioni.

Funzioni Caratteristiche dell'Approccio Gerarchico

Questo approccio si distingue per la logica di vincolo imposto durante la fase di addestramento (training a cascata).

- **`HierarchicalClassifier`:** Implementa il training vincolato.
  È la funzione centrale che filtra il training set (con `_get_consistent_samples`) per includere solo i campioni la cui etichetta vera è coerente con la previsione del modello padre (es. L3 addestrato solo su campioni validi rispetto a L2).
- **`HierarchicalTrainingPipeline`:** Orchestrazione del training a cascata.
  Gestisce la sequenza di addestramento: predice L1, usa le predizioni per vincolare il training di L2, predice L2, e usa le predizioni per vincolare il training di L3. Questo assicura che il vincolo agisca sul modello stesso.
- **`plot_kfold_accuracies`:** Valutazione sperimentale dell'Ensemble.
  Genera un grafico a barre per confrontare l'accuratezza K-Fold di SVM, Naive Bayes e il loro Ensemble combinato per L3.

Funzioni Caratteristiche dell'Approccio Ricorsivo

Questo approccio si distingue per il suo focus sulla robustezza e persistenza, applicando i vincoli solo dopo l'addestramento indipendente.

- **Pre-processing e filtraggio** (nel blocco `if __name__ == "__main__":`)
  Implementa il filtraggio automatico delle classi rare (`min_samples_per_class = 5`), escludendo le categorie con pochi campioni per prevenire l'instabilità del modello e gli avvisi di Stratified K-Fold.
- **Serializzazione Dati/ModelliPersistenza**. (nel blocco `if __name__ == "__main__":`)
  Utilizza `pickle.dump` e `save_npz` per salvare e caricare il TfidfVectorizer, i modelli addestrati e le

feature combinate (`X_train_combined.npz`) per evitare un nuovo addestramento ad ogni esecuzione.

- **evaluate_and_get_metrics:** Valutazione dettagliata.
  Calcola e stampa le metriche complete (Accuracy, Precision, Recall, F1) e il rapporto di classificazione per ogni singola pipeline di modello (LR, RF, SVM, NB), non solo l'ensemble.
- **Logica CSP (implicita):** Risoluzione predittiva.
  La logica CSP è utilizzata in modo robusto su quattro pipeline separate (una per ogni modello) per garantire che ogni predizione sia gerarchicamente coerente, senza dipendere dal training vincolato.

Decisioni di progetto

La selezione dei classificatori copre un ampio spettro di complessità e tipologia di modelli, consentendo un confronto rigoroso in termini di generalizzazione e capacità predittiva:

- **Logistic Regression (LR)**: Offre un baseline **lineare**, efficiente, interpretabile e rapido. Utilizzato come modello di base e spesso per il livello L1 (più generale).
- **Random Forest (RF)**: Un modello **ensemble** robusto, in grado di catturare relazioni non lineari. È un modello a basso *bias* ma con alto *variance* (se non controllato).
- **Support Vector Machine (SVM)**: Efficace in spazi vettoriali ad alta dimensionalità (come quelli TF-IDF), ottimizzando il margine di separazione. Viene utilizzato nell'ensemble L3 gerarchico.
- **Multinomial Naive Bayes (NB)**: Ideale per la classificazione di testi (modello probabilistico) e complementare a SVM nell'ensemble L3 grazie alla sua ipotesi di indipendenza delle feature.

Questi modelli vengono confrontati attraverso metriche standard come accuracy e F1 score ponderato sul test set; verrà selezionato il migliore in termini di generalizzazione e capacità predittiva.

Il Ruolo Centrale dell'Ontologia

L'ontologia OWL è la fonte unica e autorevole per le categorie e le keyword semantiche associate.

Permette di creare feature significative costruite su concetti espressi dalla knowledge base, andando oltre la mera rappresentazione testuale.

Inoltre, definisce la gerarchia delle categorie per garantire coerenza semantica durante l'addestramento e la valutazione.

Il ruolo dell'ontologia è duplice:

1. **Feature Engineering:** Fornisce un vocabolario semantico per creare **feature avanzate** basate su matching di keyword nei testi (`create_enhanced_features`).
2. **Vincolo Gerarchico:** Definisce i **livelli gerarchici** delle categorie (`rdfs:subClassOf`) per garantire coerenza semantica durante il training (nell'approccio gerarchico) e il ragionamento CSP.

Librerie utilizzate

Per l'implementazione degli script di addestramento (`dataset_ricorsivo.py` e `dataset_create_gerarchy.py`), è stato impiegato un insieme di librerie Python specializzate, ciascuna scelta per un compito specifico all'interno della pipeline di *machine learning* e ragionamento automatico.

