UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

Studio dell'impatto delle inter-case feature per la Next Activity Prediction

Study of the impact of inter-case features for Next Activity Prediction

Candidato:
Chiara Gobbi

Relatore:
Prof. Domenico Potena

Anno Accademico 2023-2024

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

---

Studio dell'impatto delle inter-case feature per la Next Activity Prediction

Study of the impact of inter-case features for Next Activity Prediction

Candidato:
Chiara Gobbi

Relatore:
Prof. Domenico Potena

Anno Accademico 2023-2024

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

Sommario

Il Predictive Process Monitoring rappresenta una delle tecniche più avanzate e promettenti nell'ambito del Process Mining e mira a prevedere l'evoluzione dei processi aziendali attraverso l'analisi di dati storici e in tempo reale.
Tra le molteplici applicazioni di questa disciplina, la predizione della prossima attività, comunemente nota come Next Activity Prediction, riveste un ruolo centrale poiché permette di ottimizzare i flussi di lavoro, prevenire inefficienze e identificare tempestivamente eventuali colli di bottiglia.

Questa tesi si concentra sull'esplorazione delle inter-case feature, cioè informazioni provenienti dall'interconnessione tra esecuzioni diverse di un processo, e sul loro impatto nella predizione delle attività future. Tradizionalmente, le tecniche di predizione si basano esclusivamente su dati di singole tracce, trascurando così le relazioni che possono esistere tra diverse esecuzioni del processo. L'integrazione di inter-case feature consente invece di catturare pattern che emergono solo attraverso l'analisi congiunta di più tracce.

L'approccio proposto modella le esecuzioni di processo come grafi, noti come Instance Graph, per rappresentare in modo strutturato ogni traccia ed evidenziare i parallelismi tra attività concorrenti. Questi grafi vengono poi elaborati utilizzando reti neurali convoluzionali su grafi (Deep Graph Convolutional Neural Network), che permettono di estrarre caratteristiche complesse dalle connessioni tra gli eventi e tra le diverse tracce.

Gli esperimenti sono stati condotti su un dataset reale e i risultati ottenuti permettono di investigare come le inter-case feature influenzino il processo di predizione rispetto ai metodi che considerano una singola traccia. In particolare, il lavoro ha esplorato il comportamento di queste caratteristiche quando integrate nei modelli predittivi, evidenziando vantaggi e limitazioni.
Sebbene l'inclusione di tali informazioni abbia mostrato un potenziale miglioramento nelle performance delle predizioni, sono emerse anche problematiche legate all'aumento della complessità computazionale e alla gestione delle dimensioni del modello, in particolare a causa della crescita della struttura dei grafi in presenza di tracce parallele.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Nel contesto attuale, le organizzazioni sono continuamente alla ricerca di modi per ottimizzare i propri processi aziendali, migliorare l'efficienza operativa e mantenere un vantaggio competitivo.
A tale scopo, la crescente digitalizzazione e la grande quantità di dati generati dai sistemi informativi aziendali permettono un'analisi approfondita dei processi grazie a nuove tecnologie. Una di queste tecnologie emergenti è il *Process Mining*.
Il Process Mining rappresenta una vera e propria rivoluzione nella gestione dei processi aziendali combinando tecniche di *Data Mining* e modellazione dei processi per estrarre conoscenza utile dagli eventi registrati. Questa disciplina, infatti, propone di ottenere una visione chiara e dettagliata dei processi reali, identificando inefficienze, colli di bottiglia, deviazioni delle procedure standard e opportunità di miglioramento.
Come mostrato in Figura 1.1, il problema è che, nella realtà operativa delle aziende, i processi non seguono quasi mai una sequenza lineare e prevedibile di attività. Al contrario, i processi reali tendono ad essere complessi e dinamici.
Questa variabilità è dovuta a diversi fattori tra cui le interazioni umane, le eccezioni nei flussi di lavoro e le interdipendenze con altri processi.
Per tale ragione, i metodi di analisi tradizionali non sono spesso in grado di catturare la realtà operativa in modo accurato. Questo si traduce in una visione incompleta e potenzialmente fuorviante dei processi aziendali.
Il Process Mining si distingue quindi per essere una soluzione innovativa, offrendo una visione più dinamica.

Il punto di partenza per il Process Mining è un log di eventi. Ogni evento rappresenta un'attività ed è associato ad un *case*, ovvero, ad un'istanza di processo.
I log possono anche contenere informazioni aggiuntive sugli eventi come la risorsa che esegue o avvia l'attività o il timestamp dell'evento.
I log possono essere utilizzati per condurre tre tipi di analisi dei processi.
Il *Process Discovery*, a partire da un log di eventi, produce un modello di processo che permette di evidenziare i parallelismi o cicli tra le attività che non sarebbero catturabili dai log a causa della loro natura sequenziale.
Il secondo tipo di analisi che è possibile effettuare prende il nome di *Conformance Checking*. In questo caso, un modello di processo esistente viene confrontato con

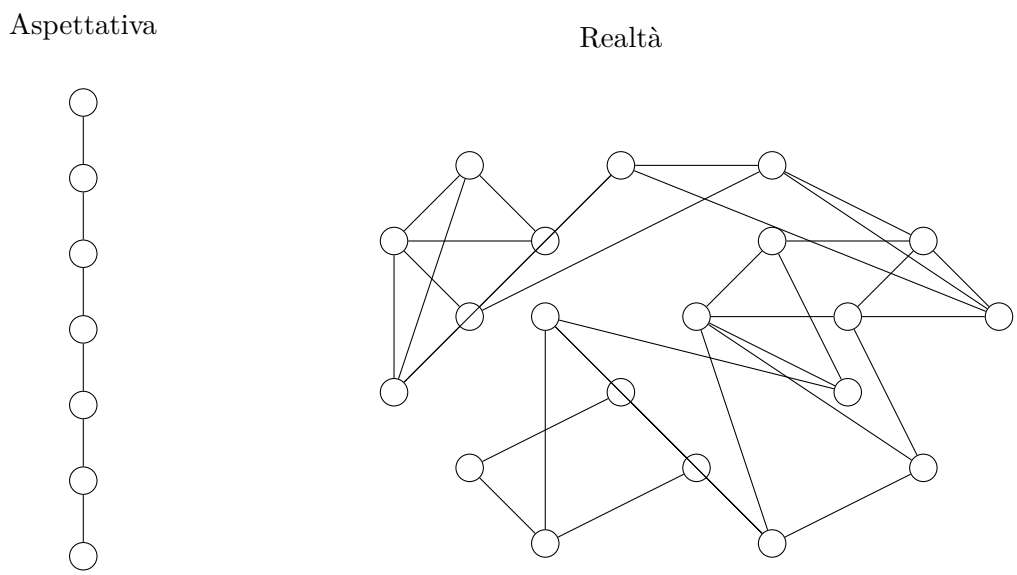


Figura 1.1: Variabilità nei processi reali

un log degli eventi dello stesso processo. Il controllo della conformità può essere utilizzato per verificare se la realtà, come registrata nel log, è conforme al modello e viceversa.
Infine, il *Process Enhancement* permette di estendere o migliorare un modello di processo esistente utilizzando informazioni registrate in un log degli eventi. In questo caso, lo scopo è quello di estendere il modello per eliminare colli di bottiglia e migliorare i tempi di throughput. [Aalst, 2012]

In questo contesto, il *Process Monitoring* svolge un ruolo cruciale in tutte e tre le aree del Process Mining e si concentra sull'osservazione continua e sull'analisi dei processi in tempo reale o su base periodica.
In particolare, utilizzando i dati di log, il Process Monitoring offre una visione aggiornata dello stato dei processi aziendali permettendo di identificare rapidamente eventuali deviazioni o inefficienze.

In questa direzione, il *Predictive Process Monitoring* si inserisce nel contesto del Process Mining come un avanzamento significativo rispetto al tradizionale Process Monitoring.
Infatti, se il Process Monitoring fornisce una visione attuale e storica dei processi, il Predictive Process Monitoring utilizza tecniche predittive per anticipare e prevenire problemi futuri, contribuendo ad una gestione dei processi più proattiva e strategica.
Le tipologie di previsione previste dal Predictive Process Monitoring possono riguardare l'esito di un'esecuzione del processo (*Outcome-based Predictions*), possono essere relative a misure di interesse che assumono valori numerici o continui (*Numeric Value predictions*) oppure possono riguardare le sequenze di future attività e i relativi dati

associati (*Next Event Prediction*). [Di Francescomarino and Ghidini, 2022]
Per quanto riguarda il task di Next Event Prediction, le previsioni del prossimo evento possono riguardare l'evoluzione degli eventi futuri di una traccia in corso incompleta, la sequenza delle classi degli eventi successivi o i prossimi dati associati agli eventi, come ad esempio i timestamp o le risorse collegate ai prossimi eventi.
Nel caso di previsioni delle attività future (*Next Activity Prediction*), la maggior parte degli approcci consiste nel predirre la classe dell'evento che si verificherà al passo temporale immediatamente successivo a quello corrente per poi seguire con la predizione delle prossime attività in modo iterativo, ovvero, aggiungendo di volta in volta l'attività predetta al passo precedente.

L'obiettivo di questo elaborato è quello di studiare l'impatto delle inter-case feature per il task di Next Activity Prediction.
Le feature inter-case rappresentano informazioni provenienti da tracce diverse, che eseguono in parallelo rispetto alla traccia di riferimento. In questo modo, non ci si limita a considerare esclusivamente l'esecuzione della singola traccia corrente, ma si includono anche influenze esterne derivanti da altre tracce in corso, arricchendo il modello predittivo con dati che riflettono le interazioni tra più esecuzioni.
In particolare, dato un log di eventi e ricavato un modello di processo espresso sotto forma di rete di Petri, ogni traccia viene convertita in una struttura a grafo detta *Instance Graph*. In questi grafi, ogni nodo rappresenta l'esecuzione di un singolo evento all'interno della traccia, mentre gli archi catturano le relazioni temporali e di dipendenza tra gli eventi.
Tali rappresentazioni a grafo non si limitano a descrivere la semplice sequenza degli eventi, ma vengono arricchite con una serie di feature, ottenute direttamente dai dati contenuti nel log, che descrivono ulteriori proprietà e contesti specifici di ogni evento.
A partire dagli Instance Graphs è poi possibile ricavare i cosiddetti prefissi (*Prefix Instance Graph*), ovvero, esecuzioni parziali di una traccia, che verranno utilizzati come input per una Graph Neural Network per predirre la prossima attività.
Si vuole comprendere come cambia la predizione della Next Activity non considerando solo l'esecuzione della traccia corrente con le relative feature ma tenendo anche in conto di eventuali tracce che eseguono in parallelo.

L'obiettivo è quello di capire se questo approccio possa contribuire ad identificare colli di bottiglia nelle risorse utilizzate e a valutare l'efficienza complessiva della gestione.
Infatti, quando i processi lavorano simultaneamente, le interazioni e le sovrapposizioni tra di essi possono rivelare aree critiche dove le risorse potrebbero essere sovraccariche o mal allocate. Ad esempio, un processo che dipende da una risorsa condivisa potrebbe rallentare se questa risorsa è coinvolta in altri processi contemporaneamente. Allo stesso modo, la gestione inefficiente di uno o più processi paralleli può causare ritardi e ridurre la qualità complessiva del servizio o del prodotto.

Per questo motivo, tenendo anche in conto delle esecuzioni parallele, è possibile ottenere informazioni preziose sull'ottimizzazione delle risorse e scoprire opportunità per migliorare i processi stessi e la performance complessiva dell'azienda.

Capitolo 2

Letteratura e stato dell'arte

Il Predictive Process Monitoring è una tecnica del Process Mining che utilizza i dati storici di eventi per prevedere l'evoluzione di un'istanza di processo. Questa capacità di fare previsioni è utile per anticipare problemi, ottimizzare risorse e fare raccomandazioni.

Le tecniche tradizionali di machine learning applicate al monitoraggio predittivo includono alberi decisionali, macchine a fattorizzazione e automi finiti probabilistici. Tuttavia, negli ultimi anni, l'avvento delle Deep Neural Network ha catturato grande attenzione. In particolare, le Reti Neurali Ricorrenti (RNN) e le loro varianti come le Long Short-Term Memory (LSTM) e le Gated Recurrent Unit (GRU) sono state utilizzate per il monitoraggio predittivo grazie alla loro capacità di gestire la natura sequenziale dei log degli eventi.[Efrén et al., 2023]

Negli ultimi anni, il numero di architetture di reti neurali, metodi di codifica e task predittivi è aumentato considerevolmente, offrendo ai ricercatori una vasta gamma di approcci da esplorare e confrontare. In questo capitolo, verranno presentate le principali tecniche di Process Monitoring mettendo in luce le diverse sfide relative a ciascun approccio.

Per fornire una panoramica generale del flusso del Predictive Process Monitoring, la Figura 2.1 mostra uno schema che illustra le principali tecniche e approcci utilizzati nel monitoraggio predittivo. Questa figura aiuta a visualizzare le relazioni tra i diversi metodi e le loro applicazioni.

2.1 Approcci al Predictive Process Monitoring

Gli approcci di Predictive Process Monitoring (PPM) possono essere distinti in due categorie principali: model-aware e model-agnostic. La differenza fondamentale tra questi due approcci risiede nell'utilizzo, o meno, di un modello formale del processo.

Gli approcci model-aware utilizzano un modello formale del processo per fare previsioni. Questo modello può essere una rete di Petri, un diagramma di flusso o

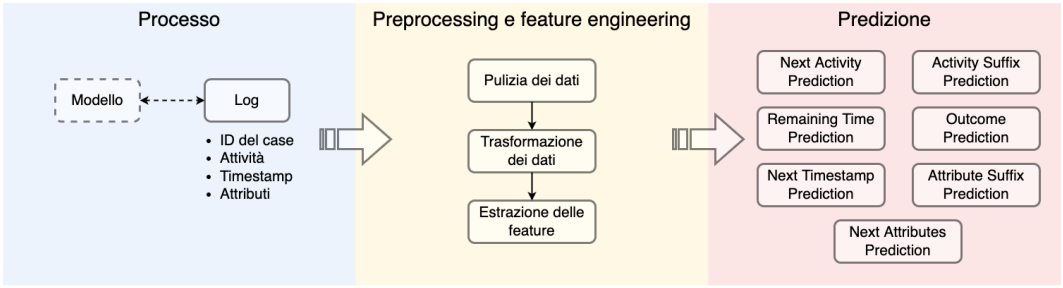


Figura 2.1: Rappresentazione Generale del Flusso del Predictive Process Monitoring

qualsiasi altra rappresentazione formale che descrive le attività del processo e le loro interazioni.
Questi approcci sfruttano le informazioni strutturali del processo per migliorare la capacità predittiva. Ad esempio, conoscendo la sequenza di attività e le possibili transizioni, avendo a disposizione il modello, è possibile fare previsioni più accurate sulla prossima attività o sul tempo di completamento della traccia.
Tuttavia, questi approcci possono essere meno efficaci quando il modello formale non cattura tutte le varianti o gli scenari eccezionali del processo reale. In altre parole, se il modello è troppo rigido, troppo semplificato oppure se è troppo complesso, le previsioni possono essere imprecise in presenza di deviazioni rispetto al comportamento previsto.

La principale criticità risiede nel gran numero di dati da processare. Da questo punto di vista, si è soliti distinguere tra approcci filtranti e non filtranti.
Gli approcci non filtranti cercano di rappresentare tutti i comportamenti registrati nel log senza escludere alcuna informazione. Questo porta spesso a modelli che tendono a generalizzare eccessivamente il processo, permettendo praticamente ogni ordine di esecuzione tra le attività. Come conseguenza, i modelli risultano spesso complessi e difficili da interpretare.
Al contrario, gli approcci filtranti cercano di costruire un modello del processo che si concentra sui comportamenti più frequenti e rilevanti, escludendo quelli meno comuni. Questo porta a modelli più semplici e interpretabili ma potrebbe escludere attività o sequenze di attività che si verificano più raramente.

