

1. INTRODUZIONE

Twitch è un popolare sito di live streaming lanciato nel 2011 da Justin Kan e Emmett Shear ed acquistato nel 2014 da Amazon per 970 milioni di dollari. Il sito consente la trasmissione in tempo reale di diversi contenuti quali: videogiochi, eventi e competizioni eSports, ma anche contenuti musicali, creativi e "in real life" (IRL). Il numero di persone presenti ogni giorno su Twitch è in continuo aumento. Solo in Italia sono circa 3 milioni gli utenti che fruiscono regolarmente dei contenuti, la crescita è avvenuta soprattutto nel 2020 e 2021 durante il periodo del lockdown.

Le opportunità offerte da questa piattaforma sono molte sia per coloro che trasmettono contenuti, sia per le aziende legate a questo settore. In particolare, i primi hanno la possibilità di guadagnare con le proprie dirette attraverso pubblicità e donazioni dirette degli utenti mentre le aziende del settore (quali case editrici di videogiochi, aziende di apparecchiature tecnologiche e altre) hanno a disposizione una vetrina per mostrare i loro prodotti. Vengono infatti spesso stretti accordi di sponsorizzazione tra streamer e aziende o vengono creati eventi esclusivi per l'uscita di nuovi videogiochi.

Le possibilità sono tante ma per avere successo è necessario avere una visione della struttura della piattaforma per capire gli interessi. In particolare è fondamentale conoscere gli streamer, gli utenti e le community attorno alla figura dello streamer che compongono la piattaforma per riuscire a capire quali contenuti proporre e quali collaborazioni stringere.

Questo progetto si focalizza quindi sul raccogliere e analizzare i dati delle dirette di streamer italiani per realizzare un grafo delle community presenti.

2. OBIETTIVI

Gli obiettivi alla base dello sviluppo di questo progetto sono molteplici e ambiziosi. Abbiamo immaginato un modello fruibile in differenti scenari, la cui struttura permettesse di soddisfare diversi intenti di ricerca, come ad esempio i seguenti:

- visualizzazione e analisi dei cluster all'interno della piattaforma virtuale di Twitch, per poter comprendere la qualità e l'entità della fandom che si sviluppa attorno a specifici topic e/o streamer;
- analisi della qualità di interazione tra lo streamer e gli utenti spettatori attraverso la chat;
- visualizzazione e analisi della concorrenza tra community: ricerca di eventuali correlazioni tra differenti cluster;
- analisi della reattività di una community a seguito della pubblicazione di un nuovo videogioco.

Lo scopo del nostro progetto è quindi creare un modello utile per streamer neofiti, affinché possano analizzare lo scenario globale e comprendere il contesto più vantaggioso in cui inserirsi, e per streamer affermati, per permettere loro di espandere la community, migliorare le loro performance in termini di follower, ore trasmesse, spettatori e aumentare

le opportunità di guadagno. Un ulteriore proposito è che il modello possa essere utilizzato da inserzionisti pubblicitari, come supporto nella pianificazione di attività promozionali e quindi nella ricerca degli streamer più influenti e in linea con il prodotto.

3. STRUTTURA DEL PROGETTO

Il flusso di raccolta, analisi e salvataggio di dati è mostrato in figura 1. La fonte principale è la piattaforma Twitch, da cui sono stati estratti dati sugli stream, gli utenti e sui bot presenti. Lo storage di questi dati è stato successivamente implementato all'interno di MongoDB [4], DBMS non relazionale orientato ai documenti. La fonte secondaria è la banca dati online di Steam: i dati estratti ed elaborati su Python sono stati utilizzati per l'enrichment del database principale. Per la natura relazionale dei dati coinvolti, abbiamo scelto infine di modellare e memorizzare i dati arricchiti sul database a grafo Neo4j, rappresentando entità e loro relazioni con nodi e archi. Questa modellizzazione permette di eseguire rapidamente query per investigare la struttura delle reti quali ad esempio i social network. Il grafo ottenuto è stato infine esportato anche sul Gephi [5], software open source per l'analisi e la visualizzazione delle reti sociali.

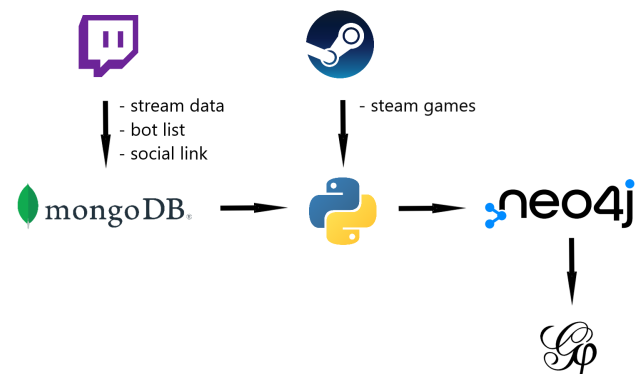


Fig. 1: Flusso di raccolta, analisi e salvataggio di dati.

4. DATA ACQUISITION

I dati raccolti per la creazione del dataset finale sono stati acquisiti tramite le modalità di scraping e API. I dati raccolti si riferiscono tutti al mese di maggio 2022. Di seguito si riportano le fonti, le metodologie applicate e gli attributi creati.

1. API REST

Questa procedura è stata utilizzata per la fonte principale dei nostri dati che è la piattaforma Twitch.

a. Stream files

Per reperire i dati relativi alle live istante per istante sono state utilizzate le API ufficiali di Twitch.tv [6]. Dopo aver creato un account developer e aver ottenuto le chiavi di accesso è stato creato uno script in Python per fare le richieste. In particolare vengono inizialmente fatte le richieste per ottenere la lista di tutti gli streamer classificati come italiani che sono live in quel momento e alcune informazioni riguardo alla live. In seguito, avendo la lista degli streamer,

vengono poi inoltrate le singole richieste per ottenere le liste dei nickname di tutti gli utenti connessi alle diverse chat.

Per ottenere un quadro dell'andamento degli spettatori è necessario eseguire lo script ad intervalli regolari attraverso un task scheduler. Per questa raccolta è stata utilizzata una macchina virtuale con il sistema operativo linux, ed è quindi stato utilizzato il comando crontab. Inoltre è stato impostato un intervallo di 5 minuti tra una esecuzione e la successiva in un range temporale dal 05/05/2022 15:09:10 al 19/05/2022 14:59:21, per un totale di 14 giorni di acquisizione.

Per gestire in modo efficiente le richieste e il flusso di dati in ingresso, sono stati adottati nello script i seguenti accorgimenti:

- utilizzo della libreria `asyncio` [7], per permettere richieste multiple in modo asincrono alle API di Twitch;
- procedure di logging per tenere traccia dei timestamp relativi all'inizio di una richiesta, al salvataggio di un file o a eventuali errori;
- blocchi `try/except` per gestire i possibili errori, tra i più frequenti ci sono quelli relativi al Timeout della richiesta ed errori relativi alla decodifica JSON di caratteri speciali.

Ciò che si ottiene dalla raccolta è un file JSON per ogni istante temporale che riporta le informazioni riguardanti i diversi streamer online e i loro spettatori. Inoltre viene aggiornato ad ogni iterazione un file di log per tenere traccia dei progressi. Di seguito è riportato un dettaglio dei risultati.

Attributi del file JSON:

- *username*: nickname dello streamer, questo campo è utilizzato come chiave ;
- *spect*: lista degli username degli spettatori alla live;
- *game_name*: nome del topic/gioco della live al momento; dell'acquisizione
- *viewer_count*: conteggio dei viewer alla live riportato da Twitch.

Lo storage di questi elementi è stato fatto inizialmente su una serie di file in formato JSON, uno per ogni sessione di raccolta. In seguito, per una gestione più efficiente dei risultati, è stato implementato il salvataggio sul DBMS MongoDB. Attraverso la libreria ufficiale `pymongo` [8] di Python è stata creata una collezione salvata in locale; questa contiene i diversi elementi in formato documentale. Di seguito è riportato un estratto di MongoDB che indica le informazioni complessive sulla collezione e un dettaglio su un singolo documento della collezione.

Twitch_stream_collection		
Storage size:	Documents:	Avg. document size:
2.24 GB	3.8 K	1.04 MB

Fig. 2: Informazioni sulla collezione di MongoDB.