Librerie Comuni (Feature Engineering e Modelli)

- **pandas**: Utilizzata per caricare, manipolare e filtrare il dataset iniziale dal file CSV in un DataFrame.
- **scikit-learn (sklearn)**: Fornisce tutti gli strumenti per la vettorizzazione TF-IDF (TfidfVectorizer) e l'addestramento dei modelli di classificazione come **Logistic Regression**, **Random Forest**, **SVM** e **Multinomial Naive Bayes**.
- **rdflib**: Indispensabile per interagire con la Knowledge Base ontologica, caricando il file in formato OWL/RDF e navigando il grafo per estrarre dinamicamente le keyword e le relazioni gerarchiche (rdfs:subClassOf).
- **NumPy**: il suo ruolo è quello di gestire in modo efficiente array e operazioni numeriche, inclusa la manipolazione delle matrici sparse.
- **Scipy**: Utilizzata in combinazione con NumPy e scikit-learn per la gestione delle **matrici sparse** (**`hstack, csr_matrix, save_npz`**) e operazioni matematiche avanzate.
- **collections**: Fornisce strutture dati specializzate:
    - **Counter**: Per contare in modo efficiente le occorrenze delle keyword ontologiche nei testi.
    - **defaultdict**: Per aggregare facilmente le keyword per categoria dopo l'estrazione dall'ontologia.
- **re (Regular Expressions)**: Utilizzato per la ricerca di pattern specifici e il *matching* delle keyword ontologiche all'interno del testo durante la creazione delle feature semantiche.
- **Constraint:** Fondamentale per l'implementazione del **Constraint Satisfaction Problem (CSP)** e l'applicazione dei vincoli gerarchici durante la predizione.

Librerie Distintive (Ragionamento e Persistenza)

- **Pickle (Approccio Ricorsivo)**: Utilizzata in modo estensivo per la **persistenza** e la serializzazione (`pickle.dump`) dei modelli addestrati, del TfidfVectorizer e delle etichette.
- **sklearn.base.BaseEstimator/ClassifierMixin (Approccio Gerarchico):** Utilizzata per definire la classe **HierarchicalClassifier**, che è il wrapper personalizzato che aggiunge la logica di **vincolo al metodo fit()** (addestramento a cascata).
- **sklearn.model_selection.StratifiedKFold (Approccio Gerarchico):** Utilizzata per la valutazione sperimentale del modello Ensemble tramite Cross-Validation a K=3.

Capitolo 4: Valutazioni e visualizzazioni dei dati

La fase finale del progetto consiste nella valutazione rigorosa delle performance dei modelli e nella generazione di report visivi completi.

Questa sezione confronta i risultati ottenuti dall'**Approccio Ricorsivo** (modelli indipendenti con CSP) e dall'**Approccio Gerarchico** (Ensemble con training a cascata).

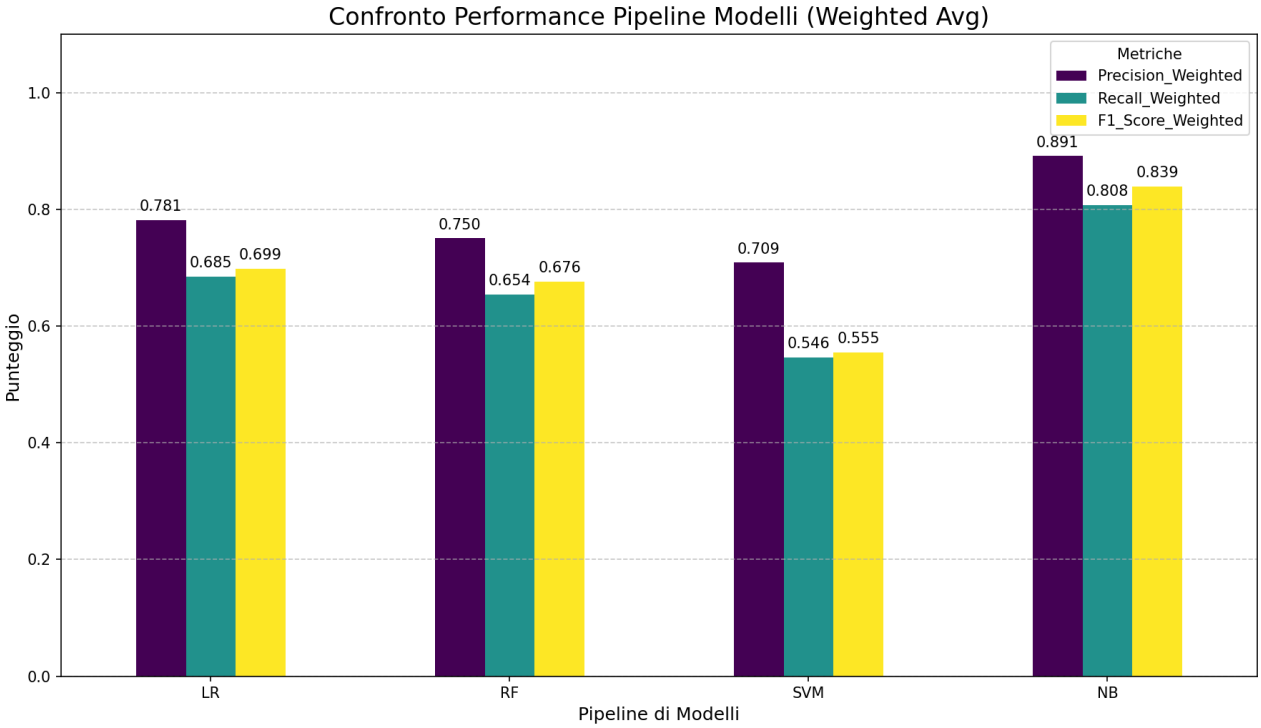
Metodologie di Valutazione

Approccio Ricorsivo (Analisi Dettagliata per Modello)

