Documentazione RCSMovie

Alessandro Petruzzelli - Alessio Pagliarulo

Indice

1 Struttura Files **2**

2 Struttura Modelli **3**

3 Interfaccia generale **4**
3.1 Caricamento del modello 4
3.2 Ottenimento delle raccomandazioni 5
3.2.1 Raccomandazioni basate su preferenze esplicite 5
3.2.2 Raccomandazioni basate su proprietà 6
3.3 Raccomandazioni basate su preferenze implicite 7
3.3.1 Raccomandazioni basate su testo libero 7
3.4 Aggiornamento del dataset 8

4 Interfaccia Web Service **9**
4.1 Link 10
4.2 Tabella riassuntiva 11

1 Struttura Files

```
Dataset
    MovieInfo.csv
    tokenFilmsDatset.csv
Models
    Doc2Vec
        Doc2Vec.py
        doc2vec_model
    Word2Vec
        Word2Vec.py
        word2vec-google-news-300.bin
        word2vec_model
    FastText
        FastText.py
        cc.en.300.bin
        fasttext_model
        fasttext_model.trainables.vectors_ngrams_lockf.npy
        fasttext_model.wv.vectors_ngrams.npy
    TFIDF
        TFIDF.py
        dictionary_tfidf
        matrix_tfidf
        matrix_tfidf.index.npy
        tfidf_model
RSCCore.py
main.py
```

2 Struttura Modelli

Tutti i modelli sono indipendenti tra loro ma, per semplicità, sono realizzati tutti con la stessa struttura. All'interno di ogni file <nome_modello>.py troviamo tre funzioni principali:

- *create_model_<nome_modello>*. I parametri sono: *documents* ovvero i dati di addestramento del modello. In questo caso si deve trattare di una lista di documenti. I documenti sono, a loro volta, lista di parole. Il secondo parametro è il nome del modello ovvero il nome che il file che rappresenterà il modello (il modello TFIDF salva il modello, la matrice e il dizionario). Ogni modello ha i propri parametri di costruzione. Quelli che si trovano nel codice sono quelli da cui sono stai ottenuti i modelli presenti nella rispettiva cartella. Nel file FastText.py troviamo anche il metodo *create_model_fasttext_fb* il quale crea il modello partendo da un uno già salvato in memoria (File: cc.en.300.bin.). La prima riga di tale metodo è commentata volutamente poiché permette direttamente di scaricare lo stesso file da caricare qualora non sia presente nella cartella. (file da circa 7GB).

- *load_model*. Funzione che permette di caricare in memoria il modello. Se il modello non esiste lo crea. I parametri di questa funzione sono quindi gli stessi della funzione per la creazione del modello (i documents sono utilizzati per la creazione del modello per cui possono anche essere None). In tutte le funzioni troviamo il paramento queue di default valorizzato None. Tale parametro è stato aggiunto per permettere di caricare il modello in memoria in un thread separato. In questo caso il modello sarà salvato nella queue e sarà possibile recuperarlo da li.

  Nel caso del modello Word2Vec c'è un parametro aggiuntivo ovvero *preprained*: valore booleano che permette di scegliere se si vuole caricare un memoria un modello pre-addestrato (File: word-google-news-300.bin) o meno.
  Benché tale possibilità è prevista anche per il FastText la funzione di caricamento del modello fasttext permette di caricare solo il modello non preaddestrato. Questo perché il caricamento del modello pre-addestrato non è fornito dalla libreria *gensim* ma dalla libreria fastext (di facebook). Quindi se vogliamo caricare il modello fasttext preaddestrato dobbiamo invocare il metodo *create_model_fasttext_fb* (anche qui c'è il parametro queue)