Gli approcci model-agnostic, d'altra parte, non fanno affidamento su un modello formale del processo ma si basano esclusivamente sui dati contenuti nei log.
Questo li rende più flessibili e capaci di adattarsi a varianti di processo e a comportamenti anomali, poiché non sono vincolati da un modello predefinito.
Tuttavia, la mancanza di un modello strutturale può rendere difficile catturare le relazioni complesse tra le attività e questo può limitare la precisione delle previsioni.

2.1.1 Rappresentazione a grafo delle tracce

La rappresentazione delle tracce come grafi è profondamente connessa al modello di processo e questo legame porta numerosi vantaggi.

Innanzitutto, i modelli di processo, come le reti di Petri, possono essere visti come grafi dove i nodi rappresentano le attività del processo e gli archi indicano le transizioni o le dipendenze tra le attività. Questo tipo di rappresentazione permette di catturare la struttura del processo in modo molto chiaro evidenziando le sequenze di attività, i loop e i parallelismi.
Quando un processo viene eseguito, le tracce registrano la sequenza delle attività che sono state effettivamente svolte.
La rappresentazione a grafo delle tracce stesse è molto utile perché permette di mantenere una coerenza con il modello di processo originale. Infatti, le strutture presenti nel modello di processo, come attività parallele o cicli, possono essere conservate anche nelle rappresentazioni delle tracce, rendendo più facile analizzare come le esecuzioni reali si configurano rispetto al modello teorico.

La rappresentazione a grafo permette anche di superare la visione monodimensionale dei log. Infatti, in un log tradizionale, ogni traccia è rappresentata come una sequenza lineare senza tenere in conto di informazioni sulle relazioni parallele o cicliche tra le attività. Al contrario, un grafo può rappresentare direttamente queste relazioni complesse, offrendo una visione multidimensionale del processo.

Questo permette anche di identificare facilmente le varianti del processo o i comportamenti anomali. Infatti, confrontando le tracce di esecuzione con il modello formale, è possibile osservare come le esecuzioni differiscono dal comportamento atteso.
I grafi, inoltre, forniscono un modo chiaro e visivo per vedere come le diverse parti del processo si collegano tra loro, facilitando la comprensione e l'analisi.

A tal proposito, l'algoritmo Building Instance Graphs (BIG) [Diamantini et al., 2016] permette di costruire grafi a partire dalle tracce presenti sul log e utilizzando tecniche di Process Discovery filtranti per catturare le relazioni causali tra le attività più rilevanti e limitare l'over-generalizzazione.

2.2 Preprocessing dei dati contenuti nei log

La selezione dei dati di input è una decisione cruciale nella progettazione di un approccio predittivo. [Efrén et al., 2023]

Nella letteratura, molti dei log di eventi che vengono utilizzati provengono da processi reali. Questi log molto spesso contengono dati rumorosi, come dati mancanti, attività

duplicate o inconsistenze.
Poichè questi problemi rendono difficile l'estrazione accurata dei processi nei sistemi di Predictive Monitoring, per migliorare le previsioni, è possibile applicare opportune tecniche di ricostruzione dei dati ai log di eventi per ridurre l'impatto del rumore.
Ad esempio, l'algoritmo BIG ([Diamantini et al., 2016]) combina l'utilizzo di tecniche di filtraggio con una tecnica originale mirata alla riparazione dei grafi.

In generale, gli attributi informativi più rilevanti in un log sono l'identificativo dell'evento (*case*), le attività e il timestamp.
Le informazioni temporali disponibili nei log richiedono un pre-processing per essere utilizzate come input ad una rete neurale. Solitamente, il pre-processing dei timestamp comporta il calcolo della differenza tra il timestamp dell'evento corrente e quello dell'evento precedente, o tra il timestamp dell'evento corrente e l'inizio della traccia. Altre informazioni utili possono, ad esempio, riguardare il giorno della settimana.

In generale, non tutti gli attributi del log aggiungono informazioni significative al problema predittivo.
Alcuni approcci utilizzano algoritmi come X-means per raggruppare gli attributi in cluster, i quali possono essere successivamente analizzati per identificare le caratteristiche più rilevanti [Efrén et al., 2023].

2.3 Tipologie di predizione

Per quanto riguarda le tipologie di *prediction target*, ci sono diverse possibilità:

- *Next Activity Prediction*: predizione della prossima attività di una traccia in corso;
- *Activity Suffix Prediction*: predizione della sequenza di attività data una traccia in corso;
- *Next Timestamp Prediction*: predizione della durata della prossima attività;
- *Remaining Time Prediction*: predizione della differenza tra il timestamp dell'ultima attività di una traccia e il timestamp corrente di un evento;
- *Outcome Prediction*: predizione dell'esito di una data traccia in corso;
- *Next Attributes Prediction*: predizione di attributi associati ad eventi futuri o ad eventi in corso;
- *Attribute Suffix Prediction*: previsione della sequenza degli attributi associati agli eventi che seguiranno la traccia in corso.

Nel proseguo, verranno esaminati degli approcci utilizzati per affrontare alcune delle diverse tipologie di predizione descritte. Sebbene ogni metodo si concentri su aspetti specifici, è fondamentale riconoscere che le problematiche e le considerazioni metodologiche sono spesso comuni a tutte queste tecniche.

2.3.1 Outcome Prediction

Nel contesto competitivo del *retail*, la capacità di prevedere il comportamento futuro dei clienti è cruciale per mantenere e aumentare la fedeltà dei clienti.
Una delle applicazioni più importanti è la predizione del *churn*, ovvero l'identificazione dei clienti che potrebbero abbandonare un certo servizio.
La predizione degli outcome, come il churn, richiede modelli sofisticati e adattivi che possano gestire le complessità e le dinamiche dei dati reali.

In questo campo, la metodologia TSUNAMI [Scardapane et al., 2023] rappresenta un approccio avanzato, affrontando in modo efficace diverse sfide critiche e garantendo un vantaggio competitivo significativo per le aziende.
L'approccio TSUNAMI, evidenzia e concentra l'attenzione su alcuni problemi comuni come lo sbilanciamento tra le classi in quanto questo squilibrio può portare i modelli di machine learning a trascurare i churner, riducendo l'efficacia delle predizioni.
TSUNAMI affronta il problema del class imbalance con diverse strategie. Utilizza tecniche di oversampling per aumentare artificialmente la classe minoritaria (churner) o undersampling per ridurre la classe maggioritaria (non churner).
Inoltre, impiega tecniche per generare dati sintetici della classe minoritaria migliorando così l'addestramento del modello.
Un'altra questione che viene evidenziata è il *drifting* che si verifica quando le caratteristiche che determinano l'outcome cambiano nel tempo a causa di nuove tendenze, cambiamenti o evoluzioni nei comportamenti dei clienti.
Questo richiede che i modelli siano adattivi e capaci di evolversi con i dati.
Per questo, TSUNAMI implementa un apprendimento continuo aggiornando il modello con nuovi dati per garantire che questo rimanga accurato e rilevante.

Questo esempio evidenzia come il Predictive Process Monitoring sia cruciale per le aziende moderne, in quanto consente di anticipare eventi futuri basandosi su dati storici e in tempo reale.
Questo approccio è fondamentale per migliorare l'efficienza operativa, ottimizzare i processi aziendali e aumentare la soddisfazione dei clienti.
Implementare soluzioni come TSUNAMI offre un vantaggio competitivo significativo, permettendo alle aziende di adattarsi rapidamente ai cambiamenti del mercato e di prendere decisioni basate su previsioni accurate e tempestive.

2.3.2 Next Activity Prediction

La previsione della prossima attività consiste nel determinare quale sarà l'attività successiva di un processo dato un elenco parziale delle attività già svolte.
Questo è il problema più comune nel monitoraggio predittivo dei processi e viene adottato per ottimizzare i flussi di lavoro e prevedere i colli di bottiglia in un processo aziendale.

In questo ambito, l'approccio [Chiorrini et al., 2023] propone una metodologia innovativa per la previsione della prossima attività in un processo aziendale.
L'approccio consiste nel rappresentare ogni traccia di eventi con un Instance Graph (IG). Questi grafi vengono poi arricchiti con informazioni aggiuntive riguardanti l'esecuzione sequenziale e con attributi temporali.
Successivamente, gli IG vengono elaborati tramite reti neurali progettate appositamente per lavorare con strutture dati a grafo, per addestrare un classificatore per la predizione della prossima attività.

2.4 Architetture di Deep Learning per il Monitoraggio Predittivo

Nel contesto del monitoraggio predittivo dei processi aziendali, [Efrén et al., 2023] esplora e confronta diverse architetture di deep learning, evidenziandone le caratteristiche distintive e i vantaggi in relazione a specifici scenari applicativi.

Le Reti Neurali Ricorrenti (RNN), in particolare le Long Short-Term Memory (LSTM) e le Gated Recurrent Units (GRU), sono spesso preferite per la loro capacità di gestire sequenze temporali, preservando informazioni rilevanti anche su sequenze temporali estese.
Questa caratteristica le rende particolarmente adatte a processi con dipendenze a lungo termine. Tuttavia, le RNN possono incontrare difficoltà nell'apprendimento di sequenze molto lunghe a causa del fenomeno del *vanishing gradient*, in cui i segnali di aggiornamento dei pesi diventano progressivamente più deboli durante l'addestramento, riducendo l'efficacia del modello in tali contesti.

In risposta a queste limitazioni, vengono utilizzate le RNN bidirezionali con meccanismi di attenzione che offrono un'analisi più ricca e completa del contesto elaborando le sequenze sia in avanti che all'indietro. Questo approccio permette di pesare in modo differente gli eventi più significativi, migliorando la capacità predittiva del modello. Tuttavia, nonostante i benefici apportati, queste reti possono risultare complesse da addestrare e richiedere risorse computazionali più elevate.

Le reti neurali con memoria aumentata, come le Neural Turing Machines (NTM) e i Differentiable Neural Computers (DNC), rappresentano un ulteriore passo avanti, introducendo unità di memoria esterne per gestire in modo efficiente le dipendenze a lungo termine. Questi modelli sono particolarmente efficaci quando si tratta di sequenze di eventi molto lunghe. Tuttavia, la loro complessità strutturale e computazionale può limitarne l'applicazione in contesti dove la semplicità e la velocità di esecuzione sono cruciali.

Di recente, i Transformer, che si basano esclusivamente su meccanismi di attenzione piuttosto che sulla ricorrenza, hanno guadagnato popolarità grazie alla loro velocità di addestramento e inferenza. Tuttavia, il loro utilizzo nel monitoraggio predittivo dei processi aziendali è ancora limitato da sfide tecniche, come la gestione di dati eterogenei. Questa eterogeneità richiede spesso pre-elaborazioni complesse o modifiche architetturali per sfruttare al meglio le capacità dei Transformer. Queste sfide tecniche possono limitarne l'efficacia o l'adozione immediata in ambiti come il monitoraggio predittivo dei processi aziendali.
Nonostante ciò, i Transformer rappresentano una promettente area di ricerca, in particolare per applicazioni che richiedono l'elaborazione di grandi volumi di dati in tempi ridotti.

Infine, le Reti Neurali Convoluzionali (CNN), tradizionalmente utilizzate per il riconoscimento di immagini, possono essere adattate per la previsione di sequenze trattando i dati sequenziali come griglie unidimensionali. Le CNN offrono il vantaggio di apprendere caratteristiche invarianti rispetto alla posizione, ma possono risultare meno efficaci rispetto alle RNN o ai Transformer in applicazioni che richiedono la gestione di dipendenze temporali complesse.

Rispetto ad approcci tradizionali come i modelli basati su regole o le tecniche statistiche, le architetture di deep learning offrono una maggiore capacità di generalizzazione e una maggiore flessibilità nell'adattarsi a dati complessi e variabili. Tuttavia, questa flessibilità comporta un aumento della complessità computazionale e richiede una quantità significativa di dati di addestramento per raggiungere prestazioni ottimali. Questo evidenzia come la scelta dell'architettura più appropriata dipende fortemente dalle caratteristiche specifiche del processo aziendale e dal tipo di dati disponibili.

Capitolo 3

Metodologia

L'approccio proposto in questo elaborato è mostrato in Figura 3.1.

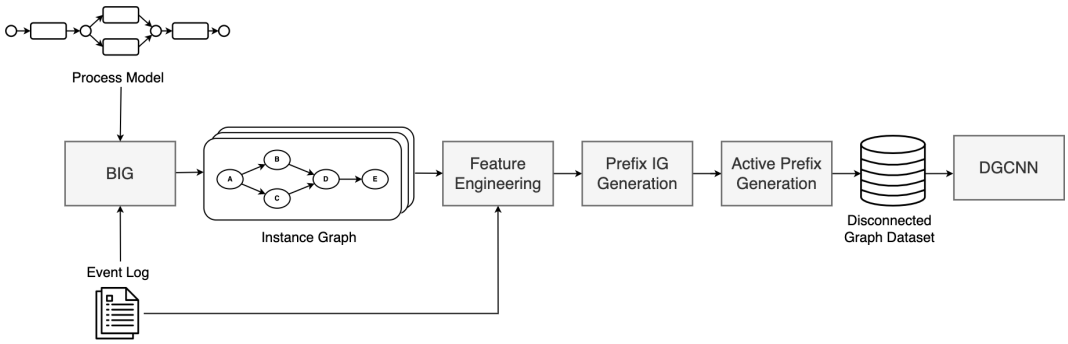


Figura 3.1: Metodologia

Dato un log di eventi e il relativo modello di processo espresso sotto forma di rete di Petri, ogni traccia viene rappresentata come un grafo orientato detto Instance Graph (IG). La costruzione degli IG avviene tramite l'algoritmo Building Instance Graphs (BIG) [Diamantini et al., 2016] che è robusto contro la possibile presenza di outlier o comportamenti anomali.
I grafi ottenuti vengono poi arricchiti con attributi aggiuntivi calcolati a partire dalle informazioni contenute nel log.
Successivamente, a partire dagli IG, vengono ricavati i Prefix Instance Graph, ovvero, particolari sotto-grafi che rappresentano le esecuzioni parziali delle tracce.
Tutti i prefissi relativi al medesimo IG hanno il nodo di partenza in comune e, seguendo gli archi orientati dell'IG, sono costruiti iterativamente aggiungendo di volta in volta il nodo in posizione successiva. Questo permette anche di etichettare i prefissi con la label dell'attività immediatamente successiva all'ultima aggiunta.
I Prefix Instance Graph sono poi ulteriormente integrati con i *Prefissi Attivi*, ovvero, prefissi di altre tracce che vengono eseguiti in parallelo a quello corrente.
L'insieme di questi grafi disconnessi è poi processato attraverso reti neurali progettate appositamente per lavorare con strutture dati a grafo in modo da addestrare un classificatore per la Next Activity Prediction. Tra le varie architetture deep proposte in letteratura, è stato scelto di adottare la Deep Graph Convolutional Neural Network (DGNN) [Zhang et al., 2018] [Chiorrini et al., 2023].

3.1 Premesse Teoriche

Prima di entrare nel dettaglio dei diversi step della metodologia, è importante introdurre la nomenclatura che verrà utilizzata nelle sezioni successive.

Definizione 1 (Rete di Petri etichettata)**.** Una rete di Petri etichettata è una tupla (P, T, F, A, l) dove:

- P è un insieme di place,
- T è un insieme di transizioni,
- $F \subseteq (P \times T) \cup (T \times P)$ è la relazione di flusso che collega place e transizioni,
- A è un insieme di etichette per le transizioni,
- $l : T \nrightarrow A$ è una funzione parziale che associa un'etichetta a un sottoinsieme delle transizioni in T.

