Audio fingerprinting in WebAssembly per l'esecuzione in browser web

Davide Pisanò, Antonio Servetti

Sommario

Negli ultimi anni si è assistito a una trasformazione nel paradigma di sviluppo delle applicazioni desktop, il quale si è mosso con determinazione verso un'architettura web-based. JavaScript è passato dall'essere un linguaggio per pagine web ad un linguaggio *general purpose*, rendendo necessaria l'introduzione di nuove tecnologie come WebAudio e WebAssembly. In questo documento si propone di indagare la possibilità di sfruttare queste tecnologie web per realizzare un sistema di audio fingerprinting che possa operare direttamente nel browser, valutandone l'effettiva fattibilità. Verrà inoltre trattata una declinazione innovativa del sistema di riconoscimento, attraverso la realizzazione di una *second screen application* eseguita nel browser. È inoltre stata condotta un'analisi dello stato dell'arte nel campo dell'audio fingerprinting, per comprendere le soluzioni già esistenti. I risultati ottenuti da questa ricerca promettono di fornire nuove prospettive per lo sviluppo di applicazioni audio web-based, aprendo le porte a possibili futuri sviluppi e applicazioni nell'ambito del riconoscimento audio in tempo reale all'interno del browser.

1 Introduzione

Negli ultimi anni si è notato un trend sempre crescente nell'utilizzo di JavaScript per la creazione di applicazioni desktop[6], grazie all'esistenza di tecnologie come Node.js[6] e Electron[7]. Il trend di scrivere applicazioni in JavaScript è stato amplificato dalla crescente importanza del web come piattaforma per la distribuzione di applicazioni software. Software utilizzati quotidianamente da miliardi di utenti sono basati sul web e, per forza di cose, devono essere scritti in JavaScript.

Da qui la nascita delle cosiddette *Rich Internet Applications*[1][2], ovvero applicazioni web che offrono un'esperienza utente interattiva e avanzata simile a quella di un'applicazione desktop tradizionale. Le RIA sono caratterizzate da una vasta gamma di funzionalità e interattività che le distinguono dalla semplici pagine web statiche. Oltre alla classica triade HTML + CSS + JavaScript, le RIA possono fare uso di tecnologie più avanzate e recenti come WebSockets, WebAudio, WebAssembly, WebRTC. In sostanza il browser diventa un'interfaccia o un'astrazione della macchina sottostante, alla quale si può accedere utilizzando JavaScript. Nello specifico WebAudio[1] è un'API JavaScript avanzata che consente di manipolare e generare audio all'interno del browser. È stata progettata per consentire agli sviluppatori di creare RIA che includono funzionalità audio, come la registrazione, la riproduzione e l'elaborazione di suoni. Inoltre, la necessità di scrivere applicazioni real time ha portato la necessità di dover eseguire codice ad alta efficienza, obiettivo non realizzabile facilmente con un linguaggio interpretato quale JavaScript. Per questo motivo nasce *WebAssembly*[2][3]: un formato di codice binario portabile che consente di eseguire codice di basso livello all'interno del browser web.

L'obiettivo di questo documento è quello di discutere la realizzazione di un sistema per l'identificazione di audio, da eseguire su dispositivi eterogenei all'interno di un web browser, sfruttando Wasm e WebAudio. Questo presenta numerosi vantaggi, tra i più importanti:

- l'evitare all'utente il download di un'app addizionale
- la riduzione del carico lato server: parte della complessità può essere spostata sul dispositivo dell'utente finale

Si guiderà il lettore attraverso la realizzazione del sistema di identificazione tra le varie sezioni; trattando la realizzazione di una *second-screen application* per la presentazione, in sincronia con un brano riprodotto, del testo del brano stesso. Verranno analizzate anche le differenze e il funzionamento di un altro algoritmo concorrente open-source, per poi passare, infine, ai possibili utilizzi futuri.

