**Relazione Progetto 2**

*Giovanni Sacco, matricola 451627*

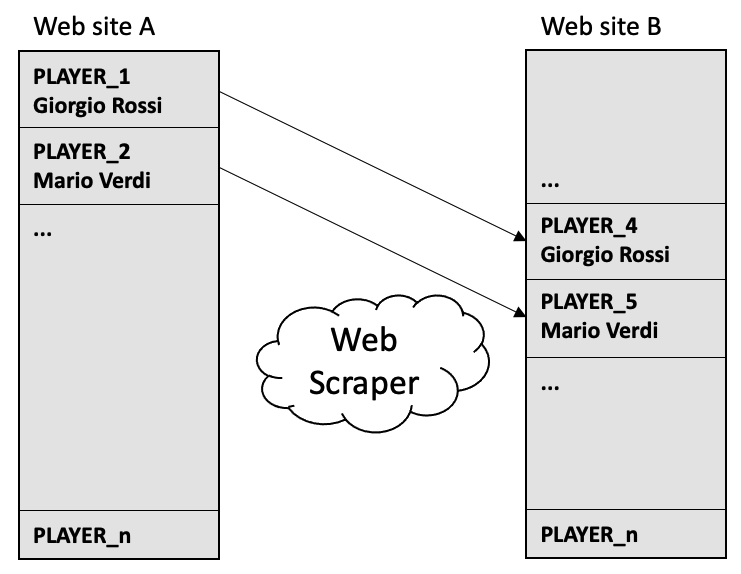
**Definizioni preliminari**

* **Entità**: entità del mondo reale di un tipo particolare. Ad ex un’entità può essere un pilota di formula uno, oppure una persona, o un’automobile etc.. Le entità sono divise in tipi, sulla base delle caratteristiche contenute;
* **Entity-page:** pagina web che pubblica dati che rappresentano un’entità di un tipo particolare. Si tratta della forma più comune di dato strutturato disponibile sul web. Parte di loro vengono generate dinamicamente popolando modelli fissi di HTML con contenuti letti da un DBMS e sono dette *template-based*.

**1 Descrizione del problema**

Il problema affrontato dal seguente progetto è quello di individuare coppie di entity-page, rappresentanti una stessa entità, ricavate da due fonti di dati - siti web - differenti.

Date quindi in input due liste di *entity-page,* relative a giocatori dell’NBA, ricavate rispettivamente da due differenti siti web, si vuole creare un web scraper in grado di individuare il numero maggiore di coppie di *entity-page*, appartenenti alle due diverse liste, rappresentanti lo stesso giocatore.



**2 Descrizione della soluzione**

Si procede ora alla descrizione della soluzione implementata:

* Il capitolo 2.1 è incentrato sugli aspetti tecnici della soluzione ed effettua un veloce focus sulle librerie utilizzate;
* Il capitolo 2.2 si concentra sulla logica e sull’implementazione della soluzione;
* Il capitolo 2.3 descrive i casi di test effettuati ed i relativi risultati.

**2.1 Linguaggio e librerie utilizzate**

Per l’implementazione del web scraper ho optato per l’utilizzo di NodeJS e, in particolare, per l’utilizzo delle seguenti librerie:

* *Cheerio*: libreria utile alla raccolta dei dati ricavati da un sito web. Con tale libreria è possibile manipolare i dati ricavati da un sito web, filtrarli ed organizzarli in apposite strutture dati;
* *Puppeteer*: libreria utile ad effettuare il download di tutti i contenuti di un dato sito web. La caratteristica principale che mi ha convito ad utilizzare tale soluzione è quella di consentire il corretto download dei dati di un sito web sul quale i contenuti vengono caricati in modo asincrono, tramite chiamate http successive al caricamento della pagina;
* *Stopword*: libreria utile ad eliminare da un testo tutti i termini non significativi;
* *Tf-idf-search*: libreria che, a partire da due testi, tramite l’implementazione di *TF-IDF* e della *Cosine Similarity* rileva l’indice di somiglianza tra di essi.

**2.2 Idea alla base della soluzione**

Selezionati due siti web ho creato, per ciascuno di essi, un array contenente circa 100 url di *entity-page* rappresentanti giocatori di basket.