- *get_recommendations_<nome_modello>*. Questo è il metodo che permette di ottenere le raccomandazioni. Questo modello è quello che ha il numero maggiore di parametri. Questa scelta è stata fatta per permette a questo metodo utilizzato in maniera indipendente dagli altri metodi. Se abbiamo bisogno di raccomandazioni basate su un modello ma non abbiamo interesse a mantenere il modello caricato in modello basta chiamare questo modello senza preoccuparci degli altri. Di fatti i parametri sono:
  - *token_string*: Le frasi che identificano le preferenze (plot di films o frasi scritte dall'utente) divise per token. In caso di più frasi abbiamo una lista di liste di token.
  - *documents*: gli stessi di sopra.
  - *titles* e *IDs*: Poiché questo sistema è stato pensato per essere utilizzato su un dataset di film, ogni film ha un titolo e un ID univoco. In questo caso sono per documents sono intesi i plot dei film (divisi in token) e titles e IDs sono i titoli e gli IDs dei film. Ovviamente le tre liste mantengono lo stesso ordine ovvero nel dataset il primo film ha ID = IDs[0], Title = titles[0], Plot = documents[0].
  - *model<nome_modello>*: Se il modello è già caricato in memoria possiamo comunque evitare di caricalo passandolo come parametro. In alternativa è None e quindi viene caricato (o creato).
  - *prefIDs*: Se le preferenze dell'utente sono dei film presenti nel dataset allora si possono indicare anche gli ID di tali film al fine di non avere come suggerimento proprio il film oggetto della preferenza.
  - *pretrained* (Word2Vec e FastText): Se il modello non è caricato si decide che modello caricare.
  - *most_similar* (Doc2vec): Il Doc2Vec può dare risultati in due modi. La prima è con il calcolo del centroide (calculate_centroid altro metodo presente in Doc2Vec, FastText e Word2Vec). La seconda è attraverso il metodo *most_similar* che è un metodo messo a disposizione dal modello stesso. Il valore di questo parametro è, quindi un boolean. True per utilizzare il metodo *most_similar* (meno preciso), False per il centroide.

3 Interfaccia generale

L'interfaccia per utilizzare tutti i modelli senza entrare nei delle implementazioni è descritta dal File *RSCCore.py*. Questo file, poiché è stato pensato come interfaccia delle implementazioni, è stato scritto rispettando gli standard di python ovvero: le variabili o i metodi che dovrebbero essere privati sono definiti __<nome>__.

3.1 Caricamento del modello

Per garantire un'ottimizzazione dei tempi il caricamento del modello in memoria è separato dall'ottenimento delle raccomandazione. Ovviamente, il tutto è gestito con delle variabili globali che permettono, in ogni punto del programma, di mantenere in memoria le variabili da utilizzare. Per caricare il modello stato implementato il metodo: *select_model*. Questo metodo prende in input un solo parametro (*selected_model*) che può essere valorizzato:

1. per usare Doc2Vec con il metodo most_similar.
2. per usare Doc2Vec la similarità è con il centroide.
3. per usare Word2Vec per utilizzare un modello pre-addestrato.
4. per usare Word2Vec.
5. per usare FastText per utilizzare un modello pre-addestrato.
6. per usare FastText.
7. per usare TFIDF.

Se il valore è uno di quelli sopra indicati allora sarà eseguito il codice che avvia un thread e carica in memoria il modello scelto. Questo avviene per non "fermare" l'algoritmo chiamante. Il codice eseguito è il seguente:

```
__returned_queue__ = queue.Queue()
thread = threading.Thread(target=__ft__.load_model, args=(__tokenized_plots__,
                                                          "Models/FastText/fasttext_model",
                                                           __returned_queue__))
```

Poiché l'operazione viene eseguita in maniera asincrona il valore di ritorno di questa funzione sta a significare se il modello è stato correttamente riconosciuto (e non caricato). Possono essere restituiti:

- 200: per segnalare che il valore di *selected_model* è corretto e che il thread per il caricamento è partito;
- 404: per segnalare che il valore non è riconosciuto

Prima di procedere al caricamento del modello il metodo verifica se sono caricate in memoria relative al film. Si tratta dei plots (divisi in token), dei titoli e degli ID. Questi vengono caricati dal File *tokenFilmsDataset.csv*. Questo file è un csv il quale presenta sole tre colonne: *Tokens*, *ID* e *Title*. Questo implica che si potrebbe facilmente lavorare su un altro dataset: basterebbe formattare i dati in questo modo.