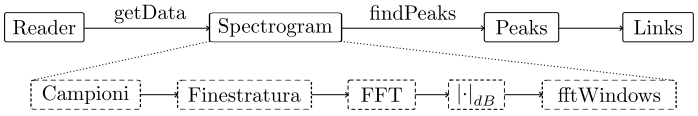

Figura 1: Schema estrazione Links

2 L'estrazione delle features

Il primo passo è quello di estrarre dai segnali audio delle features, sulla base delle quali effettuare il fingerprinting e la ricerca. Il processo è indicato in figura 1. Si parte da una rappresentazione del dominio nel tempo del segnale audio (**Reader**), si procede creando lo spettrogramma (**Spectrogram**), successivamente si estraggono i picchi (**Peaks**) più intensi[3] (**Peaks**) e infine si creano i **Links**, ovvero il *fingerprint* del segnale audio.

Lo spettrogramma è ottenuto finestrando il segnale audio con overlap del 50%. In figura 2 è possibile vedere il

[1] D'ora in avanti RIA

[2] Più semplicemente Wasm

[3] Nonché i più significativi

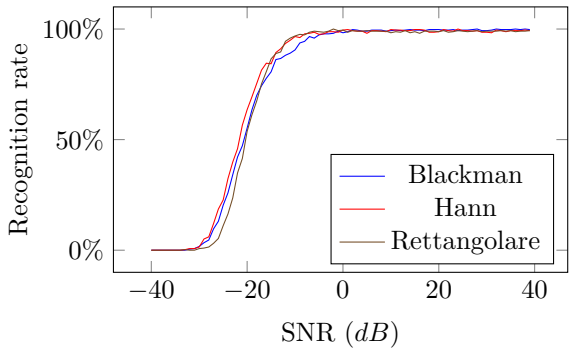

Figura 2: Recognition rate in funzione dell'SNR per le varie finestre

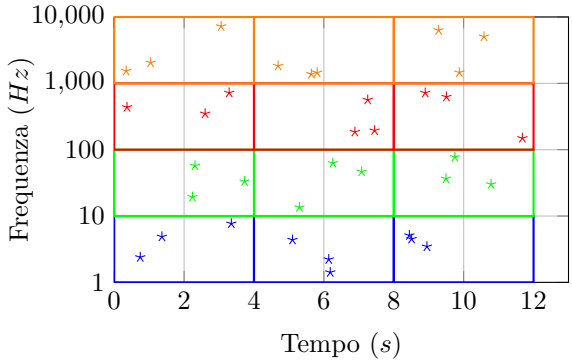

Figura 3: Spettrogramma con 3 picchi per cella

recognition rate al variare della funzione finestra utilizzata e dell'SNR del segmento da riconoscere: si è scelto di utilizzare la finestra di *Hann*[5].

L'individuazione dei picchi avviene suddividendo lo spettrogramma in celle composte da 32 fineste e range di frequenze in base alla *scala Mel*[4] (figura 3).

Infine ogni picco viene considerato un *anchor point*, al quale è associata una determinata *target zone* (ovvero una porzione della costellazione dei picchi), per ogni picco della target zone viene creato un *Link* (figura 4), ovvero una