Questo approccio permette di verificare periodicamente l'acquisizione dei dati e permette inoltre di fare query ed exploration sui dati grezzi.

```
_id: ObjectId('6297d6b0d6f6edd4dc99cb5a')
> ilMasseo: Object
> pizfn: Object
> Xiuder_: Object
> ocwspost: Object
> Juventibus: Object
> Justees: Object
> Paoloidolo: Object
  > spect: Array
    game_name: "League of Legends"
    viewer_count: 1332
    date_time: "2022-05-05 15:09:00"
> NanniTwitch: Object
> fragolaqt: Object
> MarcoMerrino: Object
> S7ORMyTv: Object
> Multiplayerit: Object
> AmalaTV: Object
> Everyeyait: Object
> Efesto96: Object
> Yoshi93: Object
> Fantacalcio: Object
```

Fig. 3: Esempio di documento presente nella collezione.

b. Link social network

Si sono utilizzate le API anche per ottenere i link dei social network dei diversi streamer presi in esame. Questi sono presenti nella descrizione del canale degli streamer ma non è possibile recuperarli direttamente con le API ufficiali di Twitch, come mostrato nella figura 4. In questo caso per fare le richieste è stato necessario inoltrare le richieste a un endpoint non ufficiale di Twitch [9]. Trattandosi di diverse richieste anche in questo caso sono stati tenuti i seguenti accorgimenti:

- utilizzo della libreria `asyncio` [7], per permettere API request multiple in modo asincrono;
- blocchi `try/except` per gestire i possibili errori.

Ciò che si ottiene dalla raccolta è un file CSV che associa ad ogni streamer la lista dei social presenti nella sua descrizione del canale.

- *streamer*: username dello streamer;
- *social_list*: lista dei link social presenti sul profilo dello streamer.

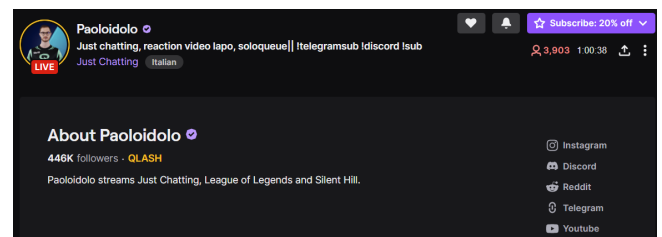


Fig. 4: Esempio di descrizione del canale. In alto a sinistra è presente il nome dello streamer, in alto a destra è visibile il numero dei viewer mentre in basso a destra sono presenti i link ai social network dello streamer.

2. Web scraping

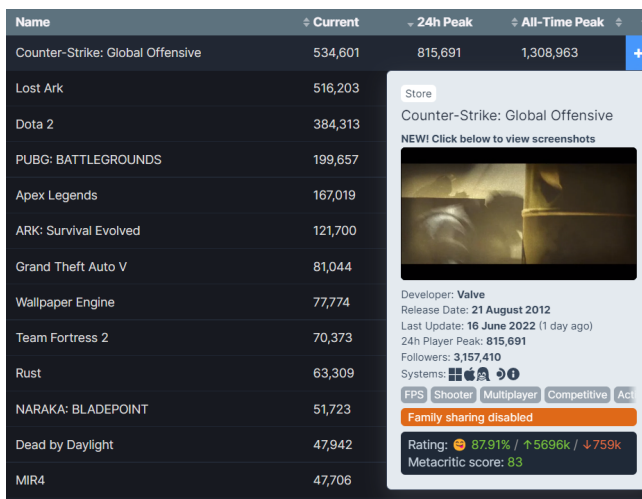
Per la realizzazione del modello finale si è reso necessario raccogliere anche dati provenienti da altre fonti. In particolare sono stati ottenuti altri due dataset utilizzando diverse tecniche di scraping. Qui di seguito riportiamo il dettaglio.

a. Lista giochi SteamDB

Il primo dataset ottenuto con questa tecnica è quello relativo alle informazioni sulle categorie trasmesse, in particolare per quanto riguarda i videogiochi. Questo è stato ottenuto dalla piattaforma Steam [10], si tratta di uno store digitale che si occupa di gestire e distribuire una vasta gamma di videogiochi, alcuni in modo esclusivo. Il database di questo sito è ospitato dal sito SteamDB [2] e risulta quindi molto fornito per quanto riguarda il mondo dei videogiochi. Contiene diverse informazioni riguardo alle caratteristiche, allo sviluppo e al rilascio di diversi titoli.

Per ottenere le informazioni sui videogiochi abbiamo effettuato in particolare lo scraping della pagina relativa ai giochi più giocati di sempre [2]. Questa pagina fornisce i titoli che hanno avuto almeno una volta dalla data del lancio un picco di almeno 1000 utenti attivi. Al momento della raccolta la lista era composta da 5621 giochi.

Lo scraping è stato effettuato con Python utilizzando la libreria Selenium [11]. Questa è una libreria di browser automation che consente di automatizzare il browser ad eseguire le operazioni più comuni durante la navigazione web. In questo modo è possibile interagire con gli elementi del sito per simulare il comportamento umano. Nel caso di SteamDB infatti le informazioni relative ai videogiochi sono celate in sotto finestre a comparsa che si attivano al passaggio del mouse. Qui di seguito è riportato un esempio.



Name	Current	24h Peak	All-Time Peak
Counter-Strike: Global Offensive	534,601	815,691	1,308,963
Lost Ark	516,203		
Dota 2	384,313		
PUBG: BATTLEGROUNDS	199,657		
Apex Legends	167,019		
ARK: Survival Evolved	121,700		
Grand Theft Auto V	81,044		
Wallpaper Engine	77,774		
Team Fortress 2	70,373		
Rust	63,309		
NARAKA: BLADEPOINT	51,723		
Dead by Daylight	47,942		
MIR4	47,706		

Store	Counter-Strike: Global Offensive
NEW! Click below to view screenshots	
Developer: Valve	
Release Date: 21 August 2012	
Last Update: 16 June 2022 (1 day ago)	
24h Player Peak: 815,691	
Followers: 3,157,410	
Systems: FPS Shooter Multiplayer Competitive Act	
Family sharing disabled	
Rating: 87.91% / +5696k / +759k	
Metacritic score: 83	

Fig. 5: Finestra esempio di SteamDB per il videogioco Counter-Strike: Global Offensive.

Tramite il framework composto da Selenium e da Web-Driver, in questo caso di Chrome, è stato possibile effettuare lo scraping del sito web in modo dinamico inoltrando una richiesta per ogni videogioco. Per aggirare i controlli di verifica CAPTCHA per l'individuazione dei bot è stato necessario pulire periodicamente i cookie raccolti della pagina web.

Ciò che si ottiene dalle richieste sono i file HTML delle

single finestre e tramite l'uso della libreria BeautifulSoup [12], che permette di maneggiare file HTML e XML, è stato possibile individuare le sezioni presenti in specifiche classi. Per ogni gioco sono quindi stati estratti i seguenti attributi:

- **Title:** titolo del gioco;
- **Developer:** nome dello sviluppatore;
- **Publisher:** casa editrice del gioco;
- **Release Date:** data di rilascio;
- **24h Player Peak:** picco di giocatori attivi nelle ultime 24h;
- **Followers:** numero di seguaci sulla piattaforma;
- **Categories:** categorie e tag riferite al gioco (esempio: Indie, FPS, Multiplayer).

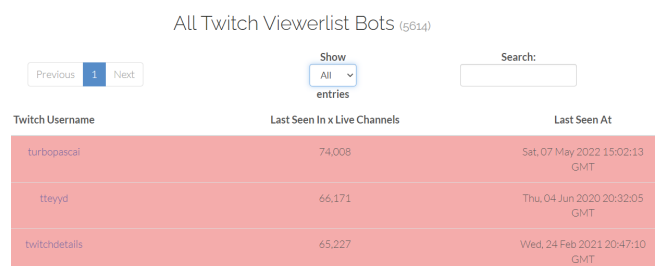
La raccolta dei dati ha permesso di raccogliere le informazioni di 3498 videogiochi dei 5621 presenti. Il motivo di questa raccolta incompleta è dovuta ad errori HTTP 406 Not Acceptable nel caricamento di alcune finestre. Questo errore indica incompatibilità tra pagina web e browser che può essere causata dal fatto che non è stato possibile creare una rappresentazione del documento sul server.