Come detto sopra, il modello e tutti i parametri ad esso collegati (il numero del modello selezionato, se si vuole usare il modello preaddestrato, il metodo most_similar, ...) sono salvati in variabili globali. Questo permette di controllare, quando viene chiamato il metodo *select_model* di controllare se il modello scelto è già caricato in memoria. In questo modo non c'è bisogno di ricaricalo, ottimizzando le risorse. Questa cosa è fatta nel seguente modo:

```
global __doc2vec__, __most_similar__
   try:
       if __doc2vec__ is not None:
           print("Already Loaded")  # Si evita di ricaricare il modello
       else:
           raise Exception
```

```
except Exception:
    __doc2vec__ = None
    __returned_queue__ = queue.Queue()
    ...
```

Il tutto è gestito con delle eccezioni poiché se tentiamo di leggere delle variabili globali a cui non è stato assegnato nessun valore in precedenza queste non sono semplicemente None ma l'operazione solleva una eccezione. Poiché controllo che la variabile del modello non sia None, se lo è (quindi non si solleva l'eccezione) sollevo l'eccezione manualmente lasciando la gestione al blocco subito sotto.

3.2 Ottenimento delle raccomandazioni

3.2.1 Raccomandazioni basate su preferenze esplicite

L'altro metodo esposto da questa interfaccia è *get_suggestion*. Con questo metodo è possibile avere delle raccomandazioni basate su preferenze per elementi presenti nel dataset. Poiché gli elementi del dataset sono distinti da un ID per ottenere le preferenze basta specificare gli ID di tali elementi. Le preferenze possono essere espresse per i film o per le singole proprietà dei film (un attore, un regista o un genere). I parametroi di questo metodo sono *preferences_IDs* e *pref_entity*. I parametri **devo** essere una lista di stringhe (ID), questo anche se la preferenza è singola (una delle due liste può essere anche vuota, in questo modo saranno valutati solo gli elementi della lista "valorizzata").

Raccomandazioni basate su film
Dagli ID dei film specificati vengono poi valorizzati gli array in cui sono contenuti i plot dei film "preferiti". Se non è mai stato chiamato il metodo *select_model* allora non sono mai state caricate in memoria le informazioni dei film; quindi, in questa fase non sarà possibile trovare il plot del film corrispondente all'ID. Per questo motivo questa funzione potrebbe restituire il valore 400 Se tutto è andato a buon fine allora si passa a chiedere le raccomandazioni. Per fare ciò si usa il metodo *get_recommendations_<nome_modello>* (in base al modello scelto) del singolo modello di cui viene valorizzato anche il parametro *model*. Poiché il modello è stato caricato in un altro thread, il risultato viene letto dalla queue (parametro nel metodo di caricamento). Per fare questo si usa:

```
__doc2vec__ = __returned_queue__.get()
```

La chiamata del metodo *get* sulla queue, se il caricamento non è ancora terminato, rimane in attesa della "chiusura" del thread. Quindi, dopo questa chiamata, il modello sarà sempre caricato.

La parte che genera preferenze basandosi solamente sui film è stata "perfezionata" per poter permettere una maggiore attendibilità dei risultati. In particolare il risultato della similarità tiene conto degli aspetti quali attori, generi e registi che i film da suggerire condividono con i film "preferiti". In particolare, definendo:

- $C_{t,d}$ come l'insieme degli attori che recitano in t e d. Formalmente, $C_{t,d} = Cast_t \cap Cast_d$;
- $D_{t,d}$ come l'insieme dei registi che hanno diretto t e d. Formalmente, $D_{t,d} = Directors_t \cap Directors_d$
- $G_{t,d}$ come l'insieme dei generi in cui rientrano t e d. Formalmente, $G_{t,d} = Genres_t \cap Genres_d$