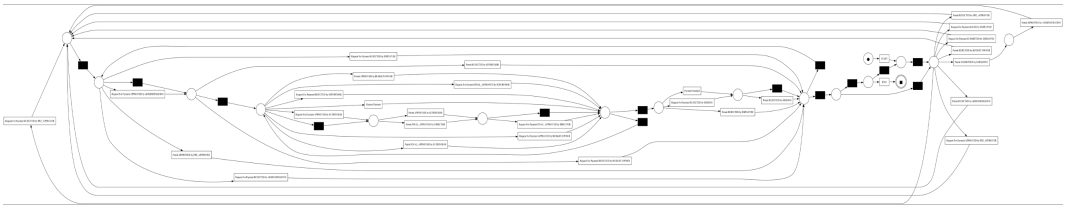


Figura 3.2: Petri Net relativa al dataset *Prepaid Travel Costs* ricavata tramite l'algoritmo Inductive Miner della libreria Python *pm4py*

In generale, il modello di un processo è una rappresentazione che descrive come sono organizzate un insieme di attività per raggiungere un obiettivo specifico. Questa rappresentazione aiuta a comprendere, analizzare e ottimizzare il flusso di lavoro in vari contesti, dalle operazioni aziendali ai progetti complessi.
Ad esempio, Figura 3.2 mostra la rete di Petri derivata da un processo reale relativo alle richieste di rimborso per spese di viaggi. [van Dongen, 2020]
Come è possibile notare, il modello di processo segue un flusso strutturato che parte dalla sottomissione della richiesta di rimborso e attraversa vari livelli di approvazione, culminando nel pagamento del rimborso o nella gestione delle spese di viaggio. Ogni fase del processo ha il suo ruolo per garantire che le spese siano appropriate e giustificate.

Definizione 2 (Evento, Traccia, Log)**.** Sia A_L l'insieme di tutti i nomi delle attività, C l'insieme di tutti gli identificatori dei case (ovvero, istanze di processo), H l'insieme di tutti i timestamp, U un insieme di valori variabili, e V un insieme di nomi di variabili. Un evento è una tupla $e = (a, D, c, i, t) \in A_L \times (V \nrightarrow U) \times C \times \mathbb{J} \times H$ che

consiste in un'attività eseguita $a \in A_L$, una funzione D che assegna un valore ad alcune variabili di processo (possibilmente tutte), un identificatore di case $c \in C$ e un numero $i \in \mathbb{N}$. Un case corrisponde a una singola esecuzione del processo; il numero i identifica la posizione dell'evento all'interno della sequenza di eventi che si sono verificati all'interno di un case. L'insieme degli eventi è denotato da $\mathcal{E}$. Una traccia di eventi $\sigma_L \in \mathcal{E}^*$ è una sequenza di eventi con lo stesso identificatore di case. Un log di eventi è un multi-insieme di tracce di eventi L.

Dunque, un evento è una tupla che rappresenta una specifica azione in un processo, con dettagli come l'attività eseguita, i dati associati e i tempi.
Si consideri, ad esempio, un evento e che rappresenta la sottomissione di una richiesta di rimborso. Questo evento può essere descritto come una tupla $e = (a, D, c, i, t)$ dove:

- a è l'attività eseguita, ad esempio, *sottomissione della richiesta*,
- D è una funzione che associa valori alle variabili di processo. Ad esempio, $D = \{\text{importo} \rightarrow 500\}$,
- c è l'identificatore del case, ad esempio, $c = 125$,
- i è la posizione dell'evento nella sequenza, ad esempio, $i = 1$,
- t è il timestamp dell'evento, ad esempio, $t =$ 2024-08-21 10:00.

L'evento può essere scritto come:

$$e = (\text{sottomissione della richiesta}, \{\text{importo} \rightarrow 500\}, 125, 1, \text{2024-08-21 10:00})$$

Una traccia di eventi è una sequenza di eventi che appartengono allo stesso case. Si supponga di avere una traccia di eventi per un caso di rimborso. La traccia σ_L è una sequenza di eventi che fanno riferimento allo stesso identificatore di case. Ad esempio:

$$\sigma_L = \langle e_1, e_2, e_3 \rangle$$

dove:

- $e_1 = (\text{Sottomissione della richiesta}, \{\text{importo} \rightarrow 500\}, 125, 1, \text{2024-08-21 10:00})$
- $e_2 = (\text{Approvazione della richiesta}, \{\text{importo} \rightarrow 500\}, 125, 2, \text{2024-08-22 09:00})$
- $e_3 = (\text{Pagamento del rimborso}, \{\text{importo} \rightarrow 500\}, 125, 3, \text{2024-08-23 14:00})$

La traccia di eventi σ_L rappresenta il flusso completo degli eventi che fanno riferimento al case 125.
Ogni traccia rappresenta la sequenza di eventi per un case specifico, mentre, il log contiene le tracce relative a tutti i case. Ad esempio:

$$L = \{\sigma_1, \sigma_2\}$$

dove:

- $\sigma_1 = \langle e_{1,1}, e_{1,2}, e_{1,3} \rangle$ rappresenta la traccia relativa al case 125,
- $\sigma_2 = \langle e_{2,1}, e_{2,2}, e_{2,3} \rangle$ rappresenta la traccia relativa al case 126.

Inoltre:

- $e_{1,1} = (\text{Sottomissione della richiesta}, \{\text{importo} \rightarrow 500\}, 125, 1, \text{2024-08-21 10:00})$,
- $e_{1,2} = (\text{Approvazione della richiesta}, \{\text{importo} \rightarrow 500\}, 125, 2, \text{2024-08-22 09:00})$,
- $e_{1,3} = (\text{Pagamento del rimborso}, \{\text{importo} \rightarrow 500\}, 125, 3, \text{2024-08-23 14:00})$,
- $e_{2,1} = (\text{Sottomissione della richiesta}, \{\text{importo} \rightarrow 300\}, 126, 1, \text{2024-08-20 11:00})$,
- $e_{2,2} = (\text{Approvazione della richiesta}, \{\text{importo} \rightarrow 300\}, 126, 2, \text{2024-08-21 10:00})$,
- $e_{2,3} = (\text{Pagamento del rimborso}, \{\text{importo} \rightarrow 300\}, 126, 3, \text{2024-08-22 13:00})$.

Definizione 3 (Prefisso di una traccia)**.** Un prefisso di lunghezza k di una traccia $\sigma = \langle e_1, e_2, \ldots, e_n \rangle \in \mathcal{E}^*$ è una traccia $p_k(\sigma) = \langle e_1, e_2, \ldots, e_k \rangle \in \mathcal{E}^*$ dove $k \leq n$.

Ad esempio, si consideri una traccia σ composta dai seguenti eventi:

$$\sigma = \langle e_1, e_2, e_3, e_4 \rangle$$

dove:

- $e_1 =$ sottomissione della richiesta;
- $e_2 =$ approvazione della richiesta;
- $e_3 =$ pagamento del rimborso;
- $e_4 =$ chiusura del caso.

Un prefisso di lunghezza $k = 2$ è:

$$p_2(\sigma) = \langle e_1, e_2 \rangle$$

Questo prefisso rappresenta i primi due eventi della traccia originale:

- $e_1 =$ sottomissione della richiesta;
- $e_2 =$ approvazione della richiesta.

In generale, la traccia originale ha $n = 4$ eventi. Un prefisso di lunghezza k contiene solo i primi k eventi di σ. In questo caso, se $k = 2$, il prefisso include solo i primi 2 eventi.

Definizione 4 (Relazione causale)**.** Una Relazione Causale (CR) è una relazione sull'insieme delle attività, $CR \subseteq A \times A$. L'espressione $a_1 \rightarrow_{CR} a_2$ indica che $(a_1, a_2) \in CR$.

In questo caso, la relazione causale considerata è quella in cui c'è una connessione diretta tra le due attività nel modello del processo. In pratica, per affermare che $a_1 \rightarrow_{CR} a_2$, è sufficiente che a_1 sia seguito direttamente da a_2 nel modello, passando attraverso un unico place della rete e senza considerare altre transizioni che potrebbero trovarsi tra a_1 e a_2.
Ad esempio, si consideri un insieme di attività $A = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$. Una possibile CR su questo insieme potrebbe essere:

$$CR = \{(\mathrm{A}, \mathrm{B}), (\mathrm{B}, \mathrm{C}), (\mathrm{A}, \mathrm{D})\}$$

Questo significa che:

- A $\rightarrow_{CR}$ B (l'attività A causa l'attività B),
- B $\rightarrow_{CR}$ C (l'attività B causa l'attività C),
- A $\rightarrow_{CR}$ D (l'attività A causa l'attività D).

Se si considera di nuovo l'esempio sull'approvazione delle spese:

- A: sottomissione della richiesta;
- B: approvazione della richiesta;
- C: pagamento del rimborso;
- D: notifica di approvazione.

La relazione causale CR implica che:

- la sottomissione della richiesta A causa l'approvazione della richiesta B;
- l'approvazione della richiesta B causa il pagamento del rimborso C;
- la sottomissione della richiesta A causa la notifica di approvazione D.

Definizione 5 (Instance Graph)**.** Sia $\sigma = \langle e_1, \ldots, e_n \rangle \in L$ una traccia e sia $\sigma' = \pi_{A_L,\mathbb{J}}(\sigma)$ la sua proiezione sugli insiemi di attività e posizioni di eventi in una traccia. Un Instance Graph (o IG) γ_σ di σ è un grafo aciclico diretto (E, W) dove:

- $E = \{e \in \sigma'\}$ è l'insieme dei nodi, corrispondente agli eventi che si verificano in σ';
- $W = \{(e_h, e_k) \in E \times E \mid h < k \wedge \mathrm{act}(e_h) \rightarrow_{CR} \mathrm{act}(e_k) \wedge (\forall e_q \in E \text{ con } h < q < k \Rightarrow \mathrm{act}(e_h) \nrightarrow_{CR} \mathrm{act}(e_q)) \vee (\forall e_w \in E \text{ con } h < w < k \Rightarrow \mathrm{act}(e_w) \nrightarrow_{CR} \mathrm{act}(e_k))\}$ è l'insieme degli archi, che definiscono un ordinamento parziale su E.

3.2 Builing Instance Graphs

l primo step della metodologia consiste nell'applicazione dell'algoritmo Building Instance Graphs (BIG) [Diamantini et al., 2016], che ha l'obiettivo di costruire delle rappresentazioni a grafo delle tracce a partire dal log e dal modello di processo. In questo modo, per ogni traccia sequenziale presente nel log, verrà creato un Instance Graph (Definizione 5).

La procedura parte dal presupposto che ogni esecuzione del processo inizi e finisca con una sola attività. Questo è necessario per modellare correttamente eventuali parallelismi all'inizio o alla fine dell'esecuzione.
Per garantire ciò, l'approccio propone di aggiungere attività di inizio e fine artificiali tramite una semplice pre-elaborazione dei dati. Tale scelta non impatta sulle caratteristiche dei grafi che ne risultano e può essere generalizzato anche su event log che contengono già attività di inizio e di fine prestabilite. Infatti, le attività fittizie sono posizionate in modo da preservare l'ordine di quelle originali. Questo garantisce che le relazioni temporali tra le attività rimangano accurate e rappresentative del processo reale.
È come se le attività artificiali fungessero semplicemente da segnaposto e, in questo modo, non influenzano le informazioni cruciali riguardanti le attività effettive e le loro relazioni. Così facendo, il comportamento del processo è mantenuto intatto e l'analisi rimane accurata.
Per quanto riguarda il modello di processo, questo può essere fornito da un esperto o ricavato tramite algoritmi di Process Discovery come mostrato in Figura 3.2.

L'algoritmo BIG funziona in due fasi. Innanzitutto viene costruito un IG per ciascuna traccia usando le relazioni causali del modello di processo. Tale operazione, però, può generare IG di bassa qualità se la traccia riportata nel log presenta eventi non conformi al modello. Per esempio, se una traccia contiene attività eseguite in un ordine errato o attività mancanti, l'IG risultante può essere disconnesso e non riflettere correttamente l'ordine temporale degli eventi.
Per risolvere questo problema viene applicata una procedura di riparazione che riconosce le tracce anomale e corregge gli IG aggiungendo (*insertion*) o eliminando (*deletion*) i collegamenti tra i nodi per riflettere correttamente le relazioni causali. La riparazione cerca di rappresentare l'anomalia nel modo più preciso possibile, senza alterare eccessivamente la struttura del grafo originale e mantenendo le relazioni di concorrenza.
Gli IG riparati potrebbero non conformarsi completamente alla definizione originale delle relazioni causali. Ad esempio, una relazione causale inizialmente definita potrebbe essere modificata durante la riparazione. Tuttavia, i grafi riparati sono adattati per rispettare una nuova definizione estesa delle relazioni causali. Questa nuova definizione include tutte le modifiche apportate durante la riparazione, per

riflettere accuratamente le attività aggiunte o modificate e le nuove relazioni causali risultanti.

Ad esempio, si consideri la traccia σ che fa riferimento al modello di processo di Figura 3.2.

$$\begin{aligned}\sigma = \langle &(1, \text{START}),\\ &(2, \text{Permit\_SUBMITTED\_by\_EMPLOYEE}),\\ &(3, \text{Permit\_FINAL-APPROVED\_by\_SUPERVISOR}),\\ &(4, \text{Request\_For\_Payment\_SUBMITTED\_by\_EMPLOYEE}),\\ &(5, \text{Request\_For\_Payment\_FINAL-APPROVED\_by\_SUPERVISOR}),\\ &(6, \text{Request\_For\_Payment\_REJECTED\_by\_MISSING}),\\ &(7, \text{Permit\_REJECTED\_by\_MISSING}),\\ &(8, \text{END})\rangle\end{aligned}$$

Si noti che, come prevede la procedura appena descritta, sono stati aggiunti due eventi fittizi START e END. L'IG relativo alla traccia σ e al processo 3.2 è mostrato in Figura 3.3.

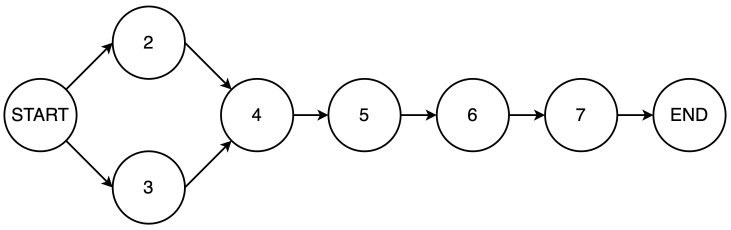


Figura 3.3: Instance Graph relativo alla traccia σ

È interessante osservare come, nel log, la sequenza degli eventi non metta in evidenza alcun parallelismo tra le attività, mentre l'IG corrispondente riesce a rivelare eventuali strutture parallele che possono essere presenti nel modello di processo.

3.3 Feature Engineering

Il secondo step della metodologia (Figura 3.1) ha l'obiettivo di arricchire gli Instance Graph con feature temporali relative ai nodi che saranno utili per la predizione.
Si noti che gli Instance Graph forniscono una rappresentazione esplicita delle relazioni causali tra le attività di un processo. Al contrario, in un event log, le relazioni causali tra eventi possono essere implicite e non immediatamente evidenti in quanto il log è principalmente una sequenza temporale di eventi.
Questo permette di calcolare gli intervalli temporali relativi ai nodi rispetto ai

predecessore secondo le relazioni causali piuttosto che rispetto all'attività precedente nella sequenza della traccia.

3.3.1 Preprocessing dei timestamp

In un event log, spesso ogni evento è associato ad un unico timestamp, che rappresenta il momento in cui l'evento è accaduto. Tuttavia, in alcune analisi o modelli, può essere necessario disporre di due timestamp per ogni evento: uno che indica l'inizio dell'evento (*start time*) e uno che indica la fine dell'evento (*end time*).
Ancora una volta, sfruttando l'ordine relativo degli eventi derivabile a partire dagli IG, è possibile calcolare la durata di ciascun evento in modo abbastanza semplice.
La procedura utilizzata è mostrata di seguito (Algoritmo 1).