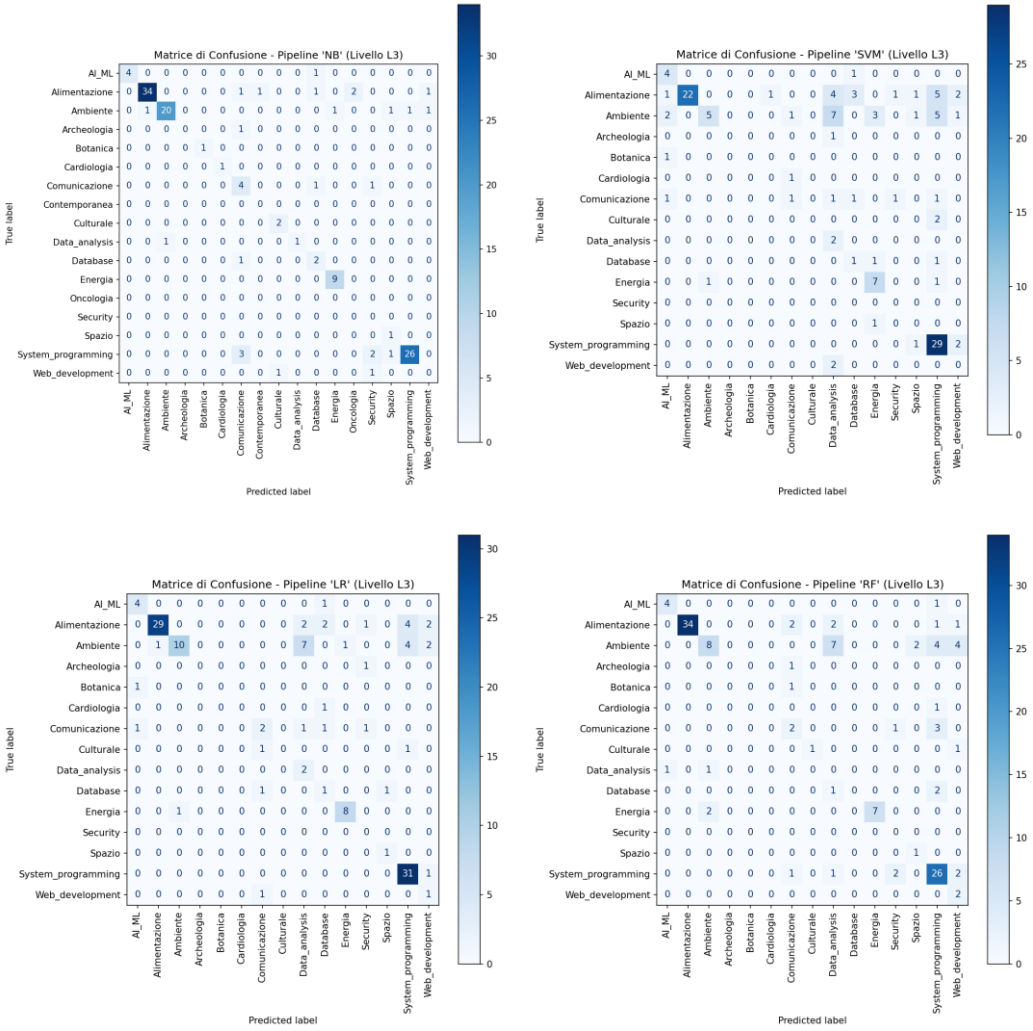
L'analisi è gestita dallo script `stats.py`, che valuta la capacità predittiva di ogni singolo classificatore (LR, RF, SVM, NB) dopo che le loro previsioni sono state rese coerenti dal CSP.

Lo script esegue diverse operazioni principali di analisi e visualizzazione.

Per prima cosa, lo script genera un **Riepilogo delle Performance Aggregate** attraverso la funzione `plot_performance_summary`. Questa funzione produce un grafico a barre che riassume e confronta le metriche di classificazione chiave (Precision, Recall e F1-Score, tutte calcolate in modo ponderato per la media complessiva) per tutte le singole pipeline di modelli (LR, RF, SVM, NB) sul set di test.
Questo *reporting* visuale fornisce una panoramica immediata delle prestazioni, evidenziando il modello più performante, come ad esempio il Naive Bayes con il suo F1-Score di **0.839**.