Il film suggerito, in questo caso, non è solamente quello suggerito dal modello ma, per ogni film viene effettuato il secondo calcolo:

$$sim_{t,d} = cos\_sim_{t,d} + (|C_{t,d}|\alpha + |D_{t,d}|\beta + |G_{t,d}|\delta) \cdot \overline{cos\_sim} \tag{1}$$

In cui

- $cos\_sim_{d,t}$ è il valore del coseno di similarità calcolato precedentemente;

- α, β, δ rappresentano il *peso* che si vole attribuire ad ogni entità che i due film hanno in comune. α rappresenta il peso che ogni attore in comune, β di ogni regista e δ di ogni genere.[1]

- $\overline{cos\_sim}$ è la media del valore del coseno di similarità dei primi N film ritenuti più simili dai modelli. Questo parametro permette di non attribuire un peso costante alle entità ma di "vincolare" il loro peso al valore del coseno di similarità. Questa scelta è stata effettuata perché possono presentarsi casi in cui il valore del coseno della similarità è più basso del peso delle entità. In questi casi, senza l'introduzione di questo parametro, la raccomandazione sarebbe data quasi unicamente dalla valutazione delle entità, snaturando la natura del recommender. Con una formula:

$$\overline{cos\_sim} = \sum_{i=1}^{N} \frac{cos\_sim_{d,t_i}}{N}$$

3.2.2 Raccomandazioni basate su proprietà

Le raccomandazioni basate su entità non prevedono l'utilizzo di nessun modello ma vanno ad incrementare (in percentuale) il valore di similarità del film che "contiene" tale entità ($new\_sim = 1.4 * old\_sim$). Questa fase è successiva a quella che valuta le preferenze sui film e il testo libero perché, se sono previste, il valore di similarità di partenza sarà dal somma della similarità delle fasi precedenti. Se le fasi di prima non sono previste (non si specificano film o frasi libere) allora il valore di similarità di base per tutti i film è 0.1. In questo caso il film suggerito è quello che "contiene" più entità di quelle specificate.

[1] Attualmente:
$\alpha = 0.15$
$\beta = 0.20$
$\delta = 0.25$

3.3 Raccomandazioni basate su preferenze implicite

3.3.1 Raccomandazioni basate su testo libero

Le raccomandazioni su testo libero sostituiscono il vecchio metodo */getSuggestionsFromSentence*. Si possono indicare più frasi che vengono inizialmente pre-processate con l'algoritmo di NLP fornito dalla libreria *spaCy*. In questa fase vengono rimosse le stops words, vengono rimossi i segni di punteggiatura e vengono rimosse le parole come "movie", "film", "like", "love", etc.

Ulteriore parametro del metodo è *evaluate_sim_word* ovvero una variabile booleana che permette di decidere se applicare la tecnica di espansione o meno. Se si decide di applicarla per ogni parola estratta dal processo di NLP vengono trovati 5 sinonimi. In questo modo possiamo far "pesare" di più questa preferenza "ripetendo" il concetto con sinonimi. A questo punto ogni parola (con la sua lista di sinonimi) è trattata come una trama e quindi "data in pasto" al modello selezionato.

3.4 Aggiornamento del dataset

La funzione *update_dataset* permette di aggiornare il dataset interno (File MovieInfo.csv) composto solo dall'ID, titolo, plot (diviso in tokens), cast, registi e generi. Permettere completare questa operazione la funziona necessita dei tre parametri. Come nella scelta del modello il primo passo è controllare se sono caricati in memoria le informazioni dei film. Se non sono caricate, si provvede ad caricamento. Questa funzione non permette solo di aggiungere nuovi elementi ad dataset ma anche di aggiornare i presenti. Per questo si effettua un controllo sull'ID. Se l'ID passato come parametro è già associato ad un film allora gli altri parametri (titolo, plot, cast, registi e generi) vanno a sostituire quelle associate a tale array. In alternativa il nuovo film viene aggiunto a quelli caricati in memoria. Infine l'update avviene anche sul file.[2] Se l'operazione è andata a buon fine il metodo restituisce il valore 200 altrimenti, se il file non è stato aggiornato, è restituito il file 400.