Algorithm 1 Preprocessing dei Timestamp di un event log

Input: Event log L con attività fittizie di start s e end e
Output: Event log L' con timestamp corretti per attività sequenziali e parallele

for ogni traccia T in L **do**

Passo 1: Assegna a s un timestamp t_s tale che $t_s = t_{a_1} - 1s$ dove a_1 è la prima attività reale di T

Passo 2: Assegna a e un timestamp t_e tale che $t_e = t_{a_n}$ dove a_n è l'ultima attività reale di T

Passo 5: Scrivi questi timestamp in una colonna *f_timestamp* secondo l'ordine riportato nel log per ottenere i tempi di fine delle attività

Passo 6: Ricopia *f_timestamp* in una nuova colonna *s_timestamp* e fai uno shift verso il basso di una riga per ottenere i tempi di inizio facendo in modo che s ed e siano istantanee

Passo 7:
for ogni attività parallela p in T **do**
1. Calcola il tempo di inizio di p: $t_s(p) = \max\{t_f(a) \mid a$ è un'attività collegata con arco entrante a $p\}$
2. Calcola il tempo di fine di p: $t_f(p) = \min\{t_s(a) \mid a$ è un'attività collegata con arco uscente rispetto a $p\}$

end for

end for

L'algoritmo, per ciascuna traccia nel log, assegna all'attività fittizia di inizio un timestamp con offset di un secondo prima della prima attività registrata per quella traccia e assegna all'attività fittizia di fine lo stesso timestamp dell'ultima attività reale della traccia.

Questa ipotesi è necessaria ma non comporta problemi per il successivo calcolo delle feature in quanto le attività iniziale e finale sono fittizie e servono unicamente per la costruzione degli Instance Graph.
A questo punto, ad ogni evento del log sarà associato un timestamp.
Facendo di nuovo riferimento alla traccia σ e all'Instance Graph di Figura 3.3, questo primo step dell'algoritmo porta a definire un timestamp per ogni evento come mostrato in Figura 3.4.

evento	f_timestamp
START	2024-08-24 11:45:14
2	2024-08-24 11:45:15
3	2024-08-24 11:45:15
4	2024-08-24 14:10:00
5	2024-08-24 15:20:00
6	2024-08-24 16:30:00
7	2024-08-24 16:40:00
END	2024-08-24 16:40:00

Figura 3.4: Definizione dei timestamp di fine delle attività

Quando un log di eventi contiene un solo timestamp per ciascuna attività, il timestamp rappresenta il momento in cui l'attività è stata registrata o osservata. Questo timestamp, in effetti, rappresenta un punto specifico nel tempo, ma non indica direttamente se si tratta del momento di inizio o di fine dell'attività.
Occorre però osservare che, sia che si consideri un timestamp come tempo di inizio o di fine, la sequenza degli eventi nel log si conserva e l'ordine relativo tra le attività di una stessa traccia o tra tracce differenti rimane invariato.
Per tale ragione, si suppone che i timestamp riportati nel log corrispondono alla fine delle attività.
Questo è utile per determinare quando una traccia è completa e per calcolare la durata complessiva della traccia stessa.
Inoltre, conoscendo il momento di inizio della prima attività nella traccia, questa assunzione permette di calcolare l'intervallo di tempo di ciascun evento sottraendo il timestamp di fine da quello di inizio.
Si procede quindi con il calcolo dei timestamp di inizio tramite la procedura di *shift*.
Questo passaggio è mostrato in Figura 3.5.

evento	f_timestamp	s_timestamp
START	2024-08-24 11:45:14	
2	2024-08-24 11:45:15	2024-08-24 11:45:14
3	2024-08-24 11:45:15	2024-08-24 11:45:15
4	2024-08-24 14:10:00	2024-08-24 11:45:15
5	2024-08-24 15:20:00	2024-08-24 14:10:00
6	2024-08-24 16:30:00	2024-08-24 15:20:00
7	2024-08-24 16:40:00	2024-08-24 16:30:00
END	2024-08-24 16:40:00	2024-08-24 16:40:00
		2024-08-24 16:40:00

⟹

evento	f_timestamp	s_timestamp
START	2024-08-24 11:45:14	2024-08-24 11:45:14
2	2024-08-24 11:45:15	2024-08-24 11:45:14
3	2024-08-24 11:45:15	2024-08-24 11:45:15
4	2024-08-24 14:10:00	2024-08-24 11:45:15
5	2024-08-24 15:20:00	2024-08-24 14:10:00
6	2024-08-24 16:30:00	2024-08-24 15:20:00
7	2024-08-24 16:40:00	2024-08-24 16:30:00
END	2024-08-24 16:40:00	2024-08-24 16:40:00

Figura 3.5: Calcolo dei timestamp di inizio delle attività

Viene creata una nuova colonna chiamata s_timestamp, che è una copia della colonna f_timestamp, spostata verso il basso di una riga. Successivamente, l'ultima voce di s_timestamp viene eliminata e il timestamp di fine dell'attività iniziale fittizia viene inserito nella prima posizione della colonna. Ovvero, la convenzione che si adotta secondo questo approccio è quella di rendere le attività fittizie istantanee facendo in modo che i timestamp di inizio e fine coincidano rispettivamente.

Per gestire le attività parallele, occorre di nuovo modificare i timestamp in quanto gli step precedenti modellano correttamente solo le sequenze. L'algoritmo calcola il tempo di inizio di ciascuna attività parallela come il massimo dei tempi di fine delle attività che hanno un arco entrante verso di essa. Questo assicura che l'attività parallela inizi solo dopo che tutte le sue attività precedenti sono terminate. Il tempo di fine di un'attività parallela viene invece determinato come il minimo dei tempi di inizio delle attività che hanno un arco uscente da essa, garantendo che l'attività parallela finisca prima che tutte le sue attività successive possano iniziare.
Questo secondo passaggio è mostrato in Figura 3.6.

evento	f_timestamp	s_timestamp
START	2024-08-24 11:45:14	2024-08-24 11:45:14
2	2024-08-24 11:45:15	2024-08-24 11:45:14
3	2024-08-24 11:45:15	2024-08-24 11:45:15
4	2024-08-24 14:10:00	2024-08-24 11:45:15
5	2024-08-24 15:20:00	2024-08-24 14:10:00
6	2024-08-24 16:30:00	2024-08-24 15:20:00
7	2024-08-24 16:40:00	2024-08-24 16:30:00
END	2024-08-24 16:40:00	2024-08-24 16:40:00

⇒

evento	f_timestamp	s_timestamp
START	2024-08-24 11:45:14	2024-08-24 11:45:14
2	2024-08-24 11:45:15	2024-08-24 11:45:14
3	2024-08-24 11:45:15	2024-08-24 11:45:14
4	2024-08-24 14:10:00	2024-08-24 11:45:15
5	2024-08-24 15:20:00	2024-08-24 14:10:00
6	2024-08-24 16:30:00	2024-08-24 15:20:00
7	2024-08-24 16:40:00	2024-08-24 16:30:00
END	2024-08-24 16:40:00	2024-08-24 16:40:00

Figura 3.6: Modifica dei timestamp per le attività parallele

L'algoritmo applica questi passaggi a tutte le tracce nel log, assicurandosi che i timestamp riflettano correttamente l'ordine temporale delle attività sia sequenziali che parallele.

3.3.2 Calcolo delle feature

Questo step ha l'obiettivo di arricchire i nodi degli Instance Graph con un set di feature utili per la previsione.
In generale, le feature che è possibile considerare sono di due tipologie:

- feature dirette, cioè, attributi memorizzati nel log degli eventi,
- feature indirette che sono calcolabili a partire dalle informazioni disponibili nelle tracce.

In questa sezione, verrà mostrato il calcolo di tre feature temporali utilizzando la rappresentazione fornita dagli Instance Graph. [Chiorrini et al., 2023]

Gli Instance Graphs modellano esplicitamente le relazioni causali tra le attività e questo consente di calcolare le caratteristiche temporali offrendo una visione più chiara e precisa delle dinamiche temporali del processo. Al contrario, il log rappresenta le tracce come una sequenza senza mettere in evidenza la concorrenza tra le attività. Inoltre, la normalizzazione dei tempi tra gli eventi di un log può essere complicata, poiché si basa su sequenze di eventi che potrebbero non riflettere correttamente la struttura causale del processo. Invece, con gli IG è più facile normalizzare le caratteristiche temporali perché si ha una rappresentazione strutturata e chiara delle relazioni. Questo porta a modelli più robusti e precisi, poiché le previsioni si basano su informazioni più complete e rilevanti. [Chiorrini et al., 2023]

Tempo iniziale normalizzato ($t_{d_{n_i}}$)

La feature $t_{d_{n_i}}$ rappresenta la durata normalizzata dall'inizio del processo fino a un dato evento e_i.
La formula utilizzata è la seguente:

$$t_{d_{n_i}}(e_i) = \frac{t(e_i) - t_0}{\Delta max_t} \tag{3.1}$$

dove $t(e_i)$ è il timestamp di inizio dell'evento e_i, t_0 è il timestamp dell'inizio del processo (ossia del primo evento), e Δmax_t rappresenta la durata massima del processo. Questa durata viene calcolata come la differenza tra il timestamp dell'ultimo evento e_n e il timestamp del primo evento e_1 della traccia tramite la seguente formula:

$$\Delta max_t = t(e_n) - t(e_1) \tag{3.2}$$

dove:

- $t(e_n)$ è il timestamp finale dell'ultimo evento e_n nella traccia.
- $t(e_1)$ è il timestamp iniziale del primo evento e_1 nella traccia.

La differenza temporale $t(e_n) - t(e_1)$ rappresenta la durata complessiva del processo. Questo valore viene utilizzato anche per normalizzare le altre feature temporali al fine di consentire il confronto su una scala comune tra eventi che appartengono a tracce con durate diverse.
In generale, le diverse istanze di un processo (tracce) possono avere durate significativamente diverse. Senza normalizzazione, le differenze di scala temporale renderebbero difficile confrontare direttamente gli eventi o le durate tra diverse tracce.

Indicatore settimanale normalizzato ($t_{w_{n_i}}$)

La feature $t_{w_{n_i}}$ misura il tempo trascorso dall'ultima mezzanotte di domenica fino all'evento corrente, normalizzato rispetto alla durata totale di una settimana.

$$t_{w_{n_i}}(e_i) = \frac{t(e_i) - t_{w_0}}{\Delta t_w} \tag{3.3}$$

dove $t(e_i)$ è il timestamp di inizio dell'evento e_i, t_{w_0} rappresenta il timestamp della mezzanotte dell'ultima domenica prima dell'evento, e Δt_w è il numero totale di secondi in una settimana (604.800 secondi).

In generale, in molti processi aziendali e operativi, esistono pattern specifici che si ripetono settimanalmente. Ad esempio, alcune attività potrebbero essere più frequenti durante i giorni feriali rispetto ai fine settimana.
Per tale ragione, questa feature consente di identificare e analizzare questi pattern, distinguendo eventi che accadono in giorni e orari specifici della settimana.
Utilizzando $t_{w_{n_i}}$ per la predizione, è possibile confrontare eventi che si verificano in diversi momenti della settimana, indipendentemente dalla settimana specifica in cui si trovano. Questo è utile per analizzare la distribuzione temporale degli eventi su base settimanale e identificare tendenze o anomalie nel comportamento del processo. Inoltre, per le aziende che operano con cicli settimanali, conoscere esattamente quando un evento tende a verificarsi durante la settimana può aiutare a ottimizzare l'allocazione delle risorse e a migliorare la pianificazione operativa.

Differenza temporale normalizzata (Δt_{n_i})

La feature Δt_{n_i} rappresenta la differenza temporale tra un evento e_i e il suo predecessore causale immediato e_j ed normalizzata rispetto alla durata massima del processo.

$$\Delta t_{n_i}(e_i, e_j) = \frac{t(e_i) - t(e_j)}{\Delta max_t} \tag{3.4}$$

dove $t(e_i)$ è il timestamp di inizio dell'evento corrente e_i, $t(e_j)$ è il timestamp di fine del predecessore causale immediato e_j, e Δmax_t rappresenta la durata massima del processo.

In termini semplici, Δt_{n_i} cattura quanto tempo è passato tra due eventi consecutivi all'interno dello stesso processo. Questa caratteristica risulta essenziale per comprendere la dinamica temporale all'interno di un processo in quanto permette di capire quanto rapidamente o lentamente avvengono le transizioni tra le attività. Questa informazione è cruciale per l'ottimizzazione del processo, poiché può rivelare colli di bottiglia o inefficienze.
In questo modo, è possibile identificare pattern ricorrenti o anomalie nelle tempisti-

che tra le attività. Ad esempio, se un certo tipo di transizione tra eventi tende ad avvenire sempre entro un breve intervallo di tempo, mentre altre richiedono molto più tempo, Δt_{n_i} può evidenziarlo chiaramente. Questo aiuta nell'analisi predittiva e nel miglioramento continuo del processo.

3.4 Generazione dei Prefissi

Questo step descrive un processo per costruire un dataset di prefissi a partire da un insieme di Instance Graphs.

Definizione 6 (Prefix Instance Graph)**.** Sia (E, W) l'Instance Graph di una traccia σ. Sia E_k l'insieme degli eventi del prefisso della traccia $p_k(\sigma)$ di dimensione k. Il prefisso di un Instance Graph di dimensione k di σ è definito come il grafo $p_k((E, W)) = (\tilde{E}_k, W \cap (\tilde{E}_k \times \tilde{E}_k))$. Informalmente, un prefisso $p_k(g_j)$ è un sottografo di g_j che coinvolge solo k nodi di g_j, ovvero i nodi inclusi nel prefisso della traccia corrispondente.

Gli Instance Graphs sono strutture che rappresentano l'ordine degli eventi in una traccia, con i nodi che corrispondono agli eventi e gli archi che indicano le relazioni tra questi eventi.
Pertanto, è possibile collegare ogni nodo in un IG ad un indice progressivo che rappresenta la posizione dell'evento corrispondente nella traccia. Questo indice determina l'ordine dei nodi e può essere utilizzato per costruire progressivamente il set di prefissi-IG (Definizione 6).
In particolare, dato un IG g, il prefisso del grafo $p_2(g)$ si ottiene selezionando i primi due nodi e gli archi tra di essi. Questo prefisso è etichettato con l'attività dell'evento in posizione 3. Il prefisso successivo viene poi derivato estendendo $p_2(g)$ con il nodo di indice 3 e con i rispettivi archi che lo collegano a $p_2(g)$. L'etichetta associata è l'attività dell'evento in posizione 4. La procedura viene ripetuta fino a quando l'attività corrispondente all'ultimo nodo del grafo è selezionata come etichetta.
Ad esempio, si consideri ancora una volta la traccia σ e il corrispondente IG riportato in Figura 3.3. I prefissi $p_1(g)$, $p_2(g)$ e $p_3(g)$ corrispondenti sono mostrati in Figura 3.7.

Il concetto di prefisso potrebbe anche essere leggermente modificato rispetto alla versione tradizionale [Chiorrini et al., 2023] per tenere conto dei parallelismi presenti nell'Instance Graph.
Nella definizione classica, i prefissi vengono generati seguendo un semplice ordine progressivo degli eventi nella traccia, senza considerare la possibile esistenza di eventi che avvengono in parallelo. Tuttavia, utilizzando l'IG, è possibile catturare anche queste situazioni.