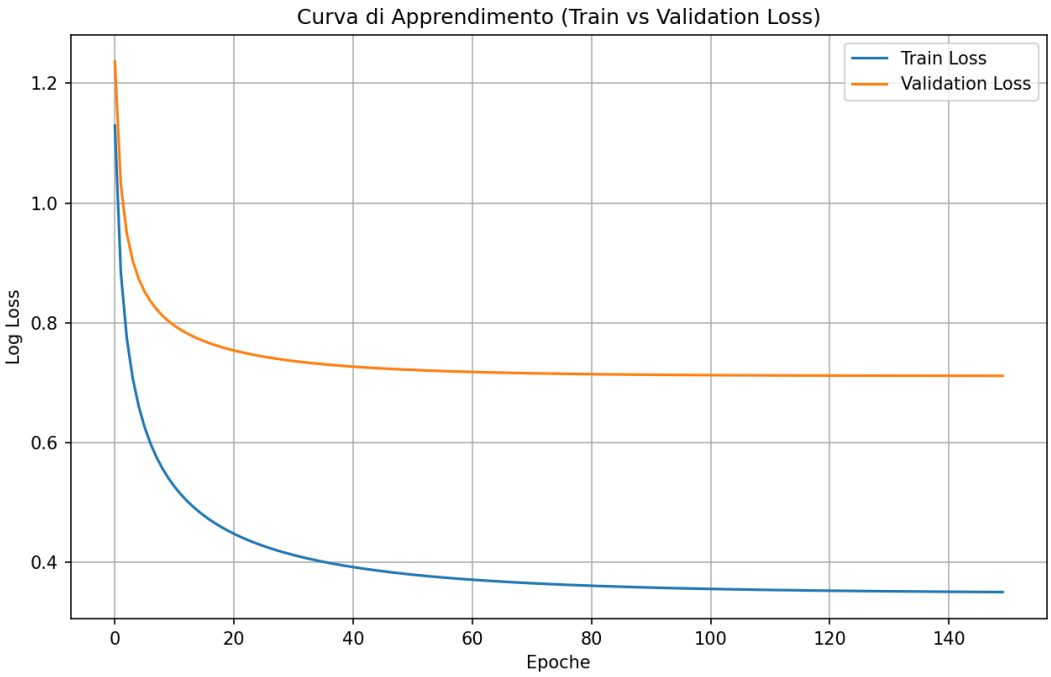
Successivamente, per una diagnosi più approfondita degli errori, stats.py esegue la generazione delle **Matrici di Confusione** tramite la funzione `plot_confusion_matrices()`.
Lo script produce una matrice separata per il **Livello L3** per ciascuno dei modelli addestrati (LR, RF, SVM, NB).



Queste matrici sono cruciali per visualizzare la distribuzione degli errori di classificazione, permettendo di identificare quali categorie specifiche vengono confuse tra loro, come ad esempio la tendenza di LR a scambiare 'Ambiente' con 'Alimentazione'.

Diagnostica del Modello e Feature Analysis

Un altro compito vitale è la **Diagnostica del Modello** eseguita dalla funzione `plot_loss_curve()`. Questa genera la **Curva di Apprendimento** confrontando la *Train Loss* con la *Validation Loss*. Per questo scopo, lo script utilizza un classificatore **SGD** (SGDClassifier) con un *learning rate* adattivo, addestrato in modo iterativo su una porzione del set di training.



La curva di apprendimento in figura è essenziale per la diagnostica del modello. Si nota un chiaro gap tra la Train Loss (che continua a scendere) e la Validation Loss (che si stabilizza precocemente a circa 0.7), un segnale di overfitting . Il modello, cioè, sta imparando troppo bene i dati di training a scapito della capacità di generalizzare.

Proprio per mitigare questo problema, è stato necessario un attento tuning dei parametri, culminato nella decisione di ridurre drasticamente max_features (in dataset_ricorsivo.py) a un valore di **30**.

Sebbene questa scelta non elimini completamente il gap di overfitting (come ancora visibile nel grafico), essa si è rivelata strategica: riducendo il vocabolario a sole 30 feature, si costringe il modello a basare le sue decisioni solo sui pattern semantici più generali e robusti. Questo ha migliorato la stabilità complessiva, evitando che il classificatore assegni categorie errate a causa di lievi variazioni testuali.

Tuttavia, questa riduzione così aggressiva delle *feature* ha introdotto un costo inevitabile: un'aumentata incertezza in alcuni modelli, in particolare nel Naive Bayes. Tale incertezza si è manifestata in una piccola percentuale di risultati 'incoerenti', dove la predizione ha portato a una violazione della gerarchia ontologica. Si è comunque scelto di procedere, valutando questo come un rischio minimo e gestito (dato che il fenomeno è stato circoscritto solo al 5% circa dei documenti nel test set) .

Infine, `stats.py` include una sezione dedicata al **Feature Engineering** con la funzione `plot_top_tfidf_features()`. Questa analizza i risultati della vettorializzazione estraendo e visualizzando i termini più rilevanti, classificati in base al loro **punteggio IDF**. Un punteggio IDF elevato indica che la parola è rara e specifica all'interno del corpus, rendendola particolarmente distintiva per la classificazione. Ad esempio, le parole 'soil' e 'cell' sono le più rilevanti per l'IDF, segnale che i documenti relativi a Biologia e Ambiente sono ben caratterizzati a livello semantico.