[2]Il Doc2Vec con il *most_similar* non suggerisce mai il nuovo film. Questo perché il documento non è aggiunto a quelli "conosciuti" dal modello. Si consiglia, se sono aggiunti tanti films, di eliminate i modelli creati (esclusi i pretrained) e di chiamare la funzione di load per ricrearli

4 Interfaccia Web Service

L'interfaccia del web service è stata realizzata in con *flask* e prevede solo due indirizzi raggiungibili:

- <indirizzo>/**selectModel**/<**int:selected_model**>: il metodo della richiesta è GET. dopo l'ultimo "/" va inserito il numero del modello che si intende utilizzare (vedi 3.1). La stringa di ritorno può essere:
  - "OK": se il caricamento è cominciato correttamente;
  - "No model loaded": se non è possibile effettuare il caricamento (probabilmente il numero non rientra tra quelli accettati)
- <indirizzo>/**getSuggestions**: in questo caso va effettuata una richiesta POST. Alla chiamata va aggiunto un oggetto JSON contente "movie" valorizzato con la lista di ID "preferiti" e "entities" contenente gli ID per le proprietà.
  Questo metodo "risponde" inviando i dati con la seguente struttura JSON:

  ```
  {
  "results": [
  {"ID": "Q1141420", "Rank": 1 },
  {"ID": "Q565623", "Rank": 2 },
  {"ID": "Q17619306", "Rank": 3 },
  {"ID": "Q2737011", "Rank": 4 },
  {"ID": "Q467516", "Rank": 5 }]
  }
  ```

  O in caso di errore nella ricerca dei suggerimenti (probabilmente non è stato selezionato nessun modello):

  ```
  {"results": "400"}
  ```

  L'implementazione di questa interfaccia comporta due notevoli vantaggi:

  1. É possibile caricare il modello con una semplice chiamata al servizio. Questo può avvenire in qualsiasi momento.
  2. Una volta caricato il modello possiamo "chiamare" più volte il metodo per ottenere le raccomandazioni. Questo, poiché il modello è già caricato, ci risponderà in pochissimi secondi.
- <indirizzo>/**updateDataset**: per aggiornare il dataset aggiungendo un altro film o sostituire le informazioni associate ad un ID. Si tratta di una richiesta POST a cui vanno "allegati" dati in formato JSON che devono rispettare il seguente standard:

  ```
  {
  "plot":"Cobb, a skilled thief who commits corporate ...",
  "title":"Inception",
  "ID":"123"
  "cast":['Q8927', 'Q38111', 'Q123351', 'Q173399', 'Q177311']
  "genres":['Q188473', 'Q319221', 'Q471839', 'Q496523', 'Q2484376']
  "directors":['Q25191']
  }
  ```

  In alternativa si riceve una stringa che avvisa dell'errore *"Format Error"*.
  Se l'ID è associato già ad un film allora di questo saranno aggiornati Titolo e Plot. Se l'ID è "nuovo" allora sarà inserito nella lista dei film con le informazioni ad esso associato.
  Dopo la chiamata a questo metodo non c'è bisogno di ricaricare il modello per avere raccomandazioni.

- <indirizzo>/**getSuggestionsFromSentence**: Questo metodo permette di ottenere delle raccomandazioni basandosi su frasi data in input e non sulle preferenze. L'utilizzo consiste in una chiamate utilizzato il metodo POST. L'oggetto JSON da inviare deve contente la lista di frasi come valore dell'elemento "sentences" e un valore booleano associato a "expand". I risultati e gli errori sono gli stessi del metodo <indirizzo>/**getSuggestions**.