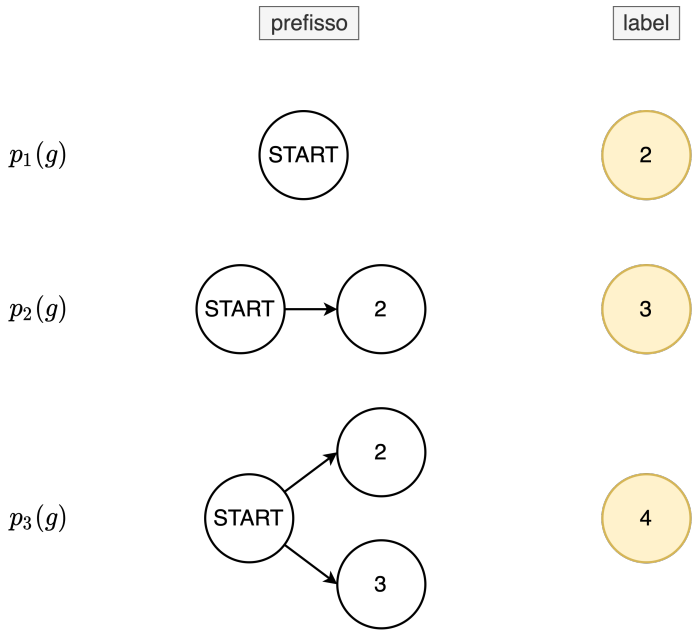


Figura 3.7: Prefissi di lunghezza 2 e 3 relativi all'Instance Graph di Figura 3.3

Secondo la nuova definizione, quando un prefisso termina con un nodo che ha due o più nodi collegati in parallelo, la label associata a quel prefisso non è più un singolo evento. Invece, la label viene estesa per includere tutti i nodi paralleli collegati. Questo approccio permette di preservare la complessità e le dipendenze dei processi rappresentati, riflettendo in maniera più accurata le dinamiche del sistema.
Ad esempio, supponendo di avere un prefisso che termina con un nodo n, se da n partano due archi verso due nodi paralleli n_1 e n_2, nella definizione modificata, la label considerata per questo prefisso includerà entrambi i nodi n_1 e n_2. Questo riflette il fatto che, a partire da n, il sistema può progredire in parallelo con n_1 e n_2, anziché avanzare in modo lineare.
Questa modifica rende il concetto di prefisso più flessibile e aderente alla realtà dei processi con attività parallele, migliorando la capacità del modello di rappresentare processi complessi con maggiore fedeltà.
Ad esempio, secondo questa nuova accezione, il prefisso di lunghezza 1 relativo all'IG di Figura 3.3 è riportato in Figura 3.8.

Pur adottando la definizione estesa di prefisso per includere i parallelismi dell'Instance Graph, emerge un problema logico nel trattare i nodi paralleli come se fossero in sequenza. Pertanto, potrebbe essere possibile adottare ancora una nuova versione della definizione di prefisso che risolve questa incongruenza, garantendo una rappresentazione coerente dei parallelismi all'interno del grafo.
Formalmente, dato un prefisso $p_k(g_j)$ che termina con un nodo v_k collegato a due o più nodi paralleli $v_{k+1}, v_{k+2}, \ldots$, la label di v_k è l'insieme $v_{k+1}, v_{k+2}, \ldots$. Se poi

Figura 3.8: Costruzione del prefisso di lunghezza 1 tramite la versione modificata di prefisso

si considera l'estensione del prefisso $p_{k+1}(g_j)$ includendo un nodo parallelo, la label successiva dovrebbe riflettere la prosecuzione logica del processo, che esclude gli altri nodi paralleli. Dunque, sarebbe più corretto attribuire la label al prossimo nodo sequenziale.

Questa nuova versione della definizione di prefisso risolve un problema logico presente nella versione precedente. Quando un nodo v_1 è collegato a due nodi paralleli v_2 e v_3, la label di v_1 non può essere semplicemente v_2 o v_3 individualmente, poiché questi due nodi rappresentano eventi paralleli e quindi dovrebbero essere considerati insieme.
Un prefisso, però, deve rispettare la struttura del grafo, in particolare la presenza di parallelismi. Tuttavia, estendere il prefisso per includere uno di questi nodi paralleli, ad esempio v_2, secondo la definizione originale, porta ad una situazione in cui la label del prefisso costituito dai nodi v_1 e v_2 è ancora v_3 e ciò implica una sequenzialità che non esiste.
Invece, la label del prefisso v_1, v_2 dovrebbe essere il prossimo nodo non parallelo (ad esempio v_4), che rappresenta la prosecuzione logica del processo dopo che uno dei nodi paralleli è stato risolto. Questo approccio evita il controsenso logico di trattare eventi paralleli come sequenziali e garantisce una rappresentazione più accurata e coerente delle relazioni tra eventi in un processo.
Ad esempio, il prefisso di lunghezza 2 generato secondo questa nuova definizione è riportato in Figura 3.9.

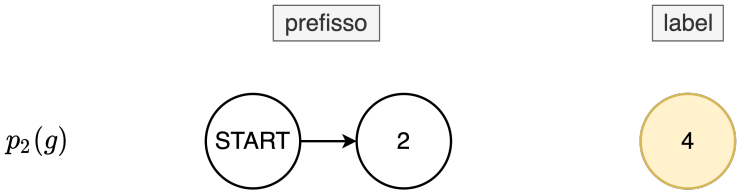


Figura 3.9: Costruzione del prefisso di lunghezza 2 tenendo in conto dei parallelismi

Nonostante le modifiche proposte permettano una rappresentazione più accurata delle tracce con parallelismi, è importante sottolineare che, per eseguire un confronto equo

con i risultati ottenuti utilizzando la definizione di prefisso presente in letteratura (Definizione 6), è necessario attenersi alla formulazione classica. Quest'ultima non tiene conto dei parallelismi presenti negli Instance Graph e considera esclusivamente l'ordine sequenziale degli eventi nella traccia. Nonostante ciò, adottare la definizione originale consente di mantenere coerenza metodologica con gli studi precedenti, evitando che le discrepanze derivanti dalla diversa gestione dei nodi paralleli influenzino negativamente l'interpretazione e la comparabilità dei risultati sperimentali.

3.5 Generazione dei Prefissi Attivi

Quando si considera un prefisso specifico e il relativo intervallo di tempo in cui esso si sviluppa, è utile identificare quali altri prefissi eseguono simultaneamente in altre tracce. Questo è particolarmente importante perché i prefissi paralleli che si verificano in contemporanea possono influenzare significativamente il comportamento complessivo del sistema. Questo tipo di analisi aiuta a rilevare e comprendere le sovrapposizioni e le interazioni tra diverse tracce, fornendo una visione più integrata delle dinamiche del sistema.
In pratica, calcolare quali altri prefissi si sviluppano in parallelo in altre tracce permette di evidenziare come i processi o le attività si intersecano e si influenzano reciprocamente. Questa conoscenza è cruciale per ottimizzare il coordinamento tra le attività, migliorare la pianificazione e gestire le risorse in modo più efficiente. Inoltre, questa prospettiva offre una base solida per analisi più sofisticate, come la previsione di conflitti o la sincronizzazione di eventi tra tracce diverse.

Identificare i prefissi che si sviluppano simultaneamente in altre tracce è particolarmente utile per la predizione e la gestione del carico del sistema, poiché fornisce una visione approfondita del carico totale che il sistema sta affrontando in un dato momento e aiuta a pianificare di conseguenza.
Conoscere i prefissi paralleli in esecuzione permette di anticipare e prepararsi per i picchi di attività. Se diversi prefissi si verificano contemporaneamente, il sistema deve gestire una quantità maggiore di risorse e processi simultaneamente. Questa informazione è cruciale per stimare adeguatamente la capacità del sistema e garantire che sia in grado di affrontare il carico senza compromettere le prestazioni.
Inoltre, la consapevolezza dei prefissi paralleli facilita una gestione più efficiente delle risorse. Se è noto che determinati prefissi richiedono risorse intensivamente e si sviluppano in parallelo, è possibile allocare le risorse in modo più strategico per evitare colli di bottiglia e sovraccarichi. Questo approccio contribuisce a mantenere le prestazioni del sistema e a prevenire rallentamenti o interruzioni.

Per tale ragione, è stata pensata una procedura di calcolo dei prefissi paralleli ad un dato prefisso di riferimento.

L'idea generale per questo calcolo è riportata in Figura 3.10.

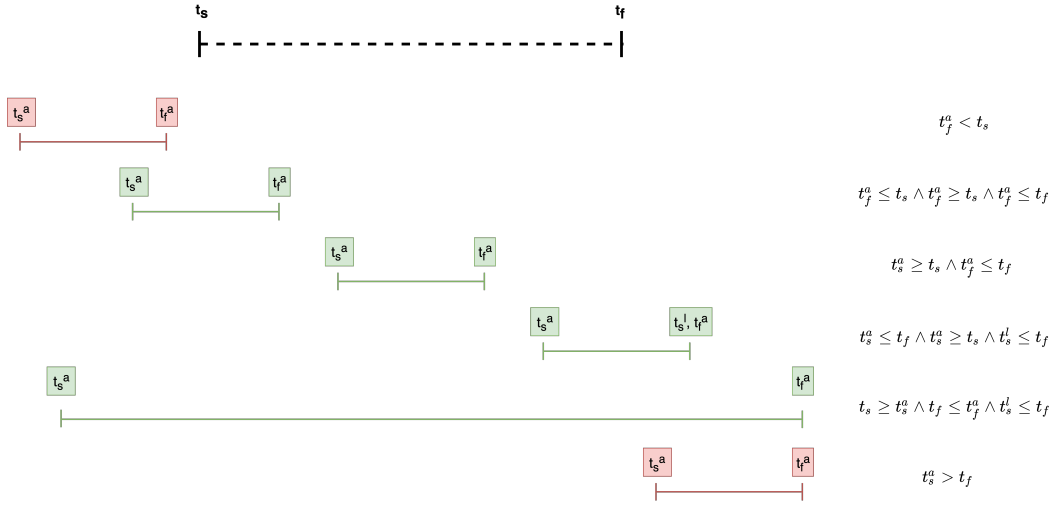


Figura 3.10: Calcolo dei prefissi paralleli

In particolare, t_s e t_f sono rispettivamente il tempo di inizio e di fine del prefisso rispetto a cui si vogliono calcolare i prefissi paralleli.
Un prefisso è considerato parallelo ad un altro (linee verdi) quando le loro esecuzioni si sovrappongono temporalmente e appartengono a tracce diverse.
Questo significa che il prefisso p_a esegue in parallelo ad un prefisso p se l'intervallo temporale in cui esegue p_a ha un'intersezione non nulla con l'intervallo in cui esegue p.
Se questa condizione non è rispettata (linee rosse), p_a non può essere considerato parallelo a p.
Questo avviene, ad esempio, quando i prefissi appartengono alla stessa traccia, rendendo impossibile la loro esecuzione simultanea senza conflitti, oppure quando non c'è una sovrapposizione temporale per cui i tempi di inizio e fine dei prefissi non interferiscono l'uno con l'altro.

Definizione 7 (Prefissi attivi rispetto ad un prefisso p di riferimento)**.** Un prefisso p_a che inizia al tempo t_s^a, che finisce al tempo t_f^a e la cui ultima attività inizia al tempo t_s^l è considerato parallelo a un prefisso p che esegue nell'intervallo temporale $[t_s, t_f]$ se e solo se sono soddisfatte tutte le seguenti condizioni:

- $t_s^a \leq t_f$, ovvero, il tempo di inizio del prefisso p_a è minore o uguale al tempo di fine del prefisso p,

- $t_f^a \geq t_s$, ovvero, il tempo di fine del prefisso p_a è maggiore o uguale al tempo di inizio del prefisso p,

- $t_s^l \leq t_f$, ovvero, l'ultima attività del prefisso p_a è iniziata prima della fine del prefisso p.
- Il prefisso p_a appartiene ad una traccia diversa rispetto a p.

La Definizione 7 individua tutti i prefissi paralleli ad un dato prefisso p.
Quando si parla di prefissi paralleli, è cruciale considerare la definizione basata sugli intervalli temporali. In questo contesto, se due prefissi di riferimento hanno lo stesso intervallo temporale, tutti i prefissi che eseguono in parallelo con uno di essi dovranno eseguire in parallelo anche con l'altro.
In altre parole, a prefissi di riferimento che hanno la stessa estensione temporale corrispondono stessi prefissi attivi.

Nel dataset, i prefissi sono rappresentati come grafi, ognuno dei quali differisce per almeno un nodo. Quando si considera un Instance Graph che rappresenta una traccia con n eventi, si ottengono $n - 1$ prefissi che appartengono allo stesso case, ovvero, alla stessa esecuzione.
Ora, se si considera un prefisso di riferimento e i relativi prefissi attivi, può accadere che alcuni di questi condividano lo stesso case. Ciò avviene perché, i prefissi attivi possono derivare dallo stesso IG e condividere una sequenza di eventi simile o identica, differendo solo per il numero di nodi. Quindi, alcuni prefissi attivi non si distinguono necessariamente nella loro sequenza di eventi, poiché provengono dalla stessa traccia iniziale.
In Figura 3.11 è mostrata una sequenza di prefissi derivati da una traccia σ. Ciascun $p_i(\sigma)$ indica una progressione crescente di nodi e archi, rappresentando eventi aggiuntivi nella traccia.
L'esempio dimostra che, sebbene i prefissi attivi siano eseguiti in parallelo, è possibile che più di uno di essi faccia riferimento alla medesima traccia originaria. Questo significa che i prefissi attivi possono condividere una parte della struttura della traccia da cui derivano, differenziandosi per il numero di eventi o per la modalità di esecuzione parallela.
Tuttavia, se si vogliono considerare solo i prefissi che hanno un impatto significativo o che sono di particolare rilevanza, è utile focalizzarsi solo sul prefisso parallelo più lungo appartenente ad una data traccia. Quest'ultimo rappresenta la porzione più estesa di attività che si sovrappone temporalmente con il prefisso di riferimento.
Se si considerano più prefissi paralleli per una stessa traccia, questi potrebbero includere attività che si sovrappongono o si ripetono in modo non intenzionale. Ogni prefisso di una stessa esecuzione, infatti, rappresenta attvità che sono frammenti della stessa traccia.
Per questo, includere più prefissi che fanno riferimento ad uno stesso case può comportare una ripetizione delle stesse attività o eventi.
Questo non solo complica l'analisi, ma può anche portare a una distorsione dei risultati. Ad esempio, se una traccia contiene più prefissi che coprono intervalli

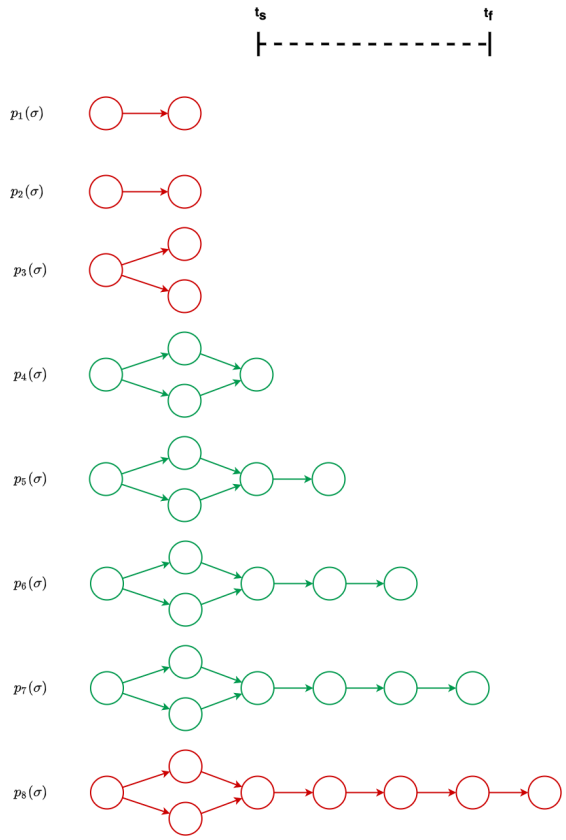


Figura 3.11: Esempio di prefissi attivi che condividono la stessa traccia

sovrapposti, questo potrebbe portare ad interpretare erroneamente alcune attività come più frequenti o più complesse di quanto lo siano effettivamente.
Di conseguenza, per ottenere un'analisi chiara e utile, è preferibile concentrarsi su un solo prefisso parallelo per ogni traccia, precisamente quello più lungo.
Ad esempio, facendo di nuovo riferimento alla Figura 3.11, il prefisso parallelo più lungo corrispondente alla traccia σ è $p_7(\sigma)$. Questo prefisso rappresenta la porzione più estesa di tempo in cui le attività sono parallele al prefisso di riferimento senza ripetizioni. In questo modo, si evita di includere attività ripetute e si ottiene una visione più precisa e completa dell'interazione temporale.