Approccio Gerarchico (Analisi Sperimentale dell'Ensemble)

L'analisi è integrata negli script **`dataset_create_gerarchy.py`** e **`metriche gerarchy.py`**.
Si concentra sul risultato dell'Ensemble finale (SVM + Naive Bayes) e sulla diagnostica interna del training a cascata (K-Fold e Curva di Loss).
L'analisi dell'Approccio Gerarchico non si concentra sulla *reportistica* di ogni singolo modello (come fa `stats.py`), ma è una **valutazione sperimentale** focalizzata sulla validazione di due decisioni chiave di progetto: l'efficacia del **Training Vincolato a Cascata** e il miglioramento prestazionale fornito dal **Metodo Ensemble** per il livello L3.
Questa analisi è distribuita tra l'azione dello script di training (`dataset_create_gerarchy.py`) e il modulo di misurazione dedicato (`metriche gerarchy.py`).

Modulo di Misurazione: metriche_gerarchy.py

Il modulo `metriche gerarchy.py` ha il compito primario di elaborare le predizioni finali dell'Ensemble (SVM + Naive Bayes) confrontandole con le etichette vere del set di test.
A differenza dell'approccio Ricorsivo, che si concentra sull'output del modello L3, reso coerente dal CSP, l'analisi Gerarchica deve garantire che la coerenza del *training* si traduca in accuratezza a tutti i livelli.
La funzione **`calcola_metriche()`** (in `metriche_gerarchy.py`) esegue i seguenti passaggi:

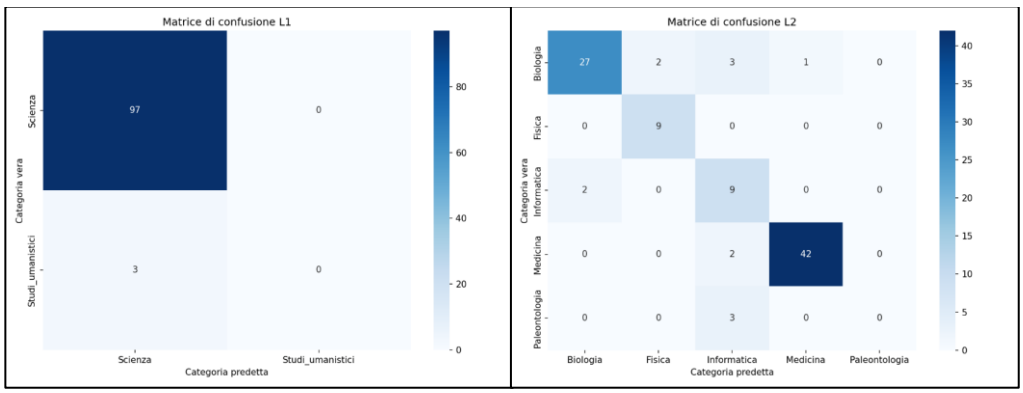
1. Prima di confrontare le predizioni, il modulo utilizza l'Ontologia (tramite le funzioni di supporto come `get_hierarchy_levels`) per ricostruire e validare i livelli L1 e L2 associati a ciascuna categoria L3 vera. Questo assicura che il confronto con le predizioni L1 e L2 dell'Ensemble sia significativo.
2. Calcola le metriche di performance standard (**Precision, Recall, F1-Score**, tutte in versione **Weighted Average**) separatamente per i livelli L1, L2 e L3.

```
Livello L1: Precision=0.941, Recall=0.970, F1=0.955
Livello L2: Precision=0.869, Recall=0.870, F1=0.864
Livello L3: Precision=0.880, Recall=0.830, F1=0.848
```

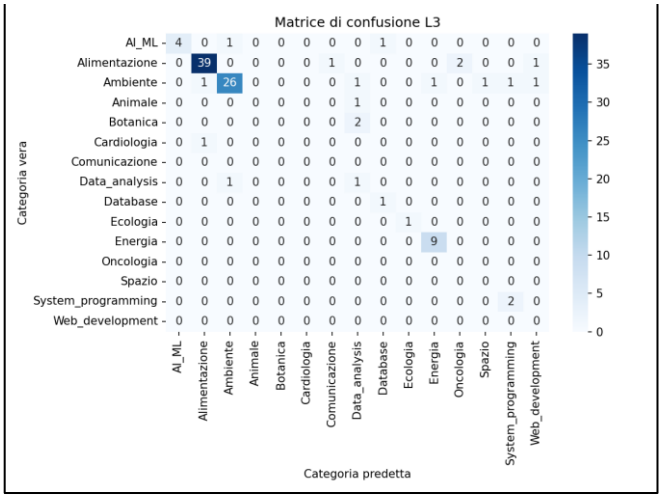
Generazione delle Matrici di Confusione

Il modulo genera e salva tre matrici di confusione distinte, una per livello (L1, L2, L3). L'analisi di queste matrici ha confermato l'efficacia del training a cascata nel mantenere l'accuratezza ai livelli superiori:

- La **Matrice L1** ha mostrato una classificazione quasi perfetta (97 corretti su 100), dimostrando che i modelli L1 e L2 beneficiano della coerenza gerarchica imposta.



- La **Matrice L3** evidenzia la difficoltà intrinseca della classificazione a grana fine, sebbene le classi più popolari siano ben gestite.