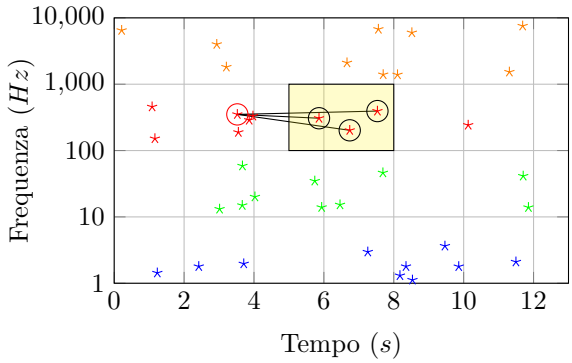

Figure 4: Selezione dell'anchor point e della target zone

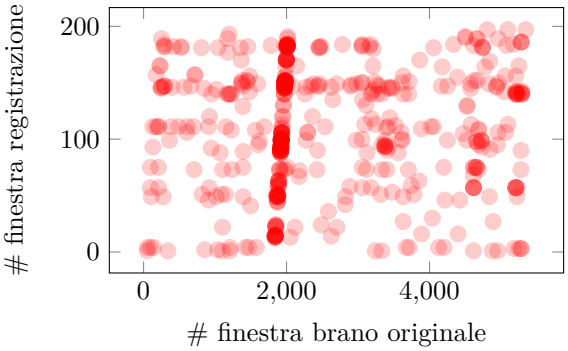

Figura 5: Scatterplot di un match

struttura dati definita come:

$$Link \begin{cases} hash = h(\delta_w, \delta_f, \alpha.frequency) \\ window = \alpha.window \end{cases}$$

Dove α è l'anchor point e β un picco nella target zone, allora $\delta_w = \beta.window - \alpha.window$, $\delta_f = \beta.frequency - \alpha.frequency$ e $h(\odot)$ è una funzione di hash.

3 Matching

Estratti e memorizzati i Link di una libreria di brani allora si può procedere al *matching*. Si ammetta di voler riconoscere un segmento audio proveniente da un brano non noto, allora si estrarranno i Link anche in questo caso e si metteranno i due set di Link in relazione tenendo in conto che tra i Links del brano originale $Links_O$ e quelli del segmento registrato $Links_R$ dev'esserci una differenza temporale ΔT costante e non negativa se si riferiscono allo stesso brano e che gli hash di $Links_O$ e di $Links_R$ devono corrispondere. A questo punto basta quindi raggruppare per ΔT e per id del brano originale, definendo con n il numero di elementi che ricadono nello stesso gruppo, ordinare per n decrescente ed, infine, estrarre il primo gruppo che rappresenta il match migliore.

In figura 5 sull'asse delle ascisse è rappresentato il numero di finestra nella quale è contenuto un Link di $Links_O$, analogamente per l'asse delle ordinate ma per i $Links_R$: si noti la diagonale a densità maggiore intorno al valore 2000 sulle x, ad indicare un match.

4 Testo del brano in sincronia

Sfruttando l'algoritmo di fingerprinting e matching descritto precedentemente è possibile realizzare una *second screen application* che identifichi un brano in riproduzione attraverso il microfono e che ne mostri il relativo testo sincronizzato in tempo reale, il tutto all'interno di un web browser.

L'applicazione, facendo ricorso a WebAudio e a Wasm, registrerà un breve segmento audio attraverso il microfono dell'utente, ne estrarrà i Links e li invierà a un server dove sono contenuti i Links dei brani originali, il server cercherà di individuare un match e risponderà al client con il nome

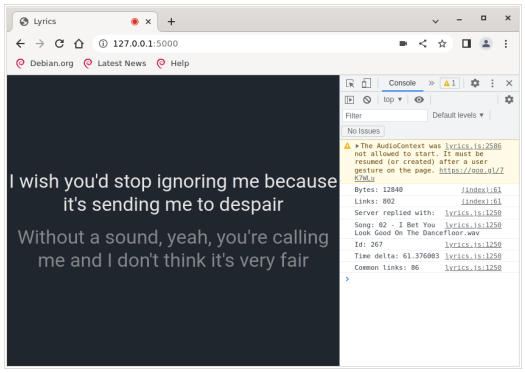

Figura 6: Screenshot testo sincronizzato

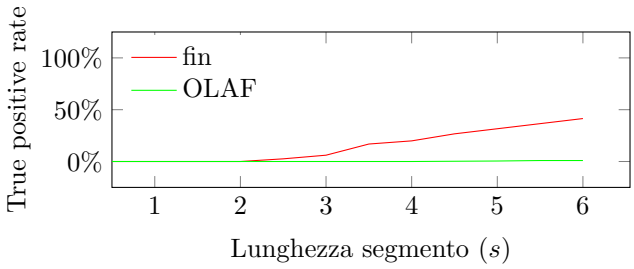

Figura 7: Rate positivi con $SNR = -20dB$

del match e punto del brano originale in cui sono contenuti i Link del segmento, il client riceve queste informazioni, ottiene il testo del brano identificato e lo sincronizza in base al tempo trascorso. In figura 6 è presentato il sistema in azione.