Definizione 8 (Prefissi attivi massimi)**.** Sia P l'insieme di tutti i prefissi p, con ogni prefisso p associato a un identificatore di case case_p. Si supponga che ogni prefisso p abbia un intervallo temporale definito da t_s (tempo di inizio) e t_f (tempo di fine). I prefissi massimi $P_{max} \subseteq P_{par} \subseteq P$ e paralleli a $p \in P$ secondo la Definizione 3.10 sono tali che:

$$p_{\max} = \arg \max_{p_i \in P, \text{case}_p = \text{case}} |\text{nodes}_{p_i}|, \forall p_{max} \in P_{max}$$

Dove arg max identifica il prefisso p_i di massimizza lunghezza per ogni case. La

formula completa per ottenere l'insieme dei prefissi attivi massimi è:

$$P_{\max} = \left\{ p_i \in P \mid \text{case}_{p_i} = \text{case} \wedge |\text{nodes}_{p_i}| = \max_{p' \in P_{\text{case}}} |\text{nodes}_{p'}| \right\}$$

Dove P_{case} è l'insieme di tutti i prefissi associati a un dato case e che sono paralleli a p e la funzione $\max_{p' \in P_{\text{case}}} |\text{nodes}_{p'}|$ restituisce la lunghezza massima tra tutti i prefissi del case specifico.

La Definizione 8 garantisce che per ogni identificatore di case verrà selezionato un solo prefisso parallelo. Questo prefisso è il più lungo tra tutti i prefissi paralleli per quel case specifico. Questo implica che non ci siano ambiguità o duplicazioni nel risultato, poiché solo il prefisso con la durata massima viene selezionato.

Si noti inoltre che un prefisso può contenere più attività parallele, ovvero, che iniziano e finiscono nello stesso intervallo temporale. Per questo, se la definizione di lunghezza di un prefisso si basasse solo sulla durata temporale, ci potrebbero essere dei casi in cui più prefissi relativi allo stesso case sarebbero in parallelo ad un dato prefisso di riferimento. Questo perché due prefissi con la stessa durata temporale potrebbero contenere un numero diverso di attività.
Ad esempio, i prefissi $p_2(\sigma)$ e $p_3(\sigma)$ mostrati in Figura 3.11 hanno la stessa durata temporale, ma, di questi, solo $p_3(\sigma)$ dovrebbe essere considerato come prefisso attivo massimo in quanto contiene più attività. Allo stesso modo, se due prefissi hanno la stessa estensione temporale, ma uno contiene più attività istantanee, il prefisso con il maggior numero di attività viene sempre scelto a scapito dell'altro in quanto rappresenta una sequenza più ricca e dettagliata di eventi.
In definitiva, sfruttando la definizione di prefisso (Definizione 6) e considerando l'ordinamento tra i nodi definito tramite gli Instance Graph è possibile calcolare la lunghezza semplicemente contando il numero di nodi. Questo approccio garantisce che il prefisso selezionato non solo abbia una durata temporale massima, ma anche un numero massimo di attività.
Si noti anche che, selezionando il prefisso con il maggior numero di nodi, i prefissi che terminano con un maggior numero di attività parallele o che gestiscono un numero maggiore di attività istantanee verranno sempre selezionati a scapito degli altri. Ovvero, i prefissi con meno nodi, o quelli che terminano con attività istantanee o parallele, verranno sempre esclusi dalla selezione.
Definire la lunghezza di un prefisso in base al numero di nodi consente di focalizzarsi sui prefissi che meglio rappresentano la complessità del processo. Questo approccio garantisce che solo i prefissi più informativi e complessi vengano selezionati per la rappresentazione finale.

Infine, è importante notare che il fatto che un prefisso p_i sia parallelo ad un prefisso p_j non implica necessariamente che p_j sia parallelo a p_i. Questa asimmetria nella

relazione di parallelismo tra prefissi può essere spiegata da diversi fattori che riguardano l'ordine temporale e la distribuzione delle attività nelle diverse tracce.
Un motivo per cui p_i può essere parallelo a p_j, ma non viceversa, è che le condizioni temporali che definiscono il parallelismo possono essere soddisfatte in una direzione, ma non nell'altra. Ad esempio, si supponga che p_i inizi prima o contemporaneamente a p_j, ma finisca anche prima a p_j. In questo caso, p_i può essere considerato parallelo a p_j perché c'è un'intersezione temporale tra l'intervallo di esecuzione di p_i e quello di p_j. Tuttavia, p_j non è parallelo a p_i se la sua ultima attività comincia oltre la fine di p_i. In questo caso, il parallelismo intercorrerebbe tra p_i e possibilmente un'altro prefisso diverso da p_j della traccia a cui p_j appartiene.
In generale, l'ordine di esecuzione delle attività gioca un ruolo cruciale. Infatti, se p_i si sovrappone solo parzialmente a p_j, è possibile che p_i copra un intervallo temporale ridotto rispetto a p_j. In tal caso, p_i potrebbe avere meno nodi attivi o meno sovrapposizioni con altre attività rispetto a p_j. Questo riduce la probabilità che p_j venga considerato come parallelo a p_i, anche se il contrario potrebbe essere vero.

Ad esempio, si considerino i prefissi $p_8(\sigma)$ e $p_{12}(\sigma')$ relativi alle tracce σ e σ' mostrati in Figura 3.12.

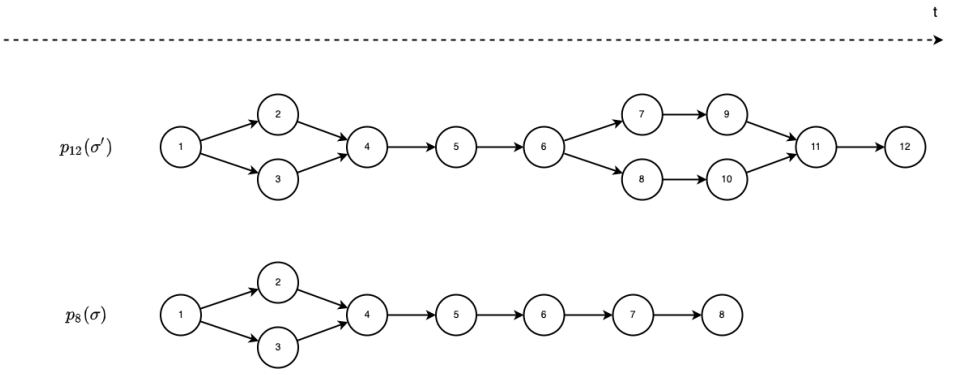


Figura 3.12: Monodirezionalità della definizione di parallelismo

L'immagine mostra i due prefissi disposti lungo una linea temporale, rappresentata dalla freccia tratteggiata in alto.
Si supponga che il prefisso $p_8(\sigma)$ rappresenti l'ultimo prefisso disponibile per la traccia σ, cioè, che non esistono prefissi successivi che includano un numero maggiore di attività. Di conseguenza, $p_8(\sigma)$ copre l'intera esecuzione della traccia σ fino al nodo 8. Invece, il prefisso $p_{12}(\sigma')$ si estende oltre la durata temporale di $p_8(\sigma)$, includendo attività fino al nodo 12 nella traccia σ'.
Nell'interpretare il parallelismo, è possibile affermare che $p_8(\sigma)$ è un prefisso attivo massimo rispetto a $p_{12}(\sigma')$, poiché le attività di $p_8(\sigma)$ si sovrappongono temporalmente con quelle di $p_{12}(\sigma')$ come descritto nella Definizione 8.
Tuttavia, il contrario non è vero: $p_{12}(\sigma')$ non è un prefisso attivo massimo rispetto a $p_8(\sigma)$, perché contiene attività aggiuntive, relative al nodo 11 e al nodo 12, che non

si sovrappongono a quelle di $p_8(\sigma)$.
Dunque, mentre $p_8(\sigma)$ può essere considerato parallelo a $p_{12}(\sigma')$, il contrario non è valido a causa delle attività aggiuntive di $p_{12}(\sigma')$ che non si sovrappongono a quelle di $p_8(\sigma)$.

Il concetto di parallelismo simmetrico tra prefissi si riferisce, invece, alla condizione in cui due prefissi si sovrappongono temporalmente in modo reciproco.
In altre parole, ciascun prefisso include attività che si svolgono all'interno dello stesso intervallo di tempo in cui si verificano le attività dell'altro prefisso, garantendo che entrambi si sviluppino nello stesso intervallo temporale di sovrapposizione. La simmetria del parallelismo non implica che i prefissi abbiano esattamente lo stesso numero di attività o che finiscano nello stesso istante, ma che le attività di un prefisso si svolgano interamente durante l'intervallo temporale di esecuzione dell'altro, senza introdurre interruzioni nella sovrapposizione temporale.
Nel caso in esame, per comprendere meglio il parallelismo simmetrico, è utile prendere in considerazione il prefisso $p_8(\sigma)$ della traccia σ e il prefisso $p_{10}(\sigma')$ della traccia σ'. In questo caso, il prefisso $p_8(\sigma)$ è parallelo a $p_{10}(\sigma')$, e viceversa, poiché entrambi i prefissi includono attività che si sovrappongono sia all'intervallo temporale in cui si estende $p_{10}(\sigma')$ che all'intervallo temporale in cui si estende $p_8(\sigma)$.

Considerando il prefisso $p_9(\sigma')$ come riferimento, $p_8(\sigma)$ sarebbe comunque parallelo a $p_9(\sigma')$ poiché entrambe le sequenze di attività si sovrappongono. Tuttavia, $p_9(\sigma')$ non rappresenterebbe un prefisso parallelo massimo rispetto a $p_8(\sigma)$. Perché un prefisso sia parallelo massimo, deve estendersi il più possibile senza rompere la sovrapposizione temporale, cosa che accade nel caso di $p_{10}(\sigma')$, che si estende fino al nodo 10, rimanendo ancora sovrapposto a $p_8(\sigma)$ ma includendo ulteriori attività.
In questo scenario, quindi, $p_{10}(\sigma')$ sarebbe il prefisso parallelo massimo rispetto a $p_8(\sigma)$, mentre $p_9(\sigma')$ no.
Così facendo, il prefisso $p_{10}(\sigma')$ include una gestione di attività parallele (nodi 9 e 10), rappresentando una maggiore complessità rispetto a $p_9(\sigma')$.

Infine, l'analisi del parallelismo tra prefissi deve tenere conto della possibilità di prefissi con durate temporali molto diverse. Se un prefisso p_j ha una durata molto lunga rispetto a p_i, è più probabile che p_j si sovrapponga a molti altri prefissi, incluso p_i. Tuttavia, p_i potrebbe coprire solo una piccola porzione del tempo in cui p_i è attivo, rendendo meno probabile che il parallelismo sia bidirezionale. La differenza nelle durate può creare situazioni in cui uno dei prefissi è significativamente più lungo o più breve dell'altro, alterando la percezione del parallelismo.

In sintesi, l'analisi dei prefissi paralleli attivi offre un modo potente e dettagliato per comprendere i processi. Tuttavia, il costo computazionale associato a questa analisi può essere significativo, richiedendo strategie e strumenti avanzati per gestire la

complessità e garantire che l'analisi sia realizzabile entro tempi e risorse ragionevoli. Nonostante le criticità esposte, il presente lavoro ha lo scopo di valutare se un'analisi di questo tipo sia giustificata da una comprensione più accurata del processo.

3.6 Predizione tramite DGCNN

L'ultimo step della metodologia consiste nella classificazione dei grafi attraverso l'architettura Deep Graph Convolutional Neural Network (DGCNN) che verrà approfondita nel capitolo succesivo.
In questa fase, i grafi generati e trasformati vengono utilizzati come input per il modello, con l'obiettivo di assegnarli ad una delle classi predefinite.
All'interno dell'architettura DGCNN sono state incluse le tre feature temporali associate ai nodi, selezionate per catturare l'evoluzione dei nodi nel tempo e per migliorare la capacità del modello di identificare pattern dinamici nei grafi.
In aggiunta, queste rappresentazioni sono integrate con i prefissi attivi che rappresentano l'evoluzione del processo e che potrebbero fornire informazioni cruciali sulla dinamica complessiva del grafo stesso.
Lo scopo è quello di comprendere se l'inclusione dei prefissi attivi possa migliorare la precisione e l'affidabilità della classificazione all'interno delle classi predefinite.

Capitolo 4

Deep Graph Convolutional Neural Network

L'apprendimento automatico su dati strutturati come grafi sta diventando sempre più rilevante in diversi campi applicativi, tra cui la bioinformatica, la chimica e l'analisi delle reti sociali [Zhang et al., 2018].

Le reti neurali convoluzionali (CNN), originariamente progettate per la classificazione delle immagini, sfruttano l'idea delle convoluzioni per catturare le caratteristiche spaziali e locali di un'immagine. Le convoluzioni applicate su immagini si basano sull'assunzione che i dati siano strutturati in una griglia regolare (come i pixel in un'immagine), il che consente di applicare filtri convolutivi per estrarre caratteristiche come bordi, texture e forme.
Quando si tratta di dati strutturati come grafi, la sfida è che questi non seguono una struttura regolare come le immagini. I nodi di un grafo possono avere un numero arbitrario di vicini e le connessioni tra i nodi non formano una griglia ordinata. Tuttavia, le idee fondamentali delle CNN possono essere adattate per lavorare anche con i grafi.
La Graph Convolutional Network (GCN) è un'estensione della CNN che applica il concetto di convoluzione ai grafi. Invece di operare su una griglia di pixel, una GCN esegue convoluzioni direttamente sulla struttura del grafo, considerando i nodi e i loro vicini.
L'adattamento delle CNN per la classificazione dei grafi ha portato a sviluppare modelli di reti neurali deep come la DGCNN (Deep Graph Convolutional Neural Network), progettata per classificare grafi mantenendo la loro struttura intrinseca.

L'architettura della Deep Graph Convolutional Neural Network (DGCNN) è progettata per risolvere alcune delle problematiche più significative nella classificazione dei grafi.
La prima sfida riguarda l'estrazione delle caratteristiche dai grafi. Infatti, a differenza delle immagini o delle serie temporali, i grafi non possiedono una struttura fissa o ordinata. I nodi e le connessioni possono essere disposti in modo molto vario, il che rende difficile stabilire un modo coerente per rappresentare queste strutture in un formato che una rete neurale possa elaborare.
La seconda sfida è legata all'ordinamento dei nodi del grafo. In un'immagine, i

pixel hanno un ordine naturale che è dato dalla loro posizione nello spazio bidimensionale. Al contrario, nei grafi non esiste un ordine prestabilito dei nodi. Questo complica l'applicazione di metodi tradizionali di apprendimento automatico che di solito richiedono un input ordinato. Ad esempio, nei modelli convoluzionali per le immagini, l'ordine dei pixel è essenziale per applicare correttamente le operazioni di convoluzione. Nel caso dei grafi, è necessario trovare un modo per rappresentarli in un formato che permetta un ordinamento coerente dei vertici, in maniera che il modello possa elaborare i dati in maniera efficace.
L'architettura DGCNN affronta queste problematiche attraverso l'uso di strati convoluzionali progettati per funzionare sui grafi, seguiti da un meccanismo di ordinamento (*SortPooling*) che permette di convertire la struttura non ordinata in una sequenza ordinata che può essere elaborata dai successivi strati della rete neurale. Questo approccio consente alla rete di apprendere rappresentazioni significative dei grafi, rendendo possibile la loro classificazione in modo efficiente e accurato.