Terminata la raccolta i dati così ottenuti sono stati salvati in un file JSON chiamato *steam_games*.

b. Lista bot

Sulla piattaforma Twitch c'è la possibilità di realizzare dei bot che possono interagire con la chat al fine di aggiungere funzionalità. Spesso vengono utilizzati per rimuovere i messaggi ed eventualmente bannare gli utenti che non rispettano le linee guida della piattaforma. Questi bot però, come gli utenti reali, compaiono nella lista degli spettatori e devono essere rimossi al fine di evitare errori nella fase di analisi.

Sul sito Twitch Insights [13], di proprietà di Twitch, è possibile scaricare la lista completa dei nickname e le informazioni relative ai bot account comparsi nelle viewerlist delle live stream da agosto 2018. Questa lista è la stessa che usa la piattaforma per correggere i dati sul numero di viewer, nella figura 6 è riportato un estratto della tabella presente sul sito.



All Twitch Viewerlist Bots (5614)		
Twitch Username	Last Seen In x Live Channels	Last Seen At
turbopascal	74,008	Sat, 07 May 2022 15:02:13 GMT
ttheyd	66,171	Thu, 04 Jun 2020 20:32:05 GMT
twitchdetails	65,227	Wed, 24 Feb 2021 20:47:10 GMT

Fig. 6: Lista dei bot di Twitch.

Per scaricare questa abbiamo utilizzato Table Capture [14], una estensione del browser che permette di ottenere file CSV scaricabili a partire da tabelle presenti sul web in formato HTML. Al momento dell'acquisizione sono stati scaricati 5576 nickname appartenenti a bot. I dati così ottenuti sono stati salvati nel file *Twitch_bot_list.csv*.

5. DATA EXPLORATION

1. Exploration

Terminata la fase di acquisizione si procede all'analisi esplorativa dei dati per scoprire i primi insight e possibili problematiche da affrontare. Dai dati relativi alle live degli streamer si possono estrarre le seguenti statistiche preliminari.

	minuti trascorsi in live	spettatori totali
mean	1466.684582	1656.022506
std	1735.805536	7450.436019
min	15.000000	10.000000
25%	345.000000	82.000000
50%	960.000000	231.000000
75%	2010.000000	801.000000
max	19125.000000	157718.000000

Fig. 7: Statistiche sul tempo trascorso in live e sul numero di spettatori dello streamer.

In due settimane in media uno streamer ha trascorso in live circa 24 ore e 27 minuti, con una media di 1656 spettatori.

Una prima problematica che emerge dall'analisi delle liste degli spettatori in live è la discrepanza tra la lunghezza delle suddette e il campo relativo al conteggio dei viewer. I motivi sono due: la presenza dei bot pesa solo sul numero di spettatori e non sul numero dei viewer, inoltre i nomi degli spettatori raccolti comprendono solo gli utenti connessi alla chat della live *this list only includes logged in Twitch users that have connected to your chat – including lurkers, anyone that has chat popped out of the video page, or in chat-only mode on mobile devices* [15].

Juventibus	Viewer Count: 2824	Spect Count: 1914	Difference: 910	Bot Number: 11
ocusport	Viewer Count: 2575	Spect Count: 1966	Difference: 609	Bot Number: 11
pizfn	Viewer Count: 2115	Spect Count: 1576	Difference: 539	Bot Number: 10
Justees	Viewer Count: 1768	Spect Count: 1280	Difference: 488	Bot Number: 10
Xiuder_	Viewer Count: 1752	Spect Count: 1185	Difference: 567	Bot Number: 17
Sabaku_no_Sutoriimaa	Viewer Count: 1664	Spect Count: 1172	Difference: 492	Bot Number: 6

Fig. 8: Differenza tra numero di viewer e numero di spettatori.

Per concludere l'analisi esplorativa dello *streamer_dataset* si mostra come la distribuzione degli utenti online sulla piattaforma nell'arco di una giornata rivela un picco posizionato alle 22 di sera mentre il minimo corrisponde alle 6 del mattino (figura 9 e figura 10).

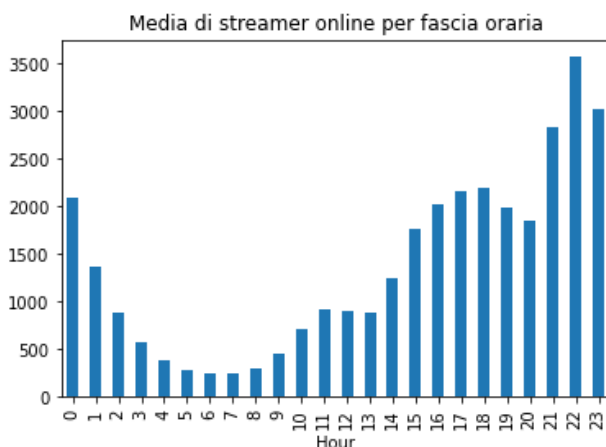


Fig. 9: Numero medio di streamer attivi su Twitch ogni ora.

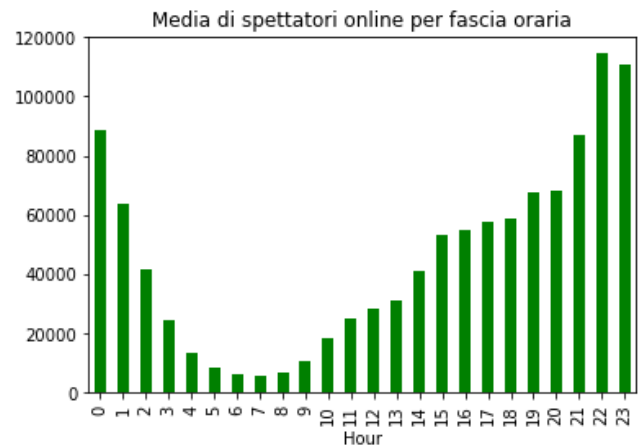


Fig. 10: Numero medio di spettatori che visualizzano una live su Twitch ogni ora.

Lo scatterplot in figura 11 indaga la relazione tra il numero di spettatori e i minuti in live per gli streamer con il più alto numero di viewer medi.

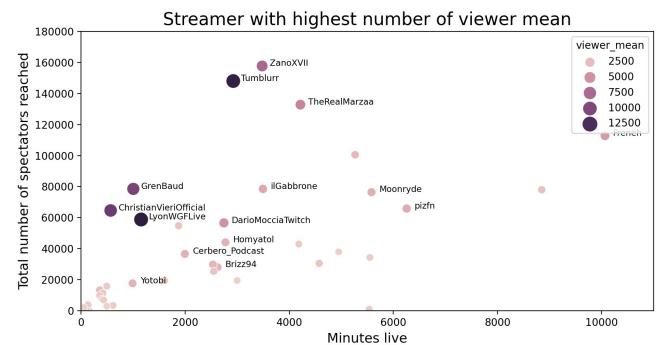


Fig. 11: Scatterplot degli streamer con il maggior numero di viewer (number of spectators vs minutes in live).

L'analisi esplorativa del dataset sui videogiochi mostra come il gioco più vecchio su SteamDB è "The Curse of Monkey Island" del 1997, mentre quello più recente è "Card Shark" del 02/06/2022. Si mostra in figura 12 il numero di giochi lanciati negli ultimi 5 anni.

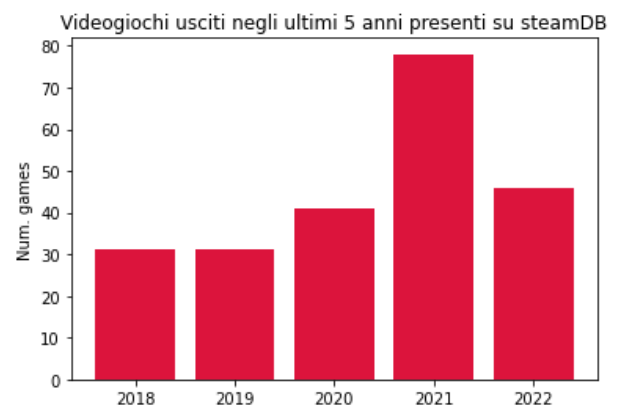


Fig. 12: Barchart dei videogiochi lanciati ogni anno.