5 Altre soluzioni

OLAF[8] è un'applicazione per l'identificazione audio, simile, nello scopo, all'algoritmo già presentato in questo documento.

Sono stati effettuati dei confronti tra i due sistemi, testandone il funzionamento su segmenti audio molto rumorosi, alcuni risultati sono riportati nelle figure 7 e 8, dove con *fin* ci si riferisce all'implementazione oggetto di questo documento. Si nota ad occhio la superiore robustezza al rumore rispetto ad OLAF: questo è dovuto al fatto che l'implementazione qui descritta presenta un algoritmo per la scelta dei picchi più rifinito, in cui viene sfruttato il concetto di banda critica, permettendogli di essere più immune al rumore.

6 Conclusioni

Il sistema proposto in questa tesi è nato come un progetto personale, foraggiato dalla voglia di avere una migliore comprensione delle tecnologie impiegate nel riconoscimento audio, interrogandosi sul come fosse possibile realizzare un sistema di tale portata.

La second screen application ha rappresentato un'evoluzione naturale del sistema di fingerprinting e matching, ma

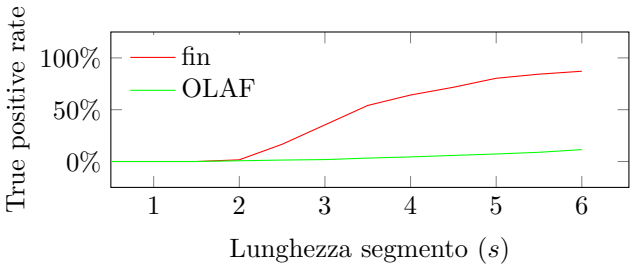

Figura 8: Rate positivi con $SNR = -10dB$

la sfida principale è stata l'esecuzione dell'algoritmo nell'ambiente browser web.

È importante sottolineare che il percorso di ricerca non si conclude con questa tesi: l'algoritmo presentato può essere usato anche in ambiti diversi rispetto a quello del riconoscimento musicale, quali la sincronizzazione di stream multimediali e il riconoscimento di particolari segnali audio che indicano l'accadere di un determinato evento. Non si esclude la possibilità di scoprire nuove e inaspettate applicazioni non trattate in questo documento.

L'informatica è un settore in continua evoluzione: nulla è da considerarsi *irrealizzabile*.

Riferimenti bibliografici

[1] Hongchan Choi. Audioworklet: the future of web audio. In *ICMC*, 2018.

[2] Piero Fraternali, Gustavo Rossi, and Fernando Sánchez-Figueroa. Rich internet applications. *IEEE Internet Computing*, 14(3):9–12, 2010.

[3] Andreas Haas, Andreas Rossberg, Derek L Schuff, Ben L Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and JF Bastien. Bringing the web up to speed with webassembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 185–200, 2017.

[4] Paul Pedersen. The mel scale. *Journal of Music Theory*, 9(2):295–308, 1965.

[5] Prajoy Podder, Tanvir Zaman Khan, Mamdudul Haque Khan, and M Muktadir Rahman. Comparative performance analysis of hamming, hanning and blackman window. *International Journal of Computer Applications*, 96(18):1–7, 2014.

[6] Guillermo Rauch. *Smashing node. js: Javascript everywhere*. John Wiley & Sons, 2012.

[7] Adam D Scott. *JavaScript everywhere: building cross-platform applications with GraphQL, React, React Native, and Electron*. O'Reilly Media, 2020.

[8] Joren Six. Olaf: a lightweight, portable audio search system. *Journal of Open Source Software*, 8(87):5459, 2023.