4.1 Architettura delle DGCNN

L'architettura DGCNN, illustrata nella Figura 4.1, è suddivisa in tre fasi principali: i layers di convoluzione sui grafi (*Graph Convolution Layer*), il layer di ordinamento (*SortPooling Layer*) e i layers finali, che includono il layer convoluzionale 1-D e i *Dense Layers*. (*Remaining Layer*).

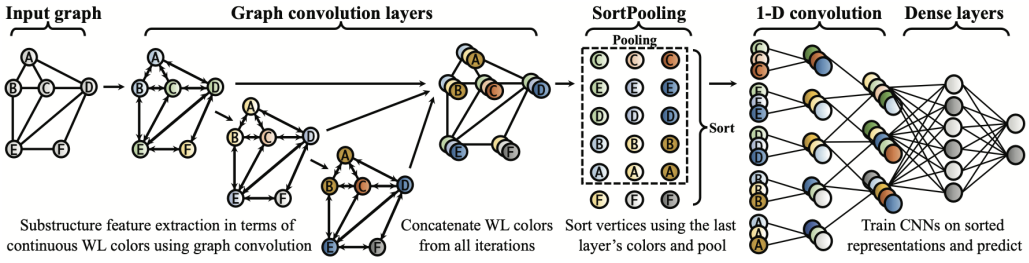


Figura 4.1: Architettura della DGCNN [Zhang et al., 2018]

L'architettura della DGCNN si distingue nettamente nel panorama delle tecniche di classificazione dei grafi grazie alla sua capacità di gestire direttamente i grafi nella loro forma nativa, senza la necessità di trasformarli in vettori o matrici con un ordinamento fisso. Questo rappresenta un significativo passo avanti rispetto alle tecniche tradizionali, che spesso richiedono complesse fasi di pre-elaborazione per adattare i grafi a formati più convenzionali al fine di poter essere trattati dai modelli di machine learning.
In queste tecniche, la trasformazione dei grafi in rappresentazioni con ordinamento fisso non solo richiede tempo, ma comporta anche il rischio di perdita di informazioni strutturali cruciali, compromettendo la qualità delle previsioni del modello.

Con questa configuazione, invece, non sono necessari passaggi intermedi per la trasformazione dei dati, che potrebbero introdurre inefficienze e ridurre la precisione complessiva del modello. I modelli tradizionali, infatti, sono spesso vincolati da ipotesi rigide sull'input che possono non catturare appieno la complessità intrinseca delle strutture dei grafi. Al contrario, la DGCNN lavora direttamente con la rappresentazione originale del grafo, mantenendo intatte tutte le informazioni strutturali e topologiche.
La DGCNN supera queste limitazioni offrendo una pipeline di addestramento end-to-end che consente al modello di apprendere direttamente dalle strutture dei grafi. In un'architettura end-to-end, l'intero processo, dalla ricezione dei dati grezzi all'emissione delle previsioni, è integrato in un unico flusso continuo. Questo permette alla rete di aggiornare e ottimizzare tutti i suoi parametri in modo simultaneo durante l'addestramento, sfruttando appieno le informazioni che si propagano attraverso i livelli della rete.

In sostanza, questo approccio non solo preserva la ricchezza dei dati originali, ma permette anche alla rete di sfruttare appieno le caratteristiche specifiche dei grafi, come le relazioni tra i nodi e le connessioni tra le varie parti del grafo. La capacità di operare direttamente sui grafi nella loro forma nativa rende la DGCNN particolarmente adatta a compiti complessi di classificazione dove la topologia e le caratteristiche strutturali del grafo sono cruciali per la qualità della previsione. Inoltre, l'approccio end-to-end consente una maggiore flessibilità e adattabilità del modello, poiché tutti i parametri della rete possono essere ottimizzati in modo sinergico, migliorando così l'efficacia e l'efficienza del processo di apprendimento.

4.1.1 Graph Convolutional Layers

Gli strati di convoluzione sui grafi nella DGCNN sono una componente fondamentale progettata per estrarre rappresentazioni significative dai nodi. Questi strati operano in modo diverso rispetto alle convoluzioni tradizionali applicate su immagini, a causa della natura non strutturata dei grafi.

Il processo inizia con un input costituito da un grafo. Ogni nodo del grafo è rappresentato da un vettore di feature. In questo caso, ogni vettore contiene le tre feature temporali calcolate come descritto nella sezione 3.3.2 e un one-hot encoding che rappresenta la classe del nodo. In realtà, questi vettori di feature non si limitano ai soli nodi del grafo originale ma includono anche i vettori di feature dei nodi provenienti dai prefissi attivi. Per distinguere i nodi del grafo originale da quelli degli altri prefissi, viene aggiunto un valore alla fine del vettore di feature: 1 per i nodi del grafo originale e 0 negli altri casi.
Dunque, supponendo che ci siano C classi possibili, in questo caso, il vettore completo delle feature per ciascun nodo avrà dimensione $C + 3 + 1$.

Questi vettori di feature costituiscono la matrice di input X, dove ogni riga rappresenta un nodo e le colonne rappresentano le diverse feature.
Quindi, se i nodi del grafo di partenza e quelli relativi ai prefissi attivi sono N, la matrice X avrà dimensioni $N \times (C + 3 + 1)$.
La matrice di input X può essere rappresentata come segue:

$$X = \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} & \text{one-hot}_{1,1} & \text{one-hot}_{1,2} & \cdots & \text{one-hot}_{1,C} & \text{orig}_1 \\ f_{2,1} & f_{2,2} & f_{2,3} & \text{one-hot}_{2,1} & \text{one-hot}_{2,2} & \cdots & \text{one-hot}_{2,C} & \text{orig}_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ f_{N,1} & f_{N,2} & f_{N,3} & \text{one-hot}_{N,1} & \text{one-hot}_{N,2} & \cdots & \text{one-hot}_{N,C} & \text{orig}_N \end{bmatrix}$$

Dove:

- $f_{i,1}$, $f_{i,2}$, e $f_{i,3}$ sono le 3 feature temporali del nodo i.
- $\text{one-hot}_{i,j}$ è il valore del one-hot encoding per la j-esima classe del nodo i.
- orig_i è un valore che indica se il nodo i è del grafo originale (1) o se appartiene ad uno dei prefissi attivi (0).

Anche la matrice di adiacenza A relativa ad X riflette la struttura disconnessa.
In generale, una matrice di adiacenza rappresenta le connessioni tra i nodi del grafo. Se esiste un arco tra i nodi i e j, l'elemento corrispondente nella matrice è 1, altrimenti è 0. Questa matrice descrive come i nodi sono connessi tra loro.
In questo caso, A è una matrice a blocchi, dove ogni blocco rappresenta le connessioni all'interno di uno dei grafi disconnessi, mentre non ci sono connessioni tra i blocchi stessi.

A differenza delle immagini, dove i pixel sono organizzati in una griglia regolare e ben definita, i grafi non hanno una struttura fissa. Infatti, i nodi in un grafo possono avere connessioni che variano notevolmente, creando una topologia molto più complessa e meno regolare rispetto a una griglia di pixel. Questo richiede un approccio diverso per applicare la convoluzione, adattandosi alla struttura variabile dei grafi.
Infatti, gli strati di convoluzione sui grafi propagano informazioni tra i nodi adiacenti, utilizzando la matrice di adiacenza A.
Durante l'elaborazione negli strati di convoluzione sui grafi, la matrice X, che include le feature temporali, il one-hot encoding della classe e il bit di distinzione tra nodi, viene combinata con la matrice di adiacenza A.
La convoluzione applicata in ciascun strato considera questa matrice di adiacenza disconnessa, propagando le informazioni solo tra i nodi all'interno dello stesso grafo. Ogni nodo, quindi, aggiorna le proprie feature combinando le sue caratteristiche con quelle dei suoi vicini, ma solo all'interno della sua componente connessa.

Per garantire che ogni nodo possa includere le proprie caratteristiche oltre a quelle dei nodi vicini, la matrice A viene aggiunta alla matrice identità I ottenendo così una nuova matrice $\tilde{A} = A + I$. L'aggiunta di questi *self-loop* assicura che ogni nodo possa considerare anche se stesso nella propagazione delle informazioni.
La convoluzione vera e propria avviene moltiplicando $\tilde{D}^{-1}$ per $\tilde{A}$ e poi per X, seguito da una moltiplicazione con la matrice dei pesi W e dall'applicazione di una funzione di attivazione non lineare f come $ReLU$.
Questa operazione può essere espressa come

$$Z = f(\tilde{D}^{-1}\tilde{A}XW),$$

dove Z rappresenta la nuova matrice di feature per i nodi mentre $\tilde{D}$ è la matrice dei gradi, ovvero, una matrice diagonale dove ogni elemento D_{ii} è il grado del nodo i, ovvero il numero di connessioni che il nodo ha.
Questo passaggio trasforma le feature originali di ciascun nodo, integrando informazioni dai nodi vicini e, in questo modo, catturando la struttura locale del grafo.
L'output di ciascun strato di convoluzione è una nuova rappresentazione delle caratteristiche dei nodi, arricchita dalle informazioni sul contesto locale dei nodi nel grafo. Questo processo può essere ripetuto attraverso più strati di convoluzione, permettendo alla rete di catturare informazioni sempre più globali e complesse, propagandosi lungo il grafo. Ogni strato successivo raffina la rappresentazione delle caratteristiche, rendendo possibile la costruzione di rappresentazioni gerarchiche del grafo.

4.1.2 SortPooling Layer

La funzione principale del SortPooling layer è quella di ordinare le feature dei vertici del grafo che sono state precedentemente estratte dagli strati di convoluzione secondo un criterio specifico. Le feature ordinate vengono poi concatenate per formare un vettore di dimensione fissa, indipendentemente dal numero di vertici originali del grafo.
Questo processo consente di ottenere una rappresentazione compatta e invariabile rispetto all'ordine dei vertici del grafo, garantendo al contempo una rappresentazione ordinata che può essere utilizzata dai layer successivi della rete. Infatti, diversamente dalle immagini, che hanno una dimensione costante, i grafi possono avere un numero variabile di nodi, il che rende difficile utilizzare direttamente strati che richiedono una dimensione di input fissa. Per questo, il SortPooling layer è necessario per gestire grafi di dimensioni variabili e, allo stesso tempo, per preservare informazioni topologiche rilevanti per la predizione.

In particolare, dopo aver processato un grafo $G = (V, E)$ attraverso diversi strati di convoluzione, ogni nodo $v_i \in V$ è rappresentato da un vettore di feature $h_i \in \mathbb{R}^d$ dove d è la dimensione del vettore.

Questi vettori di caratteristiche per tutti i nodi formano una matrice $H \in \mathbb{R}^{n \times d}$, ottenuta al termine dell'applicazione di tutti gli strati di convoluzione, dove n è il numero di nodi nel grafo:

$$H = \begin{pmatrix} h_1^1 & h_1^2 & \dots & h_1^d \\ h_2^1 & h_2^2 & \dots & h_2^d \\ \vdots & \vdots & \ddots & \vdots \\ h_n^1 & h_n^2 & \dots & h_n^d \end{pmatrix}$$

La sfida è che n può variare tra diversi grafi, rendendo difficile applicare tecniche che richiedono input di dimensioni fisse.

Il passaggio successivo nel SortPooling layer è ordinare i nodi del grafo. Questo ordinamento potrebbe essere fatto secondo la prima colonna della matrice H, infatti, durante il processo di convoluzione sul grafo, i valori nella prima colonna possono catturare informazioni critiche sulla posizione dei nodi nel contesto del grafo, come la centralità, il grado o altre proprietà che emergono dalle convoluzioni.
Più comunemente, invece di ordinare i nodi basandosi su una singola colonna di H, come la prima, l'ordinamento si basa sulla norma euclidea dei vettori di feature di ciascun nodo.
La norma di un vettore h_i è calcolata come:

$$\|h_i\| = \sqrt{(h_i^1)^2 + (h_i^2)^2 + \cdots + (h_i^d)^2}$$

Questa norma rappresenta l'intensità complessiva delle feature di un nodo e fornisce una misura scalare che sintetizza l'importanza del nodo nel grafo dopo le convoluzioni.
Il SortPooling layer ordina i nodi in base ai valori decrescenti di $\|h_i\|$, ovvero, i nodi con norme più elevate, che rappresentano nodi con caratteristiche più forti, vengono posizionati per primi. La permutazione degli indici π che ordina le norme dei nodi è definita come:

$$\|h_{\pi(1)}\| \geq \|h_{\pi(2)}\| \geq \cdots \geq \|h_{\pi(n)}\|$$

La matrice delle caratteristiche ordinata H_{sorted} è ottenuta riordinando le righe di H secondo questa permutazione π:

$$H_{\text{sorted}} = \begin{pmatrix} h_{\pi(1)}^1 & h_{\pi(1)}^2 & \dots & h_{\pi(1)}^d \\ h_{\pi(2)}^1 & h_{\pi(2)}^2 & \dots & h_{\pi(2)}^d \\ \vdots & \vdots & \ddots & \vdots \\ h_{\pi(n)}^1 & h_{\pi(n)}^2 & \dots & h_{\pi(n)}^d \end{pmatrix}$$

Dopo l'ordinamento, il SortPooling layer affronta il problema delle dimensioni variabili

del grafo fissando la dimensione del grafo ordinato a k, un valore predefinito. Se il grafo ha più di k nodi, si considerano solo i primi k nodi ordinati (troncamento). Se il grafo ha meno di k nodi, vengono aggiunte righe di padding (di solito vettori di zeri) per raggiungere la dimensione $k \times d$ (zero-padding):

$$H_{\text{sorted, truncated}} = H_{\text{sorted}}[1:k]$$

Questo processo garantisce che la matrice finale $H_{\text{sorted, truncated}}$ abbia una dimensione fissa $k \times d$, indipendentemente dalla dimensione originale del grafo.
La matrice $H_{\text{sorted, truncated}}$ può ora essere appiattita (flattened) in un vettore di dimensione $k \cdot d$ per essere utilizzata nei layer successivi. Questo consente di integrare i grafi in modelli di deep learning che richiedono input di dimensioni fisse, mantenendo al contempo una rappresentazione coerente della struttura del grafo.

4.1.3 Remaining Layers

Dopo che i nodi del grafo sono stati ordinati e ridimensionati nel SortPooling layer, la rete può trattare la matrice $H_{\text{sorted, truncated}}$ come una sequenza lineare di dati.
A questo punto, la matrice viene passata attraverso uno o più livelli di convoluzione 1-D. Questi strati operano su ogni riga della matrice, considerata come una serie temporale o sequenza di caratteristiche, e applicano filtri convoluzionali per estrarre pattern significativi. La convoluzione 1D permette alla rete di catturare relazioni locali tra i nodi ordinati, che sono cruciali per comprendere le strutture più complesse e globali del grafo.
Ogni filtro convoluzionale 1-D combina le feature di un gruppo di nodi adiacenti per produrre una nuova rappresentazione.
Se $W_c \in \mathbb{R}^{f \times d}$ rappresenta il kernel convoluzionale con lunghezza f, il layer convoluzionale genera nuove feature che riflettono una combinazione pesata delle feature dei nodi all'interno di quella finestra di f nodi. Questa operazione consente di creare feature più astratte che non rappresentano solo le caratteristiche dei singoli nodi, ma anche le loro relazioni con i nodi vicini.
Attraverso le convoluzioni 1-D, il modello riduce anche la dimensionalità spaziale della rappresentazione del grafo, mantenendo le informazioni più significative e permettendo la successiva applicazione di layer fully connected per la classificazione finale.