Nella figura 13 invece sono riportati i nomi delle maggiori compagnie sviluppatrici di videogames presenti nel dataset.

Particolarmente rilevante è anche l'esplorazione dei titoli lanciati sul mercato proprio durante il periodo di osser-

Top 10 Game Development Companies

CAPCOM Co., Ltd.
Square Enix
Valve
Ubisoft Montreal
Arc System Works
Capcom
Infinity Ward
Rockstar Games
Konami Digital Entertainment
BANDAI NAMCO Studios Inc.

Fig. 13: Nomi delle principali case produttrici e sviluppatrici di videogiochi.

vazione della community di Twitch. Si registra che tra il 05/07/2022 e il 19/05/2022 sono stati rilasciati 11 videogiochi (mostrati in figura 14).

gameName	categories
Source of Madness	Rogue-lite; Lovecraftian; Action; Action RPG; ...
Eiyuden Chronicle: Rising	Adventure; RPG; Action RPG; Action-Adventure; ...
Brigandine: The Legend of Runersia	Strategy; Turn-Based Strategy; Turn-Based Tact...
V Rising	Survival; Open World; Vampire; Crafting; PvP; PvE
Warhammer 40,000: Chaos Gate - Daemonhunters	Warhammer 40K; Turn-Based Strategy; Strategy R...
Arma Reforger	Action; Simulation; Strategy; Shooter; Sandbox...
Ultimate Epic Battle Simulator 2	Simulation; Action; Strategy; God Game; Sandbo...
Citizen Sleeper	RPG; Cyberpunk; Choices Matter; Tabletop; Expl...
We Were Here Forever	Exploration; Puzzle; Online Co-Op; Escape Room...
Little Witch in the Woods	Cute; Pixel Graphics; Funny; RPG; Magic; Relaxing
Songs of Conquest	Adventure; Strategy; Turn-Based Strategy; Turn...

Fig. 14: Nomi e categorie dei videogiochi rilasciati tra il 05/05/2022 e il 19/05/2022.

6. DATA PROCESSING AND ENRICHMENT

Questa sezione è il cuore del progetto che permette di passare dai dati grezzi provenienti dalle due fonti ai quattro dataset necessari per comporre il dataset finale. In particolare vengono inizialmente processati i dataset grezzi relativi alle live e agli streamer (*stream files*, *bot_list*, *social_link*) ed in un secondo momento viene utilizzato il dataset *games_dataset* proveniente da SteamDB per integrare le informazioni.

1. Data Processing

Il processing dei dati è composto da diverse parti ed è stato eseguito interamente su Python sfruttando principalmente le librerie Pandas [16] e Numpy [17] per la manipolazione dei dati.

a. Importazione dei dati

Il primo step è dunque l'importazione dei dati. Per quanto riguarda i files relativi alle stream è stata resa possibile l'importazione sia dai file multipli che dalla collezione presente in MongoDB utilizzando nuovamente la libreria pymongo per accedere al server locale. L'importazione in entrambi i casi è stata fatta campionando i dati ed importando solo un file ogni 3, questo corrisponde a considerare intervalli temporali di 15 minuti che si sono rivelati essere un buon compromesso tra quantità di informazioni e maneggevolezza

dei dataset.

b. Pulizia dei dati

In secondo luogo è stata eseguita una attenta pulizia dei dati, in particolare sono stati rimossi tutti i nickname corrispondenti a bot, come già accennato nella sezione di exploration. Sono stati quindi incrociati tutti gli spettatori per ogni istante temporale con la lista dei bot presenti nel dataset *bot_list* e si è proceduto a rimuoverli. Analizzando il risultato di questa operazione è possibile notare come le discrepanze segnalate nella sezione esplorativa tra il conteggio dei viewer e degli spettatori è andata a ridursi ma non ad annullarsi completamente. Questo è dovuto al fatto che ci sono altre ragioni che concorrono a questa differenza: la possibilità che alcuni utenti non siano effettivamente connessi alla chat e la possibilità che siano dati con tasso di aggiornamento leggermente diverso (di fatto non è noto quale siano le frequenze di aggiornamento dei due dati).

c. Analisi dei dati

Terminata questa fase di pulizia si opera una espansione del dataset relativo alle stream. È stata creata una tabella con presente in ogni riga uno streamer con i dettagli della live per ogni istante. Di seguito sono riportati gli attributi della tabella:

- *streamer*: nome dello streamer;
- *game_name*: categoria/gioco che sta trasmettendo;
- *viewer_count*: numero dei viewer fornito da twitch;
- *spect_count*: numero degli spettatori estratto dalle liste;
- *spect*: lista completa e univoca degli spettatori che erano presenti in chat nel periodo di raccolta dei dati.

Questo dataset, in seguito ad una aggregazione temporale delle live, permette di passare al dataset con informazioni relative al singolo streamer e al dataset contenente le categorie trasmette che verranno poi integrate con le informazioni provenienti da SteamDB.

Per quanto riguarda il dataset relativo agli streamer sono stati calcolati i campi relativi alle statistiche sui numeri degli spettatori e viewer (valori medi e picchi massimi) ed è stato calcolato il tempo trascorso in live da ogni streamer sulla base del numero di intervalli temporali in cui era presente. In seguito, al fine di sfoltire il dataset dei valori non significativi per la creazione del grafo finale, si stabiliscono delle soglie per eliminare gli streamer che non hanno mostrato un livello di attività sufficiente. Nello specifico le soglie sono:

- *min_stream_time* = 4 tempo minimo in cui uno streamer è stato in live (corrispondente a 60 minuti);
- *min_spect_mean* = 10 numero minimo di spettatori (in media);
- *min_viewer_mean* = 10 numero minimo di viewer (in media).

Con queste restrizioni il dataset contenente i dati di 45583 streamer si riduce a 2977. In vista poi della fase in cui verranno create le relazioni al dataset viene aggiunto l'attributo

streamerId. Questo ha il compito di fare da indice e permettere la creazione di tabelle ponte. Il dataset così ottenuto viene esportato in un file CSV.

Inoltre anche per quanto riguarda il dataset relativo alle categorie trasmesse viene aggiunto l'attributo *gameId* per facilitare le referenze. Così composto il dataset è pronto per essere integrato (dettagli nella sezione data enrichment).

Creati i due dataset che comporranno i nodi è necessario creare le tabelle ponte che costituiranno le relazioni tra le entità. In particolare le tabelle che permettono la relazione streamer-game e streamer-streamer.

La prima ad essere creata è la tabella che permette di associare lo streamer alle categorie che ha trasmesso nelle due settimane di raccolta dati. Dall'aggregazione al punto precedente vengono associati gli id delle due tabelle in una tabella ponte con gli attributi:

- *ID_streamer*: chiave del dataset streamer;
- *ID_game*: chiave del dataset games;
- *minutes*: campo calcolato che indica il numero di minuti per il quale lo streamer ha trasmesso una certa categoria.

La tabella così creata contiene 7709 relazioni streamer - categoria. La tabella ponte viene quindi esportata in un file CSV.

La seconda tabella creata è quella relativa alle relazioni streamer-streamer. La relazione tra due streamer viene calcolata sulla base del numero di spettatori in comune che presenta la coppia. L'overlap deve quindi essere valutato per ogni coppia di streamer come percentuale di spettatori in comune rispetto all'altro. In questa fase deve essere preso l'accorgimento di mettere a numeratore nella divisione per il calcolo della percentuale lo streamer con il numero inferiore di spettatori totali e a denominatore quello con il numero maggiore. Questo accorgimento è necessario per permettere anche a streamer con numeri minori di avere un'alta percentuale di associazione con gli streamer maggiori se presentano molti spettatori in comune.

Anche in questo caso sono stati imposte delle soglie minime per poter considerare significative le relazioni. È stata impostata quindi la soglia *min_overlap_percentage = 10* per indicare la percentuale minima per ritenere valido l'overlap. Questo passaggio è particolarmente importante perché in un'ottica di scalabilità il numero di connessioni cresce esponenzialmente con la soglia minima considerata. In figura 15 si mostra come varia il numero di relazioni tra streamer al variare della soglia minima di overlap.

