SmartSort: sistema di documentazione intelligente

Gruppo di lavoro

- Vito Stefano Birardi, 755782, v.birardi3@studenti.uniba.it
- Simone Columpsi, 758299, s.columpsi@studenti.uniba.it

SmartSmort - repository

AA 2024-25

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.

(L'agente costruisce relazioni tra i documenti, crea automaticamente cartelle e sottocartelle e vi inserisce i file correlati, anche se presentano estensioni differenti.)[Ancora non implementata]

Elenco argomenti di interesse

- **Apprendimento supervisionato e Incertezza**: Utilizzo di un modello di classificazione single-label per assegnare ai documenti le categorie più pertinenti con un relativo grado di confidenza.
- **Progettazione di Ontologie**: Definizione di una struttura ontologica per distinguere in modo formale tra "Risorse" (i documenti) e "Posizioni" (le cartelle), garantendo coerenza nella rappresentazione della conoscenza.

Strumenti utilizzati

L'implementazione si avvale di un ecosistema di librerie Python.

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.

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.

L'analisi esplorativa è supportata da:

- **Matplotlib**,
- **NumPy**

- **Seaborn**
- **WordCloud**
- **pickle**

Capitolo 1: Creazione del .csv e Estrazione e Pulizia del Testo

Descrizione delle funzioni principali

Creazione del file .csv

Il dataset di partenza viene costruito attraverso lo script `create_csv.py`, che esplora ricorsivamente una gerarchia di directory alla ricerca di file (in particolare PDF) dai quali estrarre i metadati fondamentali.
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 cartella `./References`

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, insieme al percorso relativo del file, in una struttura dati che viene poi scritta in un file `output.csv`. Questo file costituisce la base informativa per le fasi successive del progetto e consente di tener traccia in modo efficiente di tutte le risorse disponibili.

In un normale processo di estrazione dei metadati, viene seguito il seguente flusso di lavoro:

1. Un singolo file pdf viene aperto mediante la funzione `extract_metadata_from_pdf,` la quale si occupa di estrarre le varie informazioni dai metadati
2. Viene esplorato l'intero albero a partire dalla cartella `./References,` tramite la funzione `explore_folder_recursive`, con l'intento di individuare eventuali cartelle, all'interno delle quali potrebbero essere conservati altri file da classificare
3. Tutti i dati raccolti vengono scritti nel file `output.csv`, tramite la funzione `write_to_csv`

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.
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 espressioni regolari.

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 stopwords (parole non rilevanti come articoli o preposizioni) vengono rimosse, e le parole rimanenti vengono riportate alla loro forma base attraverso la lemmatizzazione.

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

Sono state inoltre calcolate feature descrittive del testo, come:

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

Queste feature statistiche saranno utilizzate nei moduli di categorizzazione e apprendimento automatico.

Librerie utilizzate

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

- **os**: Libreria standard di Python per interagire con il sistema operativo. 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**: Libreria fondamentale per l'interazione con i file PDF. 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**: Modulo standard di Python per la lettura e scrittura di file in formato 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.

Capitolo 2: Categorizzazione automatica dei documenti

Lo script categorize_all_files_updated.py prende come input un file CSV, tipicamente chiamato outputwithtext.csv, che contiene per ogni documento dati anagrafici come titolo, nome file, autore, anno, categoria iniziale, estensione, oltre al testo completo estratto e pulito.

L'obiettivo principale dello script è assegnare a ogni documento una sola categoria semantica, scegliendo la più specifica possibile all'interno di una gerarchia definita di categorie, organizzate per livello di specificità. Le categorie sono basate su un sistema di parole chiave (keyword) semantiche associate a ciascuna categoria.

Lo script agisce nel seguente modo:

- Legge tutto il CSV in un DataFrame pandas e sceglie casualmente una percentuale definita di questi file (ad es. tra l'80% e il 90%) per la categorizzazione, in modo da bilanciare efficacia ed efficienza
- Per ogni documento selezionato, la funzione findmostspecificcategory(row) esplora il testo e i metadati per individuare quante keyword associate a ciascuna categoria sono presenti.

  Le categorie sono gerarchizzate in gruppi:

  - Categorie molto specifiche (foglie ontologiche, es. AI_ML, Cardiologia, Archeologia antica)
  - Categorie specifiche (nodi intermedi nella gerarchia)
  - Categorie generali (rami principali)
  - Categoria fallback ('Altro' o categorie basate esclusivamente sull'estensione file)
- 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).
- 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.

Da allegare uno screenshot dell'output

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: categorizeallfilessinglecategorypercentage(percentage=85)

- **Input:**
    - percentage: percentuale di file da categorizzare (tra 80% e 90%, default 85%).
    - CSV di partenza letto da disco (outputwithtext.csv) contenente i dati testuali e metadati dei documenti.
- **Operazioni principali:**

1. Carica il dataset in un DataFrame pandas.
2. Filtra i file validi (con titolo non nullo).
3. Seleziona casualmente la percentuale specificata di file validi per la categorizzazione.
4. Per ogni file selezionato, utilizza la funzione findmostspecificcategory(row) per assegnare la categoria più specifica basata sulla presenza di keyword nel testo e nei metadati.
5. Se nessuna categoria viene assegnata usando le keyword, usa come fallback la categoria associata all'estensione del file. Se ancora nulla, assegna la categoria "Altro".
6. Salva il DataFrame aggiornato con la colonna category in un nuovo file CSV output, denominato in base alla percentuale scelta (es. ftrainingsetsinglecategorypercentage85.csv).
7. Stampa statistiche di riepilogo sulle categorie assegnate (distribuzione percentuale, livello di specificità).