Per la creazione di tali array ho seguito la seguente condizione:

* L’intersezione tra i due array deve possedere circa 50 *entity-page* in comune. Quindi le due liste di url devono avere circa 50 url rappresentanti lo stesso giocatore di basket.

La logica del sistema segue i seguenti step:

1. Download, tramite la libreria *Puppeteer* di tutti i contenuti presenti nelle pagine dei due array:
   1. *Input*: due array di url di *entity-page*;
   2. *Output*: strutture dati contenenti ciascuna tutti i contenuti presenti nel body di ogni pagina web associata alle url in input;
2. A partire dai contenuti ottenuti per ogni pagina dallo step precedente, si effettua l’estrazione, tramite la libreria *Cheerio*, e salvataggio dei contenuti maggiormente significativi, pagina per pagina, in due array separati, uno per sito web di input. L’obiettivo di tale step è quindi quello di estrapolare da ogni pagina i contenuti maggiormente significativi andando quindi ad effettuare un’importante operazione di filtraggio. Tale filtro viene effettuato tramite le seguenti regole:
   1. Ignora da ogni pagina tutti quei contenuti che sono identici tra tutte le pagine appartenenti allo stesso sito web. Si tratta quindi di contenuti appartenenti al template della pagina e non al contenuto in essa rappresentato;
   2. Ignora le sezioni di un sito web che sicuramente non hanno informazioni significative (ex footer, heder);
   3. Ignora le “stop word”, tramite la libreria *Stopword*.

Tale step, quindi, restituisce due array di strutture dati contenenti ciascuna le informazioni, in forma testuale, maggiormente significative di ogni pagina web.

1. A partire dai due array ricevuti dallo step precedente, si procede con la ricerca delle coppie di *entity-page* rappresentanti la stessa entità. Per l’individuazione delle coppie ho utilizzato l’unione di due strategie:
   1. Calcolo del grado di somiglianza tra due pagine tramite la libreria *Tf-idf-search.* Ciò è stato possibile in quanto nello step precedente le informazioni maggiormente significative di ogni pagina sono state estratte in forma testuale. Quindi per verificare il grado di somiglianza tra i contenuti di due pagine è stato sufficiente sottomettere le rispettive informazioni testuali così ricavate alla libreria TF-IDF;
   2. Calcolo del grado di somiglianza tra le parti variabili delle url tramite la libreria *Tf-idf-search.*
2. Al termine di questi step, vengono prese in considerazione solo le coppie di pagine web che possiedono un grado di similarità (pari alla somma dei gradi di similarità presentati nello step 3-a e 3-b) maggiore di 0.8. Tale valore di soglia è stato individuato come il migliore in tutti i test effettuati.

Di seguito si riporta per completezza la descrizione della logica funzionale del sistema in pseudocodice:

***Input****: due liste di entity-page, L1, L2, rappresentanti giocatori di basket reperite, rispettivamente, da due siti web differenti.*

***Output****: una lista di coppie di pagine web, appartenenti rispettivamente ai due differenti siti web, rappresentanti lo stesso giocatore di basket*

//Array che conterranno le liste dei contenuti estratti dalle singole pagine

*firstDataSetPagesContent = [];*

*secondDataSetPagesContent = [];*

//Per ogni dataset estraggo, filtro e salvo i contenuti di ciascuna pagina

// NB I due cicli for vengono eseguiti in parallelo asincronicamente

*for (urli in L1)*

*pageContent = downloadAllContent(urli);*

*pageFilteredContent = filterContent(pageContent);*

*firstDataSetPagesContent.push(pageFilteredContent);*

*for (urli in L2)*

*pageContent = downloadAllContent(urli);*

*pageFilteredContent = filterContent(pageContent);*

*secondDataSetPagesContent.push(pageFilteredContent);*

*pageSimilarityList = [];* // Contiene la lista di coppie pagine simili

//Calcola il grado di somiglianza tra le possibili coppie di pagine

*for (firstDatasetPageContenti in firstDataSetPagesContent)*

*for (secondDatasetPageContentj in secondDataSetPagesContent)*

*contentSimilarity = tf\_idf(firstDatasetPageContenti, secondDataSetPagesContent);*