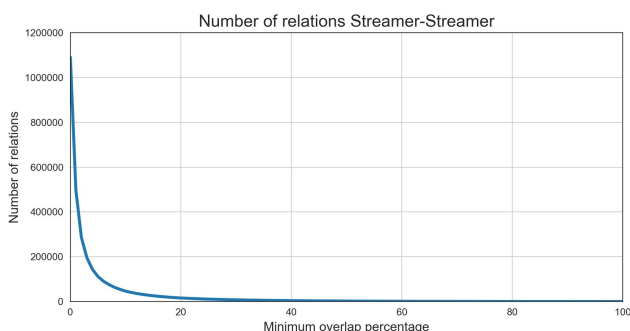


Fig. 15: Numero di relazioni intercettate tra streamer in base alla percentuale minima di overlap.

Il 10%, che corrisponde a 47222 relazioni, risulta essere una percentuale adeguata per intercettare la maggior parte dei collegamenti e rimuovere quello che può essere definito rumore di fondo. Questo può essere dato da una apertura errata di una live da parte di un utente o dall'arrivo su una live tramite host. Alla fine della trasmissione infatti uno streamer ha la possibilità di ospitare un altro streamer, cioè i suoi spettatori vengono sintonizzati su un'altra live. Questo sistema è fatto per permettere ad uno streamer di dare visibilità ad un altro streamer ma per i fini della nostra analisi comporta dei possibili errori. Considerare quindi gli streamer con una percentuale di sovrapposizione superiore al 10% dovrebbe permettere di mitigare questo fenomeno. Una volta ottenuta, anche questa tabella ponte viene esportata.

Al termine della fase di processing i dataset ottenuti sono quindi quelli riportati in figura 16.

1. Dataset Streamer

- *idStreamer*
- *streamer*
- *minutesLive*
- *viewerMean*
- *viewerPeak*
- *spectMean*
- *spectTot*

2. Dataset Games

- *idGame*
- *gameName*

3. Dataset Streamer-Games

- *ID_streamer*
- *ID_game*
- *minutes*

4. Dataset Streamer-Streamer

- *ID_streamer_i*
- *ID_streamer_j*
- *overlap_percentage*

Fig. 16: Dataset ottenuti durante la fase di processing.

Nella figura 17 è riportato un riassunto dei parametri e delle soglie che sono state impostate nella realizzazione dei dataset.

1.1 Setup parameters

```
interval = 5 # Interval between observations (fixed)
freq = 3 # Sample frequency (1=5min, 2=10min, 3=15min ecc.)
min_stream_time = 4 # Minimum live time parameter
min_watch_time = 2 # Minimum threshold of time as a spectator (k= min_stream_time)
min_viewer_mean = 10 # Minimum number of mean viewers
min_spect_mean = min_viewer_mean # Minimum number of mean spectator
min_overlap_percentage = 10 # Minimum overlap percentage between 2 streamers
min_game_time = 2 # Minimum number of minutes streamed for each category/game

print(f'Minimum live time threshold:\t\t\t {interval * freq * min_stream_time} min')
print(f'Minimum threshold of time as a spectator:\t {interval * freq * min_watch_time} min')
executed in 12ms, finished 18:45:07 2022-06-15

Minimum live time threshold: 60 min
Minimum threshold of time as a spectator: 30 min
```

Fig. 17: Tabella riassuntiva dei parametri impostati in fase di processing.

2. Data Enrichment

La fase di enrichment ha permesso di arricchire il contenuto informativo di *streamer_dataset* e *games_dataset* tramite un left outer join basato su instance level matching. Questo processo è stato essenziale ai fini del nostro progetto, dal

momento che ha permesso di incorporare ai dataset iniziali informazioni fondamentali per un possibile utilizzo del modello con finalità promozionali e per una possibile analisi delle community all'interno di Twitch.

a. Link social network

La prima fase di enrichment è stata effettuata sul *streamer_dataset*, contenente 2977 streamer, incorporando per ognuno di essi, ove presente, la lista di link social estratta in fase di acquisition tramite endpoint non ufficiale di Twitch, contenente 2585 streamer. È possibile che la differenza del numero di streamer sia dovuta al fatto che le due acquisizioni siano state effettuate in tempi diversi: in quell'arco temporale alcuni streamer potrebbero essere stati bannati dalla comunità virtuale oppure potrebbero aver modificato il proprio nickname. Si osserva inoltre che all'interno delle bio di streamer attivi in piattaforma non sempre sono presenti i link ai social personali e, quando presenti, essi possono comprendere link a pagine di sponsor o al proprio merchandising.

b. Informazioni di SteamDB

La seconda fase di enrichment è stata effettuata utilizzando il *games_dataset*, contenente 1144 topic/game di Twitch, ed il file JSON estratto dalla piattaforma di Steam, contenente 3498 giochi. Tale fase ha permesso di arricchire, attraverso un left outer join, il contenuto informativo del dataset dei giochi di Twitch con le informazioni relative alla data di rilascio, al nome dello sviluppatore e della casa editrice e le categorie di gioco estratte dal database digitale di Steam. Occorre evidenziare che la lista *gameName* estratta da Twitch comprende anche topic non relativi al gaming, come ad esempio la categoria *Just Chatting*, sempre più in voga, che mette lo streamer al centro dello spettacolo della live. Inoltre non tutti i videogiochi di Twitch sono presenti nella piattaforma di Steam. La fase di arricchimento riguarda quindi un subset dei vari topic/game estratti da Twitch.

Un'analisi esplorativa dei dati del file JSON ha permesso di individuare all'interno della lista di titoli dei giochi estratti da Steam la presenza di alcuni duplicati: è possibile infatti che, in seguito all'uscita di un videogioco, vengano pubblicate patch di aggiornamento per migliorare o risolvere bug ma anche per modificare contenuti, regole e algoritmi di gioco. Sono stati quindi eliminati i duplicati, mantenendo le informazioni relative alla prima data di pubblicazione, corrispondente all'effettiva data di uscita del singolo gioco. Successivamente è stato effettuato un match preliminare, senza alcuna operazione di cleaning, tra le stringhe di titoli relativi ai giochi di Twitch e dei titoli relativi ai giochi di Steam: 318 giochi sui 1144 totali di Twitch hanno prodotto un match perfetto con i titoli di Steam: per ognuno di questi match è stato associato il valore 100, corrispondente allo score massimo in termini percentuali. In seguito è stata effettuata un'operazione di cleaning sulle stringhe di titoli non ancora matchate. Sono stati normalizzati gli spazi tra le stringhe, convertiti i caratteri in minuscolo ed eliminati i vari caratteri non alfanumerici, come le emoticon, i numerosi caratteri cinesi e giapponesi, e la punteggiatura. L'operazione di cleaning ha permesso di individuare la presenza di ulteriori duplicati, che sono stati nuovamente rimossi. Il confronto

tra le liste di stringhe così normalizzate ha prodotto 84 nuovi match, a cui è stato ancora associato il punteggio massimo di score. Per effettuare il confronto tra le stringhe di titoli rimanenti infine, è stata utilizzata la libreria di Python *TheFuzz*[18]. L'algoritmo di *Fuzzy string matching* utilizza la distanza di Levenshtein, anche detta Edit Distance, normalizzata in un range tra 0 e 100, per determinare quanto due stringhe siano simili. Tale distanza, corrispondente al numero minimo di modifiche elementari (cancellazione, sostituzione o inserimento di un carattere) che consentono di trasformare la stringa A nella stringa B, era infatti la metrica più adatta alle nostre necessità. L'utilizzo di tale algoritmo ha prodotto 453 nuovi match parziali, associando ogni coppia ad uno score percentuale, e ha permesso di rintracciare match perfetti, come in figura 18.

twitch_name	steam_name
Monkey Island 2 Special Edition: LeChuck's Rev...	Monkey Island™ 2 Special Edition: LeChuck's Re...
Assassin's Creed IV: Black Flag	Assassin's Creed® IV Black Flag™
BlazBlue: Central Fiction	BlazBlue Centralfiction
Tom Clancy's The Division	Tom Clancy's The Division™
Assassin's Creed: Brotherhood	Assassin's Creed® Brotherhood
Puyo Puyo Tetris	Puyo Puyo™Tetris®
MotoGP 22	MotoGP™22
Stronghold Crusader II	Stronghold Crusader 2
Dark Messiah of Might and Magic	Dark Messiah of Might & Magic
Outlast II	Outlast 2

Fig. 18: Esempi di coppie dello stesso gioco con differenze sintattiche nelle stringhe.