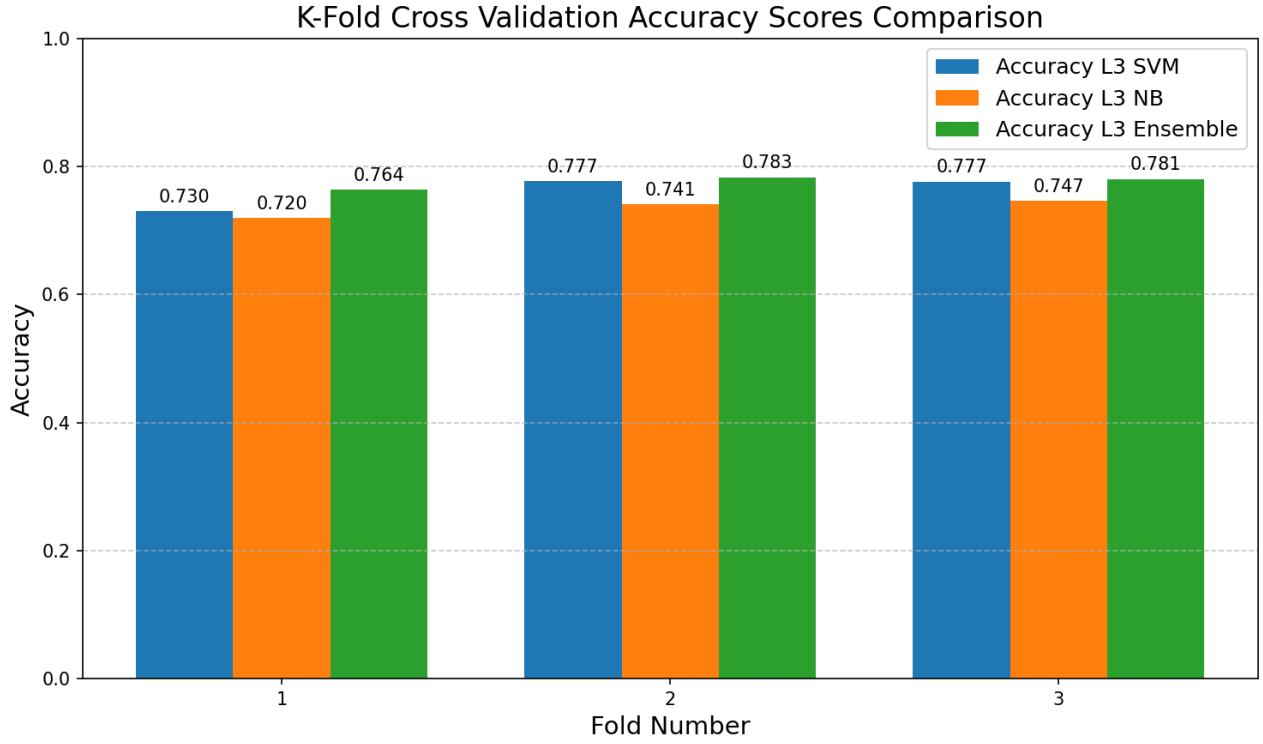


Funzioni Integrate in `dataset_create_gerarchy.py` (Diagnostica Sperimentale)

L'approccio Gerarchico include funzioni diagnostiche integrate direttamente nello script di training (`dataset_create_gerarchy.py`) che mirano a convalidare la strategia dell'Ensemble e a diagnosticare il comportamento di apprendimento specifico del training vincolato a cascata.

Confronto K-Fold delle Accuratezze - `plot_kfold_accuracies()`

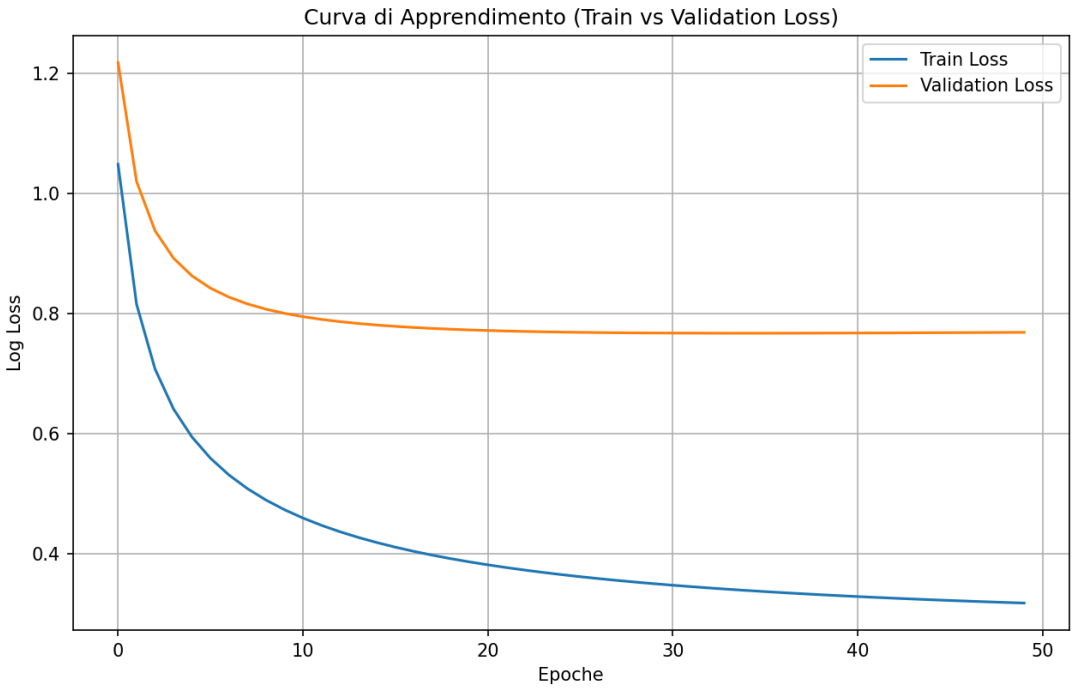
La funzione `plot_kfold_accuracies()` è centrale per la validazione quantitativa della strategia di Ensemble. Essa genera un grafico a barre che valuta le accuratezze ottenute dai singoli modelli L3 (**SVM** e **Naive Bayes**) rispetto alla loro **combinazione Ensemble**. Lo scopo primario è dimostrare che l'utilizzo di questo Metodo Ensemble (che si basa sulla media aritmetica delle probabilità) produce risultati sistematicamente superiori rispetto all'uso del miglior modello L3 preso singolarmente.
Infatti, il risultato chiave di questa analisi ha confermato che l'Accuracy media L3 dell'Ensemble è la più alta in ogni iterazione K-Fold, stabilendosi a circa **0.7762**, il che convalida pienamente la decisione progettuale di utilizzare l'Ensemble per il livello più specifico.