Funzione chiave: findmostspecificcategory(row)

- **Scopo:**
  Determinare la categoria più specifica per un singolo documento.
- **Come funziona:**

1.1. Estrae il testo completo disponibile (titolo, filename, abstract, testo pulito) dal record.
1.2. Per ogni categoria definita in un dizionario di categorie e relative parole chiave, conta le occorrenze delle keyword nel testo (usando espressioni regolari per ricerche di parola intera).
1.3. Calcola uno score per ciascuna categoria basato sul numero di keyword trovate e il loro peso (lunghezza in parole).
1.4. Seleziona la categoria con punteggio più alto, dando priorità alle categorie più specifiche (definite come "very specific" o "specific") rispetto a quelle generali.
1.5. Se nessuna categoria con keyword positiva è trovata, assegna fallback basato sull'estensione del file (ad es. .py → AI_ML, .pdf → Altro).
1.6. Restituisce la categoria scelta.

Altri dettagli implementativi

- Lo script mantiene e usa liste e dizionari globali di categorie organizzate per specificità:
    - categoriesbyspecificityveryspecific: categorie molto specifiche (es. Informatica, Biologia).
    - categoriesbyspecificityspecific: categorie intermedie (es. Scienza).
    - categoriesbyspecificitygeneral: categorie generali (es. Altro).
    - extensioncategories: associazioni tra estensioni di file e categorie.

- La funzione esegue una selezione progressiva: prima cerca nelle categorie molto specifiche, poi se nulla, passa a categorie intermedie, poi generali e infine fallback.
- Vengono forniti messaggi di log che dettagliano il numero di file da categorizzare, quelli validi e la distribuzione finale delle categorie assegnate.

Decisioni di progetto

È stato deciso di assegnare una sola categoria per ciascun documento per motivi di chiarezza, gestione e interpretabilità della classificazione automatica.
Questo approccio semplifica i processi di analisi e consente di valutare in modo univoco l'appartenenza di ogni documento a un ambito specifico.

La scelta di assegnare la categoria più specifica possibile (la categoria "foglia" nell'ontologia) è motivata dal fatto che:

- Le categorie più specifiche forniscono un livello granulare di classificazione utile per analisi dettagliate e applicazioni mirate.
- La gerarchia ontologica organizza le categorie da generali a specifiche: assegnare una categoria più generale reprimerebbe informazioni preziose che descrivono più precisamente il contenuto del documento.

In sintesi, si punta a massimizzare la precisione semantica della classificazione, assegnando l'etichetta più dettagliata che i dati e il matching keyword permettono di determinare, mantenendo coerenza con la struttura gerarchica dell'ontologia.

Librerie utilizzate

Lo script categorize_all_files_updated.py utilizza queste 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
- random: per campionamento casuale dei file da categorizzare
- collections (defaultdict, Counter): per contare occorrenze e gestire dizionari di categorie e parole chiave

Queste librerie permettono allo script di leggere il CSV con i dati di partenza, effettuare analisi testuali sui campi di testo, associare categorie basate su keyword, e scrivere un CSV di output con le categorie assegnate.

Capitolo 3: Addestramento dei Modelli

Lo script `dataset_create_ontology_integrated.py` è responsabile dell'addestramento e della valutazione di modelli di classificazione multiclasse, partendo dal dataset di documenti precedentemente categorizzato.

I modelli coinvolti sono:

- Logistic Regression,
- Random Forest
- Support Vector Machines (SVM).

Questi algoritmi vengono addestrati su caratteristiche testuali vettorializzate mediante TF-IDF, arricchite da feature semantiche estratte dall'ontologia.

Il flusso di lavoro avviene nel seguente modo:

1. **Caricamento Dataset**
   Viene caricato il dataset categorizzato dal file CSV, che contiene per ogni documento colonne con testi puliti e la categoria assegnata (single-label).

2. **Caricamento e utilizzo dell'ontologia**
   Lo script carica un'ontologia OWL, cioè una Knowledge Base strutturata che contiene classi (categorie) e keyword associate (property hasKeyword nelle triple RDF). Questa ontologia definisce il dominio delle categorie, la gerarchia semantica e associa parole chiave a ciascuna categoria.

3. **Creazione di feature combinate**

   - Viene costruito un vettore TF-IDF sui testi, considerando n-grammi di 1 e 2 parole per catturare contesti.
   - Aggiunge feature semantiche basate su conteggi di keyword specifiche di ciascuna categoria, estratte grazie all'ontologia (es. frequenza delle parole chiave associate ad "Informatica" nel testo).
   - Queste feature sono combinate tramite un'operazione di concatenazione vettoriale (sparse matrix hstack).