L'utilizzo della distanza di Levenshtein ha permesso inoltre di attribuire un match score elevato anche alle coppie di stringhe relative a edizioni differenti di una specifica serie di videogiochi: in tal caso i videogiochi condividono ancora il set di categorie di gioco, il nome dello sviluppatore e della casa editrice. L'algoritmo tuttavia ha causato la produzione erronea di alcuni match tra stringhe di videogiochi omonimi ma differenti o tra stringhe di videogiochi con nomi sintatticamente molto simili tra loro, come in figura 19.

twitch_name	steam_name
Infermax	Inferna
Spellbreaker	Spellbreak
Cyber Hunter	Cyberhunt

Fig. 19: Esempi di coppie di giochi differenti con stringhe sintatticamente simili.

A fronte di queste osservazioni è stata effettuata un'analisi esplorativa della qualità dei fuzzy match con score elevato, maggiore di 70, seguendo un approccio di valutazione delle corrispondenze probabilistico ed empirico. Tale analisi ha permesso di rintracciare manualmente il subset di missing match corretti, associando a questi lo score massimo, e di contro rimuovere i match tra videogiochi differenti ma con nomi sintatticamente simili, associando a questi lo score nullo. L'analisi ha permesso inoltre di fissare una soglia minima di score al 79%, escludendo l'area grigia di match con score inferiore, individuando così il subset di giochi di Twitch da poter arricchire con le informazioni estratte dalla piattaforma di Steam. Infine è stato arricchito il *game_dataset* iniziale secondo lo schema seguente:

- enrichment completo con attributi *title*, *Developer*, *Publisher*, *ReleaseDate*, *categories* per match con score 100;
- enrichment parziale con attributi *title*, *Developer*, *Publisher*, *categories* per match con score nel range [79, 99].

In termini probabilistici si è quindi scelto di arricchire le corrispondenze con score massimo con tutti gli attributi di gioco estratti dalla piattaforma di Steam, compresa la data di pubblicazione, e di arricchire parzialmente le corrispondenze con score nel range [79, 99], corrispondenti a edizioni differenti di un'unica serie di videogiochi. Lo scopo di questa integrazione parziale è di permettere all'utente che interagisce con il modello finale di risalire, a partire da un singolo gioco, al nome di un'edizione precedente o successiva e ai dettagli comuni. La fase di enrichment ha permesso di ottenere la struttura finale dei dataset mostrata in figura 20.

1. Dataset Streamer

- idStreamer
- streamer
- minutesLive
- viewerMean
- viewerPeak
- spectMean
- spectTot
- socialLinks

2. Dataset Games

- idGame
- gameName
- steamName
- developer
- publisher
- releaseDate
- categories

Fig. 20: Dataset ottenuti tramite enrichment.

7. DATA MODELING AND STORAGE

Per la modellazione di questi dati si è scelto di utilizzare il modello a grafo poiché le relazioni tra le entità sono l'aspetto principale di questo progetto. Per la gestione del database si è utilizzato Neo4j [3], un software open source che permette di trattare basi di dati a grafo. In particolare è stata utilizzata la versione community server di Neo4j che permette di accedere al dataset tramite REST facendo query e ricevendo i dati in remoto.

1. Importazione dei dati

Ciò che si è ricavato dalla fase di processing sono quattro file CSV che raccolgono tutte le informazioni che andranno rappresentate nel grafo. In particolare: due di queste contengono i dettagli relativi ai diversi nodi che saranno di tipo streamer e di tipo games, mentre le altre due sono tabelle ponte che mettono in relazione gli streamer tra loro e gli streamer con la categoria che hanno streammato. In particolare, la struttura che dovrà assumere il grafo sarà di questo tipo:

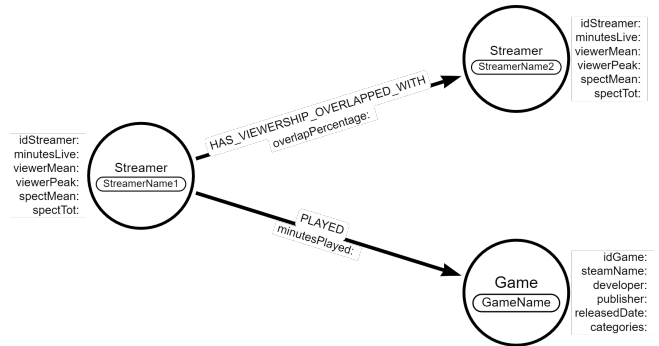


Fig. 21: Struttura dei nodi e delle relazioni.

a. Creazione constraint

Gli streamer sono quindi rappresentati dai nodi con label Streamer, i nodi giochi sono rappresentati da nodi con label Game, la relazione streamer-streamer è rappresentata da archi del tipo HAS_VIEWERSHIP_OVERLAPPED_WITH e la relazione streamer-game è rappresentata da archi del tipo PLAYED.

Il caricamento dei dati su neo4j è stato eseguito tramite REST con Python usando la libreria ufficiale neo4j. Il processo ha seguito due step principali: la definizione dei constraint per i nodi e il caricamento vero e proprio dei dati.

Il primo step prevede la creazione di constraint per i nodi streamer e i nodi game così da evitare che ci possano essere ripetizioni all'interno del database. Questa condizione è stata imposta con le seguenti due chiamate.

```
CREATE CONSTRAINT StreamerNameKey IF NOT EXISTS
FOR (s:Streamer)
  REQUIRE s.Name IS UNIQUE
```

```
CREATE CONSTRAINT GamesNameKey IF NOT EXISTS
FOR (g:Game)
  REQUIRE g.Name IS UNIQUE
```

b. Creazione grafo

Il secondo step invece è la creazione delle entità partendo dai file CSV. Sono stati quindi realizzati due tipi distinti di nodi per raccogliere streamer e giochi e due tipi di relazioni per streamer-streamer e streamer-game. Di seguito è riportata come esempio la chiamata per importare i nodi di tipo Streamer:

```
LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/gianscuri/Twitch_Community_Graph/main/DataProcessing/Streamer_dataset.csv' AS row
MERGE (s:Streamer {Name: row.streamer})
FOREACH(ignoreMe IN CASE WHEN trim(row.idStreamer)
<> "" THEN [1] ELSE [] END | SET s.idStreamer =
row.idStreamer)
FOREACH(ignoreMe IN CASE WHEN trim(row.minutesLive)
<> "" THEN [1] ELSE [] END | SET s.minutesLive =
row.minutesLive)
FOREACH(ignoreMe IN CASE WHEN trim(row.viewerMean)
<> "" THEN [1] ELSE [] END | SET s.viewerMean =
row.viewerMean)
FOREACH(ignoreMe IN CASE WHEN trim(row.viewerPeak)
<> "" THEN [1] ELSE [] END | SET s.viewerPeak =
row.viewerPeak)
```

```

FOREACH(ignoreMe IN CASE WHEN trim(row.spectMean) <>
    "" THEN [1] ELSE [] END | SET s.spectMean = row
    .spectMean)
FOREACH(ignoreMe IN CASE WHEN trim(row.spectTot) <>
    "" THEN [1] ELSE [] END | SET s.spectTot = row.
    spectTot)
FOREACH(ignoreMe IN CASE WHEN trim(row.socialLinks)
    <> "" THEN [1] ELSE [] END | SET s.socialLinks =
    row.socialLinks)
RETURN s

```

Di seguito è riportata la chiamata per importare le relazioni streamer-streamer:

```

LOAD CSV WITH HEADERS FROM "https://raw.
    githubusercontent.com/gianscuri/
    Twitch_Community_Graph/main/DataProcessing/
    Streamer-Streamer_dataset.csv" AS row
MATCH (i:Streamer {ID_streamer: row.ID_streamer_i}),
    (j:Streamer {ID_streamer: row.ID_streamer_j})
CREATE (i)-[:HAS_VIEWERSHIP_OVERLAPPED_WITH {
    overlapPercentage: toFloat(row.
    overlap_percentage)}]->(j)

```

Le relazioni tra i diversi streamer, per come è strutturato Neo4j, devono avere direzionalità anche se in questo caso si tratta di una proprietà simmetrica. Per come sono state calcolate le relazioni la direzionalità va dallo streamer con numero medio di viewer maggiore a quello minore. In fase di query è poi possibile specificare di ottenere le relazioni indipendentemente dalla direzione così da riottenere la simmetria.