4.1 Link

Il codice del progetto è possibile trovarlo qui[3]
I modelli per ragioni di spazio non sono caricati nel repository, vanno scaricati separatamente e spostati nella cartella corrispondente. Si possono scaricare da qui[4]
Tutti i test effettuati sui modelli sono raccolti e disponibili al seguente link[5]

[3]https://github.com/petruzzellialessandro/RCSmovie
[4]https://drive.google.com/file/d/18aksc5-er653KZcLHBpunAh-Cibuv8X7/view?usp=sharing
[5]https://drive.google.com/drive/folders/1hT1pHs2BDf3KeqJTBmodNUBkpjEKZN5G?usp=sharing

4.2 Tabella riassuntiva

Nome metodo	Input Accettato	Output atteso
/selectModel/ <**int:selected_model**> [GET]	Se si usano i valori elencati in 3.1	"OK": se l'input è accettato "No model loaded": se l'input non è accettato.
/**getSuggestions** [POST]	JSON contente i campi "movie", "entities"	{"results": [{"ID":..., "Rank":...},...] }: lista di 5 film identificati da ID e rank. {"results":400}: se nessun modello è caricato o nella lista c'è un ID non riconosciuto.
/**updateDataset** [POST]	Oggetto JSON con attributi chiamati "plot", "title", "ID", "cast", "genres", "directors"	"OK": se l'input è accettato e l'update è andato a buon fine. "Dataset not updated": se l'input è accettato e l'update non è andato a buon fine. "Format Error" se l'input non è accettato.
/**getSuggestionsFromSentence** [POST]	JSON contenente "sencences" (lista di frasi) e un booleano "expand"	{"results": [{"ID":..., "Rank":...},...] }: lista di 5 film identificati da ID e rank. {"results":400}: se nessun modello è caricato o nella lista c'è un ID non riconosciuto.

Tabella 1: Interfaccia Web Service

Metodo	Input	Output
/selectModel/**1** [GET]		"OK"
/selectModel/**10** [GET]		"No model loaded"

/**getSuggestions** [POST]	{
	"movies":[
	"Q25188","Q46551"],
	"entities":[
	"Q25191"],
	}
		{
		"results": [
		{"ID": "Q1141420", "Rank": 1 },
		{"ID": "Q565623", "Rank": 2 },
		{"ID": "Q17619306", "Rank": 3 },
		{"ID": "Q2737011", "Rank": 4 },
		{"ID": "Q467516", "Rank": 5 }]
		}

/**getSuggestions** [POST]	[Q102438, Q27894574, Ciao]
	o
	Nessun modello caricato
		{
		"results": "400"
		}

/**updateDataset** [POST]	{
	"plot":"Cobb, a skilled thief who commits corporate ...",
	"title":"Inception",
	"ID":"123"
	"cast":['Q8927', 'Q38111', 'Q123351', 'Q173399', 'Q177311']
	"genres":['Q188473', 'Q319221', 'Q471839', 'Q496523', 'Q2484376']
	"directors":['Q25191']
	}
		"OK": se l'input è accettato e l'update è andato a buon fine.
		
		"Dataset not updated": se l'input è accettato e l'update non è andato a buon fine. Es. non c'è o è corrotto il file del dataset

/**updateDataset** [POST]	{
	"plot":"Cobb, a skilled thief who commits corporate ...",
	"title":"Inception",
	"title":"123"
	
	}
		"Format Error"

/**getSuggestionsFromSentence** [POST]	{
	"sentences":["I like film with aliens", "I like movie about enigma", ...],
	"expand":True,
	}
		{
		"results": [
		{"ID": "Q1141420", "Rank": 1 },
		{"ID": "Q565623", "Rank": 2 },
		{"ID": "Q17619306", "Rank": 3 },
		{"ID": "Q2737011", "Rank": 4 },
		{"ID": "Q467516", "Rank": 5 }]
		}

Tabella 2: Esempio Web Service