4. **Bilanciamento del dataset**
   Per evitare squilibri nella distribuzione delle classi (categorie con molti più esempi di altre), si applica un under-sampling di quelle classi con troppi campioni, mantenendo un massimo di esemplari per ogni categoria bilanciando così il training set.

5. **Divisione Train/Test**
   Il dataset bilanciato viene diviso in train e validation set mantenendo la distribuzione delle classi (stratificazione).

6. **Addestramento dei modelli**

   - **Random Forest**: Modello ensemble di alberi decisionali. È scelto per la sua capacità di gestire feature miste e non lineari e robustezza a rumore e overfitting.

- **Logistic Regression**: Modello lineare efficiente, usato come base di confronto e per la sua velocità e interpretabilità.
- **SVM con kernel lineare/probabilità**: Scelto per la sua capacità di separare dati ad alta dimensionalità come quelli vettorializzati con TF-IDF e per offrire predizioni probabilistiche.

Ogni modello viene addestrato usando funzioni dedicate (trainevalsinglelabelmodel) che calcolano metriche di valutazione come accuratezza, F1 weighted e generano report dettagliati.

7. **Predizioni e salvataggio risultati**
   I modelli addestrati vengono testati sull'intero dataset (non solo il training set) per valutare la precisione finale, e le predizioni insieme alle confidenze di classificazione vengono salvate in file CSV.

Descrizione delle funzioni principali

Le principali funzioni utilizzate all'intenro dello script sono:

- **loadkeywordsfromontology**: carica le keyword per ogni categoria direttamente dall'ontologia OWL (in formato RDF) via SPARQL, sostituendo liste hardcoded.
- **loaddatasetwithsinglecategory**: carica il dataset CSV, associa a ogni riga la categoria più specifica tra quelle presenti, secondo l'ontologia.
- **createenhancedfeatures**: per ogni documento conta le occorrenze di keyword delle categorie (dall'ontologia) nel testo, creando feature semantiche che poi si combinano con TF-IDF.
- **balancesinglelabeldataset**: bilancia il dataset campionando uniformemente le classi con troppi esempi.
- **trainevalsinglelabelmodel**: addestra un classificatore (Random Forest, Logistic Regression o SVM) su train e valuta su validation set, restituendo le metriche.
- **predictsinglecategory**: genera predizioni e confidenze per tutto il dataset testato.

Decisioni di progetto

Motivazione delle scelte e confronto modelli

La scelta dei modelli copre un ampio spettro di complessità e tipologia di modelli:

- Il modello di Logistic Regression offre un baseline lineare efficiente, interpretabilità e rapidità.
- Random Forest, come modello ensemble, è in grado di catturare relazioni non lineari e interazioni tra le feature, risultando robusto a rumore e instabilità.
- SVM con opzioni di probabilità è particolarmente efficace in spazi vettoriali ad alta dimensionalità come quelli TF-IDF, e ottimizza il margine di separazione.

Questi modelli vengono confrontati attraverso metriche standard come accuratezza e F1 score ponderato su dati validation per selezionare il migliore in termini di generalizzazione e capacità predittiva.

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 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:

- Fornire un vocabolario semantico per creare feature avanzate basate su matching di keyword nei testi.
- Definire i livelli gerarchici delle categorie per la corretta interpretazione delle etichette nel dataset.

Librerie utilizzate

Per l'implementazione dello script dataset_create_ontology_integrated.py, è stato impiegato un insieme di librerie Python specializzate, ciascuna scelta per un compito specifico all'interno della pipeline di addestramento e valutazione.

- **pandas**: È stata utilizzata per caricare il dataset iniziale dal file CSV in un DataFrame, una struttura dati flessibile e potente che ha facilitato tutte le operazioni successive di manipolazione, filtraggio e aggiunta di nuove feature.
- **scikit-learn (sklearn)**
- **rdflib**: Utilizzata per interagire con la **Knowledge Base ontologica**. È stata indispensabile per caricare il file dell'ontologia (in formato OWL/RDF), navigare il grafo delle conoscenze ed estrarre dinamicamente le keyword associate a ciascuna categoria.
- **NumPy**: Ha operato "dietro le quinte" per gestire in modo efficiente gli array e le matrici sparse prodotte da TfidfVectorizer e utilizzate dai modelli di scikit-learn.
- **collections**: Modulo standard di Python che offre strutture dati specializzate. Nello script, sono state usate:
    - Counter: Per contare in modo estremamente efficiente le occorrenze delle keyword ontologiche all'interno dei testi, un passaggio chiave per la creazione delle feature semantiche.
    - defaultdict: Per aggregare facilmente le keyword per categoria dopo averle estratte dall'ontologia, semplificando la creazione di dizionari strutturati.
- **re**: Sebbene il grosso della pulizia del testo sia avvenuto in una fase precedente, re è stato utilizzato per operazioni di "pulizia al volo" e per la ricerca di pattern specifici richiesti durante la creazione delle feature.
- **textextract**: Questo è un **modulo personalizzato** del progetto, utilizzato per incapsulare funzioni riutilizzabili per l'elaborazione del testo.

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]