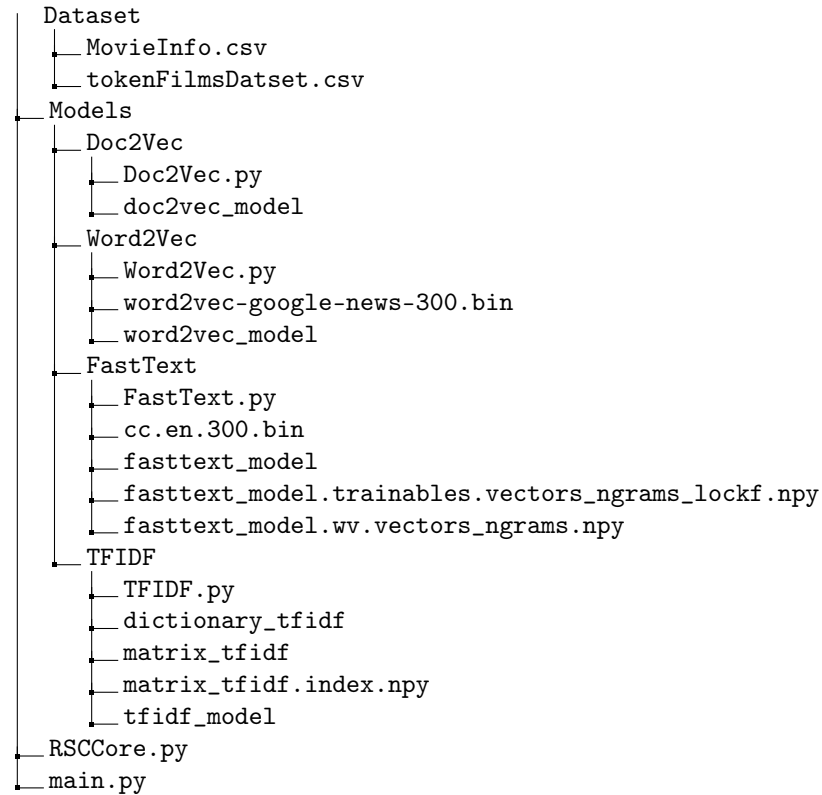


Documentazione RCSMovie

Alessandro Petruzzelli

1 Struttura Files



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 stati 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 parametro *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 lì.

Nel caso del modello Word2Vec c'è un parametro aggiuntivo ovvero *pretrained*: valore booleano che permette di scegliere se si vuole caricare in 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 preaddestrato non è fornito dalla libreria *gensim* ma dalla libreria *fasttext* (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 permettere 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 memoria basta chiamare questo modello senza preoccuparci degli altri. Di fatti i parametri sono:
 - *token_string*: Le frasi che identificano le preferenze (plot di film 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 ID 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 caricarlo 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.

Il metodo restituisce una lista di 5 elementi costruita nel seguente modo:

```
recommend_movies.append({"Rank": rank, "ID": IDs[i]})
```

Nella parte finale del metodo (in ogni modello) è commentata la parte che permette di stampare i risultati.

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 *tokenFilmDataset.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 ricaricarlo, 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

Raccomandazioni basate su preferenze

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. Per questo l'unico parametro di questo metodo è *preferences_IDs*. Il parametro **deve** essere una lista di stringhe (ID), questo anche se la preferenza è singola. Dagli ID 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.

Infine le raccomandazioni vengono fornite nella stessa forma restituita dal metodo invocato sul modello ovvero: una lista di (5) set contente un Rank e un ID.

Raccomandazioni basate su preferenze

Con il metodo *get_suggestion_from_sentence* è possibile ottenere i film raccomandati basandosi su una frase data in input (parametro *sentence*). Questo metodo non è altamente complesso poiché è molto simile a quello basato sulle preferenze. Questo è possibile per come è stata effettuata l'implementazione del metodo privato *__get_rec__* e dei vari modelli. Infatti tutti i modelli nel metodo *get_recommendations_<nome_modello>* prendono come parametro *token_string*. Nel caso delle raccomandazioni basate sulle preferenze questo parametro si "popola" con la lista dei token che rappresentano i plot dei singoli film "preferiti". In questo caso tale parametro rappresenta proprio la lista dei token in cui è divisa la frase dell'utente. Sono quindi restituite le raccomandazioni nella stesso formato usato per ottenere quelle basate sulle preferenze ovvero: una lista di (5) set contenente un Rank e un ID.