*urlSimilarity = tf\_idf(firstDatasetPageContenti.url, secondDataSetPagesContentj.url);*

//Se la somma tra il grado di similarità per contenuto e quello per url è //maggiore di 0.8, aggiungiamo la coppia tra quelle indicate come //rappresentanti lo stesso giocatore

*if ( contentSimilarity + urlSimilarity > 0.8 )*

*pageSimilarityList.push({*

*firstPage: firstDatasetPageContenti.url,*

*secondPage: secondDataSetPagesContentj.url*

*});*

*return pageSimilarityList,*

Come si può evincere dall’agoritmo sopra descritto, la complessità è pari a **O(max(N,M) + N\*M)**, con N ed M pari rispettivamente alla cardinalità di L1 ed L2:

* O(max(N,M)): dovuto ai primi due cicli for eseguiti parallelamente all’avvio per effettuare il download di tutte le pagine web associate alle url ricevute in input;
* O(N\*M): dovuto ai due cicli for annidati presenti nella sezione dedicata al calcolo della similarità tra le pagine.

**2.3 Casi di studio e risultati ottenuti**

Come test del modello creato ho utilizzato 300 url di pagine web rappresentanti giocatori di basket dell’NBA ricavate da tre differenti siti web:

* *https://www.nba.com*
* *https://www.rotoworld.com*
* [*https://basketball.realgm.com*](https://basketball.realgm.com)

Tutti e tre i gruppi hanno in comune ciascuna 50 url rappresentanti gli stessi giocatori, cioè l’intersezione dei contenuti dei tre gruppi di pagine web è pari a questi 50 giocatori.

Ho effettuato le prove tramite le seguenti combinazioni:

* DataSet 1 – DataSet 2;
* Dataset 1 – DataSet 3;
* Dataset 2 – DataSet 3.

**2.3.1 Caso DataSet 1 – DataSet 2**

In questo caso le coppie rappresentanti gli stessi giocatori di basket erano 58. Ho aggiunto volutamente delle ripetizioni per valutare in modo accurato il comportamento del sistema in una casistica simile. Si riportano di seguito i risultati in termini di Precision, Recall ed F1:

* Precision: 0.90
* Recall: 1
* F1: 0.94

**2.3.2 Caso DataSet 1 – DataSet 3**

In questo caso le coppie rappresentanti gli stessi giocatori di basket erano 60. Si riportano di seguito i risultati in termini di Precision, Recall ed F1:

* Precision: 0.93
* Recall: 1
* F1: 0.96

La maggior parte degli errori effettuati è dovuta alla presenza di caratteri speciali all’interno di alcune url del terzo dataset.

**2.3.3 Caso DataSet 2 – DataSet 3**

In questo caso le coppie rappresentanti gli stessi giocatori di basket erano 60. Ho aggiunto volutamente delle ripetizioni per valutare in modo accurato il comportamento del sistema in una casistica simile. Si riportano di seguito i risultati in termini di Precision, Recall ed F1:

* Precision: 0.91
* Recall: 1
* F1: 0.95

Gli errori in questo caso sono stati provocati dalla presenza degli identificatori numerici dei giocatori all’interno delle url in entrambi i dataset e dai caratteri speciali presenti all’interno di alcune url del terzo dataset.

**3 Conclusioni e miglioramenti applicabili**

Le performance ottenute dal sistema sono buone in termini di efficacia ma non altrettanto in termini di efficienza. E’ necessario migliorare la procedura di download dei contenuti delle pagine web, di gran lunga lo step più oneroso in termini temporali e contenente alcune criticità nel caso di download massivo di un numero molto elevato di pagine.

Sono inoltre da valutare strategie più evolute nel filtraggio dei contenuti significativi dalle pagine web in quanto quello implementato risulta sicuramente sufficientemente efficace ma non particolarmente evoluto.

E’ poi da migliorare la procedura di calcolo di similarità tra le url in quanto si sono verificati errori di valutazione, minimi negli esempi sopra citati ma potenzialmente gravi nel caso di dataset molto più estesi, nei seguenti casi:

* Presenza di caratteri speciali;
* Presenza di codici identificativi, nativi del sito web e non associati al giocatore.