Curva di Apprendimento Integrata - `plot_loss_curve()`

La funzione `plot_loss_curve()` viene utilizzata come strumento diagnostico per analizzare in modo specifico gli effetti del training a cascata sul comportamento di apprendimento del modello L3. Questa funzione genera la **Curva di Apprendimento** confrontando la *Train Loss* con la *Validation Loss* di un modello **SGD** (Stochastic Gradient Descent), spesso scelto per la sua efficienza.

Lo scopo è diagnosticare tempestivamente problemi di **Overfitting** o **Underfitting**. L'analisi di questa curva ha rivelato un **forte gap** tra la *Train Loss* (che continua a scendere) e la *Validation Loss* (che si stabilizza). Questo risultato chiave suggerisce che il training vincolato, pur aumentando l'accuratezza finale, porta il modello ad apprendere in modo troppo specifico i campioni del set di training, compromettendo la generalizzazione, specialmente per quanto riguarda le interazioni complesse delle classi foglia.



Comparazione delle Performance (Livello L3 Aggregato)

Il grafico di confronto aggregato mostra la performance media ponderata (Weighted Avg) delle singole pipeline dell'Approccio Ricorsivo (training indipendente) sul set di test.

Modello	**Precision (Weighted)**	**Recall (Weighted)**	**F1-Score (Weighted)**
Naive Bayes (NB)	**0.891**	**0.808**	**0.839**
Logistic Regression (LR)	0.781	0.685	0.699
Random Forest (RF)	0.750	0.654	0.676
SVM	0.709	0.546	0.555

Risultato Chiave: Nella pipeline Ricorsiva, il modello **Naive Bayes (NB)** è chiaramente il più performante, ottenendo un F1-Score (Weighted) di 0.839.

Approccio	**Modello Vincitore**	**Accuratezza/F1 (Chiave)**
Ricorsivo (Indipendente + CSP)	**Naive Bayes (NB)**	**F1-Score: 0.839**
Gerarchico (Cascata + Ensemble)	**Ensemble (SVM + NB)**	**Media Accuracy: 0.7762** (da K-Fold sul Training Set)

Capitolo 5: Ottimizzazione e Prospettive Future

Sebbene le strategie di *feature engineering* e *training* gerarchico abbiano massimizzato l'accuratezza dato il set di dati, l'analisi dei risultati ha evidenziato una limitazione strutturale legata alla **dimensione e variabilità del dataset**.

Criticità nel Test Set Limitato

Con un set di training di circa 1400 file e un set di test di soli 130 file, il problema di classificazione a grana fine (Livello L3, con 22 classi) non può essere risolto in modo ottimale.

La **variabilità insufficiente** dei documenti all'interno del test set ha avuto diverse conseguenze dirette:

- **Matrici di Confusione Sparse:** Molte classi L3 erano scarsamente rappresentate o assenti nel set di test, rendendo impossibile una valutazione statistica robusta della capacità del modello di generalizzare su quelle categorie.
- **Overfitting Persistente:** Il grande divario tra la **Train Loss (bassa)** e la **Validation Loss (alta)** è il sintomo classico di un modello che ha imparato a memoria i 1400 campioni di training, ma non ha visto abbastanza variazione per generalizzare efficacemente al piccolo set di test/validazione.
- **Matrici di Confusione Sparse:** Le Matrici di Confusione L3 appaiono sparse (molte caselle a zero) perché, per la maggior parte delle 22 classi, manca la rappresentazione nel set di test.
  È impossibile valutare con precisione la capacità di un modello di prevedere una classe se quella classe non è presente nei dati di valutazione.
- **Bias TF-IDF:** La rarità dei campioni rende il calcolo dell'IDF (Inverse Document Frequency) più sensibile a termini che appaiono solo in uno o due documenti, come 'soil' e 'cell', che diventano automaticamente i più rilevanti (IDF alto).