3.3 Aggiornamento del dataset

La funzione *update_dataset* permette di aggiornare il dataset interno (File MovieInfo.csv) composto solo dall'ID, titolo e plot (diviso in tokens). Permettere completare questa operazione la funzione 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 al caricamento. Questa funzione non permette solo di aggiungere nuovi elementi al 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 e plot) 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.¹

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.

¹Il 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 eliminare 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 vanno aggiunti dei dati. Si tratta di dati JSON che rappresentano la la lista di preferenze dell'utente (solo ID).

Questo metodo "risponde" inviando i dati con la seguente struttura JSON:

```
{
  "results": [
    {"ID": "Q133654", "Rank": 1 },
    {"ID": "Q5182815", "Rank": 2 },
    {"ID": "Q205211", "Rank": 3 },
    {"ID": "Q16250686", "Rank": 4 },
    {"ID": "Q1304560", "Rank": 5 }
  ]
}
```

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

```
{"results": 401}
```

L'implementazione di questa interfaccia comporta due notevoli vantaggi:

1. È 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"
}
```

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 associate.

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 una frase data in input e non sulle preferenze. L'utilizzo consiste in una chiamata utilizzando il metodo POST. Come "data" va "allegata" la frase per cui cercare suggerimento (no JSON, semplicemente stringa). I risultati e gli errori sono gli stessi del metodo `<indirizzo>/getSuggestions`.

4.1 Link

Il codice del progetto è possibile trovarlo qui²

I modelli per ragioni di spazio non sono caricati nel repository, vanno scaricati separatamente e spostati nella cartella corrispondente. Si possono scaricare da qui³

²<https://github.com/petruzzellialessandro/RCSmovie>

³<https://drive.google.com/file/d/18aksc5-er653KZcLHBpunAh-Cibuv8X7/view?usp=sharing>

4.2 Tabella riassuntiva

Nome metodo	Input Accettato	Output atteso
<code>/selectModel/ <int:selected_model></code> [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.
<code>/getSuggestions</code> [POST]	Lista di ID di film	<code>{"results": [{"ID":..., "Rank":...},...] }</code> : lista di 5 film identificati da ID e rank. <code>{"results":400}</code> : se nessun modello è caricato o nella lista c'è un ID non riconosciuto.
<code>/updateDataset</code> [POST]	Oggetto JSON con attributi chiamati "Plot", "Title", "ID"	"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.
<code>/getSuggestionsFromSentence</code> [POST]	JSON contenente una frase (senza chiave)	<code>{"results": [{"ID":..., "Rank":...},...] }</code> : lista di 5 film identificati da ID e rank. <code>{"results":400}</code> : 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]	[Q102438, Q27894574, Q274167]	{ "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" }	"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", }	"Format Error"
/getSuggestionsFromSentence [POST]	After the terrorist attack on the World Trade Center on September 11, 2001, the building collapses over the rescue team from the Port Authority Police Department. Will Jimeno and his sergeant John McLoughlin are found alive trapped under the wreckage while the rescue teams fight to save them.	{ "results": [{"ID": "Q1141420", "Rank": 1 }, {"ID": "Q565623", "Rank": 2 }, {"ID": "Q17619306", "Rank": 3 }, {"ID": "Q2737011", "Rank": 4 }, {"ID": "Q467516", "Rank": 5 }] }
/getSuggestionsFromSentence [POST]	NESSUN MODELLO CARICATO After the terrorist attack on the World Trade Center on September 11, 2001, the building collapses over the rescue team from the Port Authority Police Department. Will Jimeno and his sergeant John McLoughlin are found alive trapped under the wreckage while the rescue teams fight to save them.	{ "results": "400" }

Tabella 2: Esempio Web Service