2. Query

Il database così composto può essere poi interrogato per ottenere informazioni utili riguardo alle domande di ricerca. Di seguito sono riportati alcuni esempi.

a. Query 1

La seguente query è realizzata per avere una visione di come è composto il grafo.

```

MATCH p=()->()
RETURN p LIMIT 25

```

I diversi colori indicano i nodi di diversa tipologia: in viola i nodi relativi agli streamer e in arancione i nodi relativi alle categorie. È possibile inoltre vedere le due possibili relazioni tra i nodi che sono di tipo HAS_VIEWERSHIP_OVERLAPPED_WITH tra streamer e streamer e di tipo PLAYED tra streamer e game.

b. Query 2

Questa query isola gli streamer che hanno un'alta percentuale di condivisione degli spettatori (> 50%) con lo streamer "Homyatol".

```

MATCH p=(s:Streamer {Name: 'Homyatol'})-[:
    HAS_VIEWERSHIP_OVERLAPPED_WITH]-()
WHERE r.overlapPercentage > 50
RETURN p

```

Il risultato ordinato è: "roccobrazz" 70.8, "kajbomb" 67.0, "Maadux" 55.2, "StupidoHotelShow" 52.7, "LawayTV" 51.6, "panetty" 51.3, "VAXPOWER" 50.8. Da notare che la query è stata fatta ignorando la direzione della relazione così da considerare sia relazioni entranti che uscenti.

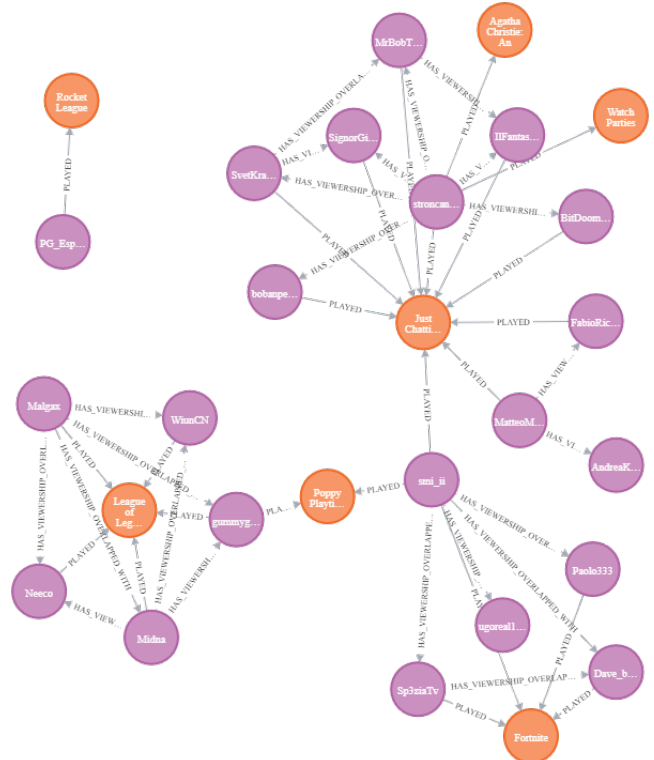


Fig. 22: Esempio query numero 1.



Fig. 23: Esempio query numero 2.

c. Query 3

Questa query permette di ottenere il dato aggregato sui minuti totali per i quali una certa categoria di videogiochi (in questo caso "Open World") è stata trasmessa dai diversi streamer.

```
MATCH p=(s:Streamer)-[r:PLAYED]->(g:Game)
WHERE g.categories CONTAINS 'Open World'
WITH sum(r.minutesPlayed) AS Totale
RETURN Totale
```

Il risultato di questa query è 540870 minuti.

3. Visualizzazione del grafo

È possibile anche visualizzare il grafo utilizzando programmi come Gephi. Questo è uno software open source che permette di modificare la dimensione dei nodi, la lunghezza degli archi e i colori in base agli attributi degli stessi. In particolare per la visualizzazione sottostante la dimensione dei nodi, e conseguentemente delle etichette, è stata scelta in base al numero medio di viewer che ha ogni streamer. In secondo luogo è stato scelto come peso per le relazioni la percentuale di sovrapposizione tra gli streamer e, grazie all'algoritmo Force Atlas opportunamente impostato, è stata generata la distribuzione dei punti nello spazio. La terza dimensione che è quella del colore è stata valutata in base alla modularità, un algoritmo che permette di individuare la presenza di community in un grafo. È curioso notare come i risultati dei due algoritmi coincidano quasi totalmente e individuino gli stessi cluster.

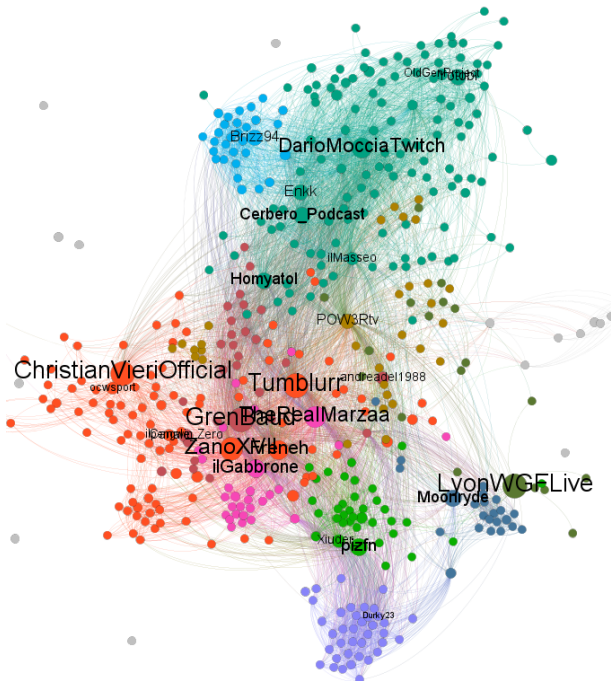


Fig. 24: Visualizzazione dei nodi streamer del grafo.

8. DATA QUALITY

1. Currency

La currency misura il livello di aggiornamento dei dati rispetto al fenomeno considerato. Una stima di tale misura

è stata effettuata a partire dal file di log come differenza tra il timestamp caratterizzante il salvataggio di un file e il timestamp relativo a una richiesta di update. Si osserva che la procedura di acquisizione di un singolo file è stata completata in media in 14 secondi.

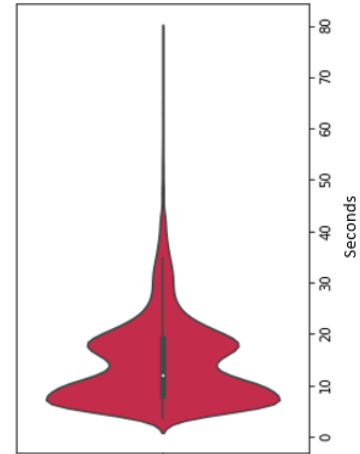


Fig. 25: Distribuzione del tempo impiegato per acquisire i file.

La distribuzione in figura 25 presenta due picchi: il primo corrisponde ai dati scaricati nelle ore notturne e al mattino, periodo in cui si registra un minor numero di live e utenti connessi, il secondo corrisponde alle ore pomeridiane e serali caratterizzate da un più alto livello di traffico sulla piattaforma. Per queste ultime l'acquisizione dei dati impiega in media 8 secondi in più.

Il livello di aggiornamento dei dati è stato inoltre influenzato dalla scelta della frequenza di update settata a 5 minuti. Tale scelta progettuale, ottenuta con più tentativi con diversi tempi di campionamento, permette di catturare il maggior numero di spostamenti degli spettatori tra live che occorrono in uno stesso intervallo di tempo.

2. Consistency

A monte dell'integrazione sono stati associati degli identificativi univoci ai singoli streamer e ai giochi. Questa scelta ha garantito la consistenza nella rappresentazione degli oggetti tra il dataset contenente i dati degli streamer, il dataset sul topic della live e la tabella ponte tra i due.