Necessità di Scala e Limiti Operativi

È fondamentale riconoscere che i risultati attuali sono una conseguenza diretta dei **limiti operativi** del progetto, in particolare la necessità di mantenere un **tempo di computazione ragionevole** e la limitazione delle **risorse hardware** (macchine personali) per l'elaborazione dei dati.
La scelta di non aumentare drasticamente il volume di file era dovuta a vincoli pratici.

Prospettiva di Ottimizzazione

In una futura ottimizzazione senza tali vincoli, il problema di Overfitting si mitigherà con l'aumento del volume e della variabilità del dataset, permettendo al modello di creare un legame più solido tra le curve di *Train* e *Validation Loss*.
Aumentando significativamente i dati di training (es. a **10.000** o più file), il modello avrebbe molta più esperienza per imparare le differenze sottili tra le classi L3.
Inoltre si dovrebbe migliore il bilanciamento**,** in modo tale che con l'aumento dei dati si distribuiscano meglio i campioni su tutte le 22 classi L3, e invece che eliminare le classi con un solo membro (come fa parzialmente l'approccio Ricorsivo con il filtro min_samples=5), cercare di includerle tutte e rendendo le Matrici di Confusione pienamente utili.

Prospettiva Funzionale: Implementazione dell'Ordinamento Fisico (SmartSort)

Il passo successivo e logico, in linea con l'obiettivo finale di *organizzazione intelligente* del progetto, è l'implementazione della funzione di **ordinamento e smistamento fisico** dei documenti.

Attualmente, il sistema si ferma alla predizione e alla generazione di un CSV con le etichette.
In futuro, lo script verrà esteso per integrare il modulo di *file system management* che gestirà dinamicamente i seguenti punti:

1. **Creazione Dinamica delle Directory:** Utilizzerà la categoria predetta a Livello L3 (la più specifica e finale) per creare nuove directory nel *file system* (es., una cartella per 'Alimentazione', una per 'AI_ML', ecc.).
2. **Smistamento Condizionale:** Sposterà (o copierà) i file originali all'interno della directory corrispondente al **risultato della predizione finale** (sia che provenga dall'Ensemble Gerarchico o dalla migliore pipeline Ricorsiva).
3. **Gestione del "Confidence Score":** Si potrebbe implementare un filtro, smistando i file solo se la probabilità di predizione L3 supera una soglia minima (es., 85%). I file con *score* inferiore verrebbero invece spostati in una cartella di revisione manuale denominata 'Altro' o 'Da Verificare', migliorando l'affidabilità del sistema automatico.

Capitolo 6: Conclusioni sul Confronto

Il modello Naive Bayes addestrato indipendentemente nell'approccio Ricorsivo ottiene il punteggio F1 più alto (0.839) sul set di test. Tuttavia, anche l'Ensemble dell'approccio Gerarchico ha dimostrato un'elevata accuratezza media K-Fold (0.7762), con la combinazione dei modelli che sistematicamente supera i modelli SVM e NB presi singolarmente.

Analisi del Feature Engineering (IDF)

Il grafico delle 25 parole più rilevanti (IDF) analizza la rarità dei termini utilizzati come feature.

- **Termini più Rari:** I termini con l'IDF più alto sono **'soil'** (>2.1), **'cell'**, e **'sample'**. Questi sono termini altamente specifici, probabilmente associati alle categorie 'Ambiente', 'Ecologia', 'Biologia', e 'Data_analysis', che sono cruciali per la classificazione L3.
- **Termini Generici/Tecnici:** Termini come 'based', 'analysis', 'model', e le date recenti ('2024', '2023') hanno un IDF più basso, indicando che sono più comuni nell'intero corpus e meno distintivi. La presenza di questi termini, sebbene meno rari, contribuisce comunque al contesto generale dei documenti.

Librerie Aggiuntive per la Valutazione

- **matplotlib.pyplot, seaborn**: Creazione di grafici statistici (barre, curve di loss) e heatmap professionali per le matrici di confusione.
- **sklearn.metrics**: Calcolo delle metriche di performance finali (Precision, Recall, F1-Score) e generazione delle matrici di confusione.
- **sklearn.linear_model.SGDClassifier**: Utilizzato specificamente nello script di diagnostica per generare la Curva di Loss (Train vs Validation).
- **Rdflib:** Utilizzato in metriche gerarchy.py per ricostruire i livelli L1 e L2 a partire dalla categoria L3 per la valutazione multilivello

Riferimenti Bibliografici

Apprendimento supervisionato: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3nd ed. Cambridge University Press [Ch.7]

Ontologie: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3nd ed. Cambridge University Press [Ch.16]

Basi di conoscenza: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3nd ed. Cambridge University Press [Ch.15]

Pianificazione con incertezza: D. Poole, A. Mackworth: Artificial Intelligence: Foundations of Computational Agents. 3nd ed. Cambridge University Press [Ch.12]