3. Completeness

La completezza misura il livello di copertura con cui il fenomeno osservato è rappresentato dall'insieme di dati. Si considera innanzi tutto che su un totale di 4032 file attesi in fase di acquisizione da Twitch sono stati registrati 3823 file. Il 5% di file mancanti è dovuto principalmente a errori di Timeout (distribuiti uniformemente lungo tutto il periodo di acquisizione) e altri errori Runtime. Tuttavia considerando l'operazione di aggregazione temporale svolta in fase di analisi non si ritiene che tale perdita sia significativa per il tracciamento dei collegamenti tra streamer.

La completezza del dataset relativo ai videogame è stata invece iniziata dalla perdita dei dati relativi al 38% dei giochi presenti sulla piattaforma SteamDB. Una simile

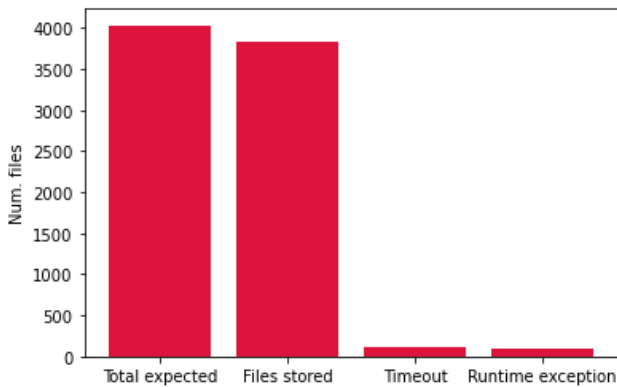


Fig. 26: Barchart del numero di file attesi, raccolti e persi per Timeout o Runtime exception.

perdita, come già spiegato nella sezione di data acquisition, è da attribuirsi a errori occorsi durante la procedura di scraping. Per quanto riguarda l'attributo completeness si osserva un livello prossimo al 100% per tutti gli attributi del dataset a eccezione di `social_link` che presenta il 13% di valori mancanti. L'8% di questi è da attribuirsi al ban dell'account o al cambio di nickname dello streamer coinvolto che ha impedito di effettuare lo scraping dei canali social, la restante parte è invece dovuta all'assenza dei link sul profilo. L'attribute completeness sui nuovi attributi nel dataset dei giochi a seguito dell'enrichment, di cui si riportano le percentuali di seguito, presenta invece alcuni valori critici.

- `idgame` 100%;
- `gameName` 100%;
- `steamName` 42%;
- `developer` 42%;
- `publisher` 20%;
- `releaseDate` 36%;
- `categories` 42%.

Come si può notare molti attributi hanno un livello di completezza inferiore al 50%, il che compromette la qualità finale dell'enrichment.

9. CONCLUSIONI E SVILUPPI FUTURI

In questo progetto sono state utilizzate le più comuni tecniche di acquisizione per ottenere i dati necessari allo scopo di analizzare le community italiane all'interno della piattaforma di livestreaming Twitch. I dati di tale piattaforma, fonte principale del progetto, sono stati acquisiti tramite API. Le tecniche di scraping sono servite invece per acquisire dati sui bot, necessari alla pulizia delle liste di spettatori di ogni streamer, e dati sui videogiochi presenti all'interno della seconda fonte principale del progetto, la piattaforma SteamDB. I file ottenuti, sia in formato JSON che CSV, sono stati sottoposti ad analisi esplorativa in cui, per mezzo della produzione di statistiche e visualizzazioni, si è cercato di cogliere le prime criticità e peculiarità nei dati. Successivamente si è passati alla pulizia e quindi al processing dei dataset, in cui si sono compiute operazioni di aggregazione e rimozione

di valori non significativi, oltre che la preparazione necessaria in vista dell'enrichment. La fase di arricchimento ha permesso di ampliare il patrimonio informativo relativo agli streamer con le informazioni dettagliate sui giochi discussi nelle live. Sono stati quindi modellati i dataset per lo storage su Neo4j, che ha permesso l'elaborazione di alcune query in relazione alle domande di ricerca poste, tra cui: *"quali streamer presentano più spettatori comuni rispetto ad un altro streamer?"*; *"per quanto tempo si è discusso di una particolare categoria di gioco nelle live?"*. Un'ulteriore analisi a cui si presta il database è inoltre lo studio delle correlazioni tra le community che si ottengono applicando vari filtri al grafo. Un esempio di domande a cui è possibile rispondere tramite le informazioni ottenute sono: *"è più ampia la community di streamer che tratta prevalentemente simulation game o quella che tratta strategy game?"*; *"quali streamer sono più reattivi rispetto alle ultime uscite sul mercato?"* (ovvero quali live trasmettono videogame appena lanciati). Oltre alle query si è esplorato il grafo attraverso il programma Gephi, per mettere in luce, anche con la visualizzazione grafica, la presenza di community nel database. In ultima analisi si è compiuta una valutazione di qualità dei dati raccolti che ha dato risultati complessivamente soddisfacenti. Tuttavia dato il livello di completezza non ottimale su alcuni degli attributi relativi ai giochi, si può considerare per uno sviluppo futuro un secondo arricchimento sfruttando come fonte una seconda piattaforma di distribuzione di giochi come Origin [19] o Battle.net [20]. La disponibilità di un catalogo più ampio di videogame potrebbe inoltre migliorare il livello di matching in fase di enrichment.

Twitch sta continuando a investire nell'area degli eSports, permettendo di organizzare tornei online, gestire e trasmettere eventi di gioco. Contemporaneamente la piattaforma ha l'ambizione di dare spazio anche a nuovi canali, avvicinando un pubblico sempre più generalista e lontano dalla natura videoludica iniziale. Un possibile sviluppo futuro di questo progetto è quindi l'estensione dell'analisi a tutte le categorie non riguardanti il mondo del gaming, come ad esempio la categoria "Just Chatting" o i vari canali dedicati alla musica dal vivo, allo scopo di individuare subcommunity in base ai topic.

Si prospetta che il mondo di Twitch sia il futuro dello streaming e della televisione: una fonte di dati sempre più preziosa.

RIFERIMENTI

- [1] "Twitch.tv." [Online]. Available: <https://twitch.tv/>
- [2] "Steamdb." [Online]. Available: <https://steamdb.info/graph/>
- [3] "Neo4j community edition 4.4.8." [Online]. Available: <https://neo4j.com/>
- [4] "Mongodb community server 5.0.9." [Online]. Available: <https://www.mongodb.com/>
- [5] "Gephi 0.9." [Online]. Available: <https://gephi.org/>
- [6] "Twitch developers." [Online]. Available: <https://dev.twitch.tv/>
- [7] "Asyncio 3.4.3." [Online]. Available: <https://docs.python.org/3/library/asyncio.html>
- [8] "Pymongo 4.1.1." [Online]. Available: <https://pymongo.readthedocs.io/en/stable/>
- [9] "Twitch gql." [Online]. Available: <https://gql.twitch.tv/gql>

- [10] “Steam.” [Online]. Available: <https://store.steampowered.com/>
- [11] “Selenium library 4.2.” [Online]. Available: <https://selenium-python.readthedocs.io/>
- [12] “Beautifulsoup library 4.9.0.” [Online]. Available: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
- [13] “Twitch insights.” [Online]. Available: <https://twitchinsights.net/bots>
- [14] “Table capture (estensione chrome browser).” [Online]. Available: <https://chrome.google.com/webstore/detail/table-capture/iebpjdmgckacbodjpjphcplhebcmeop>
- [15] “Forum viewer and spect count.” [Online]. Available: <https://help.twitch.tv/s/article/understanding-viewer-count-vs-users-in-chat>
- [16] “Pandas 1.4.2.” [Online]. Available: <https://pandas.pydata.org/>
- [17] “Numpy 1.22.” [Online]. Available: <https://numpy.org/>
- [18] “Thefuzz 0.19.0, pypi.” [Online]. Available: <https://pypi.org/project/thefuzz/>
- [19] “Origin.” [Online]. Available: <https://www.origin.com/ita/en-us/store>
- [20] “Battle.net.” [Online]. Available: <https://eu.shop.battle.net/en-us>