Successivamente, il risultato della convoluzione viene ulteriormente ridotto attraverso strati di pooling, come il MaxPooling, che seleziona i valori massimi di ogni filtro convoluzionale. Questo processo riduce la dimensionalità dei dati, eliminando informazioni ridondanti e mantenendo solo le caratteristiche più rilevanti per la classificazione. Il pooling permette anche di rendere la rappresentazione meno sensibile a piccole variazioni nell'ordine dei nodi, aumentando la robustezza del modello.

Dopo l'applicazione della convoluzione 1D e del MaxPooling, la rappresentazione del grafo, che ora è una sequenza compatta di caratteristiche, viene appiattita in un vettore unidimensionale. Questo vettore di dimensione n', dove n' dipende dal numero di filtri convoluzionali e dal pooling applicato, viene quindi passato attraverso uno o più strati fully connected, che hanno il compito di integrare e combinare le informazioni estratte in precedenza per produrre la classificazione finale.
Ogni fully connected layer applica una trasformazione lineare al suo input, seguita da una funzione di attivazione non lineare.
Se l'input al primo fully connected layer è denotato con il vettore $x \in \mathbb{R}^{n'}$, l'output z di questo layer è calcolato come:

$$z = W \cdot x + b$$

dove:

- $W \in \mathbb{R}^{m \times n'}$ è la matrice dei pesi del fully connected layers, con m unità (neuroni) nel layer;
- $b \in \mathbb{R}^m$ è il vettore di bias, aggiunto per ogni neurone;
- $z \in \mathbb{R}^m$ è il vettore risultante dalla combinazione lineare delle caratteristiche in input.

Dopo la trasformazione lineare, viene applicata una funzione di attivazione non lineare, tipicamente ReLU (Rectified Linear Unit), definita come:

$$a = \sigma(z) = \max(0, z)$$

dove $a \in \mathbb{R}^m$ è l'output del fully connected layer.
La funzione ReLU imposta a zero tutti i valori negativi nel vettore z, mantenendo inalterati quelli positivi. Questo introduce non linearità nel modello, permettendo alla rete di apprendere relazioni complesse.
Gli strati fully connected vengono spesso impilati in modo che questo processo di applicazione di trasformazioni lineari e attivazioni non lineari permetta alla rete di combinare le caratteristiche in modo sempre più astratto e complesso, distillando l'informazione rilevante per la classificazione finale.

4.1.4 Predizione

L'ultimo fully connected layer è generalmente seguito da uno strato di uscita che fornisce le probabilità di appartenenza alle diverse classi.
Se il compito è la classificazione multi-classe, lo strato di uscita utilizza una funzione di attivazione *softmax*, che converte l'output lineare in una distribuzione di probabilità:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

dove:

- p_i è la probabilità che l'input appartenga alla classe i;
- C è il numero totale delle classi;
- z_i è l'output lineare corrispondente alla classe i.

Il modello predice quindi la classe del grafo scegliendo la classe con la probabilità più alta:

$$\hat{y} = \arg\max_j \hat{p}_i$$

Capitolo 5

Esperimenti

Questo capitolo presenta una serie di esperimenti progettati per valutare l'efficacia dell'approccio proposto. I risultati verranno confrontati con l'articolo *Multi-Perspective Enriched Instance Graphs* [Chiorrini et al., 2023] in modo da contestualizzare il lavoro all'interno della letteratura esistente ed identificare possibili direzioni per ricerche future.
A tale scopo, è stato utilizzato uno dei dataset proposti nell'articolo, permettendo una valutazione precisa delle prestazioni in termini di accuratezza e F1-weighted. L'adozione dello stesso dataset consente un confronto diretto dei risultati, fornendo una base solida per analizzare le differenze nelle prestazioni tra i due approcci.

Nel corso delle seguenti sezioni, viene illustrato il design sperimentale, con una descrizione dettagliata dei parametri e delle procedure adottate per l'analisi dei dati. I risultati vengono discussi criticamente, evidenziando i principali punti di forza e le potenziali aree di miglioramento rispetto all'approccio di riferimento.

5.1 Il dataset

Il dataset scelto per condurre gli esperimenti è *Prepaid Travel Costs* [van Dongen, 2020] che contiene dati raccolti per due anni riguardo le spese sostenute per viaggi prepagati all'interno del processo di gestione delle trasferte aziendali dei dipendenti della Eindhoven University of Technology (TU/e).
Nello specifico, come mostrato in Tabella 5.1, il dataset contiene informazioni su 2.099 tracce e 18.246 eventi legati alle richieste di pagamento anticipato.
Gli eventi documentano il processo di approvazione e rimborso delle spese di viaggio, che include la presentazione di richieste da parte dei dipendenti, l'approvazione da parte dell'amministrazione e il passaggio attraverso il responsabile del budget e il supervisore, con ulteriori approvazioni da parte del direttore in alcuni casi.
Questo tipo di log risulta utile per analizzare i tempi di elaborazione delle richieste, le iterazioni che possono verificarsi in caso di errori o modifiche necessarie, nonché l'efficacia complessiva del flusso di lavoro.

N. di tracce	Eventi totali	N. di tipi di attività	Min $\|\sigma\|$	Max $\|\sigma\|$	Media $\|\sigma\|$
2099	18246	29	1	21	9

Tabella 5.1: Caratteristiche dei dataset Prepaid Travel Costs

Come mostrato in Figura 5.1, è importante notare che il dataset presenta un significativo sbilanciamento.
In particolare, alcuni tipi di attività, come la richiesta iniziale e l'approvazione, sono molto più frequenti rispetto ad altre, come l'intervento del direttore, che avviene solo in un numero limitato di casi. Di conseguenza, l'analisi dovrà tenere conto di questo sbilanciamento per garantire una valutazione equilibrata delle performance.

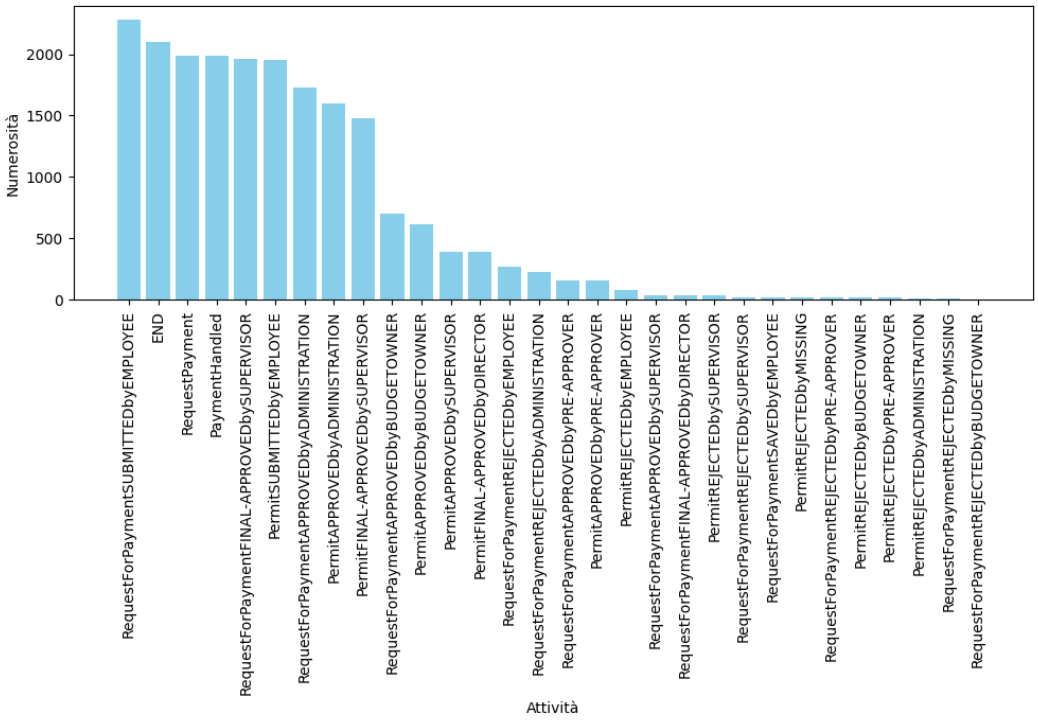


Figura 5.1: Distribuzione delle classi nel dataset *Prepaid Travel Costs*

5.2 Setup sperimentale

Come riportato nell'articolo [Chiorrini et al., 2023], la suddivisione del dataset è stata eseguita con un rapporto di $67\% - 33\%$ per l'addestramento e il test, rispettando l'ordinamento cronologico delle tracce e applicando una stratificazione.
Mantenere l'ordine temporale è cruciale per la previsione della prossima attività, poiché consente al modello di apprendere le dipendenze temporali. Inoltre, questo approccio simula meglio le condizioni reali in cui il modello deve prevedere eventi futuri basandosi su informazioni passate, evitando così che i dati di addestramento e

test si sovrappongano.
La stratificazione è stata implementata per assicurare che la distribuzione delle classi sia proporzionale sia nel set di addestramento che in quello di test. Tale scelta evita il rischio di sbilanciamento tra le classi, che potrebbe influenzare negativamente le prestazioni del modello.

5.2.1 Feature inter-case

Per quanto riguarda il calcolo delle feature inter-case, si è deciso di ricavare i prefissi attivi considerando l'intero dataset, cioè, prima di suddividere i dati in set di addestramento e test in modo da garantire una rappresentazione più completa e coerente delle esecuzioni parallele.
Tale valutazione sorge dal calcolo dell'intensità dei prefissi attivi, ottenuta considerando il numero totale di nodi associato a ciascun prefisso attivo relativo ad un prefisso di riferimento. Questa metrica è utile in quanto tiene in conto sia della lunghezza dei prefissi attivi sia della loro quantità complessiva, fornendo così una misura sintetica dell'intensità.
Infatti, osservando i dati riportati in Tabella 5.2, è evidente come l'intensità media dei prefissi attivi massimi cambi significativamente tra il momento precedente e successivo allo split. Nel caso del train, ad esempio, la numerosità media dei nodi dei prefissi attivi passa da 2136.22 a 2235.51, mentre nel test la differenza è ancora più marcata, con un aumento da 1181.48 a 1724.14.
In particolare, quando i prefissi attivi vengono calcolati prima dello split, utilizzando l'intero dataset, la differenza tra le statistiche di train e test risulta meno marcata. Questo avviene perché tutti i dati, sia di train che di test, vengono utilizzati nel calcolo, garantendo una maggiore omogeneità.
Tuttavia, quando i prefissi attivi vengono considerati separatamente su train e test dopo lo split, le differenze nelle statistiche diventano più evidenti. Infatti, in questo modo, nel test, i prefissi attivi non possono includere dati provenienti dal train, e viceversa. Di conseguenza, i prefissi attivi nel set di test vengono calcolati su una porzione più limitata di dati, con un impatto diretto sulla numerosità dei nodi degli stessi.

Set	**Split**	**Media dei nodi attivi**
Train	prima	2136.22
	dopo	2235.51
Test	prima	1181.48
	dopo	1724.14

Tabella 5.2: Media del numero di nodi relativi ai prefissi attivi calcolati prima e dopo lo split del dataset

È importante sottolineare che, in realtà, questa differenza è in parte insita nella natura cronologica dello split.
A dimostrazione di ciò, si osservino i grafici di Figura 5.2 che mostrano la distribuzione dell'intensità dei prefissi attivi massimi al variare della lunghezza del prefisso di riferimento per classe, dove lo split è stato eseguito dopo il calcolo dei prefissi attivi.

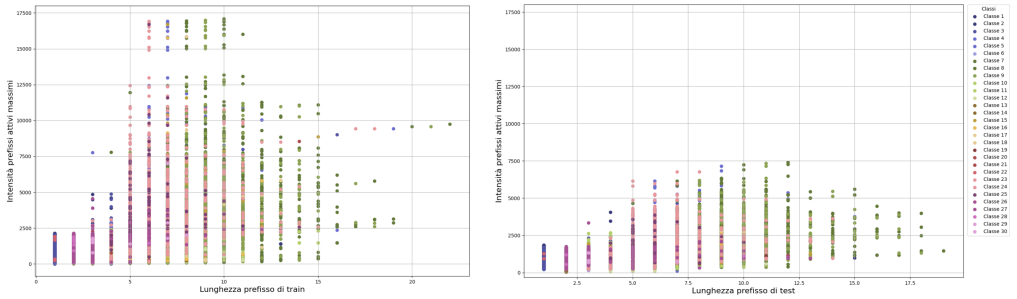


Figura 5.2: Distribuzione dell'intensità dei prefissi attivi massimi al variare della lunghezza del prefisso di riferimento per classe in train e test secondo lo split $67\% - 33\%$ delle tracce ordinate cronologicamente effettuato dopo aver calcolato i prefissi attivi massimi

L'immagine mostra che, anche quando i prefissi attivi vengono calcolati prima dello split, è normale che mediamente la dimensione dei prefissi nel set di test sia inferiore rispetto a quella del train.
Infatti, quando il dataset viene suddiviso in modo cronologico, il set di test è costituito dagli ultimi eventi del processo, che spesso includono le fasi conclusive, caratterizzate da una minore densità di attività. Al contrario, il set di train contiene la parte centrale del processo, che generalmente è più ricca di eventi, e quindi produce prefissi attivi più intensi.
Questo effetto sarebbe ulteriormente amplificato se i prefissi attivi fossero calcolati dopo lo split. In tal caso, il test, non potendo considerare i dati del train, si baserebbe su una porzione più ridotta e meno densa di informazioni, comportando un'intensità media dei prefissi attivi significativamente inferiore. Il train, al contrario, includerebbe una maggiore quantità di dati, permettendo il calcolo di più prefissi e statistiche più stabili.
In questo modo, se i dati del train e del test sono troppo diversi, il modello potrebbe non riuscire a generalizzare adeguatamente a causa delle differenze significative tra i due set di dati.
Invece, nel caso in cui lo split $67\% - 33\%$ non segue l'ordine cronologico, ma avviene in modo casuale, questa disparità si riduce notevolmente come è possibile notare osservando la Figura 5.3.
In uno split non cronologico, i dati sono distribuiti in modo più uniforme tra train e test, poiché ogni elemento del dataset (inclusi gli eventi sia iniziali che centrali e

finali) potrebbe finire in entrambe le partizioni. Di conseguenza, sia il set di train che quello di test avranno una rappresentazione più omogenea di tutto il processo, e i prefissi attivi saranno distribuiti in maniera più bilanciata.

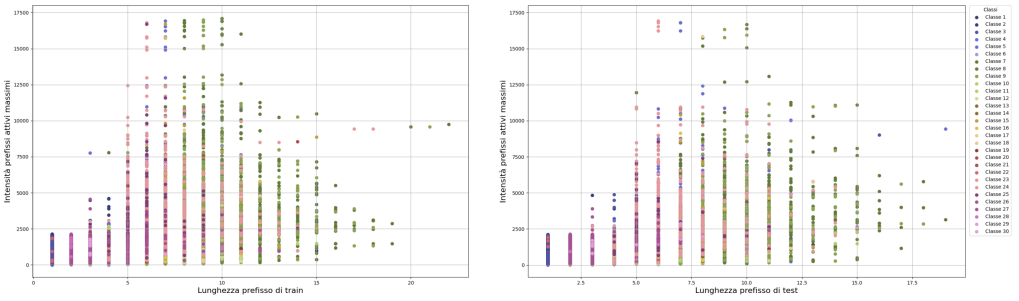


Figura 5.3: Distribuzione delle intensità dei prefissi attivi massimi al variare della lunghezza del prefisso di riferimento per classe in train e test secondo lo split $67\% - 33\%$ random effettuato dopo aver calcolato i prefissi attivi

Inoltre, c'è un altro problema legato al calcolo separato, tra train e test, dei prefissi attivi: la numerosità dei prefissi attivi è direttamente influenzata dalla quantità di dati. Ciò significa che un set di dati più ampio permette di identificare una maggiore varietà di prefissi, mentre un set ridotto potrebbe limitarne drasticamente la disponibilità.
Nel caso estremo, se nel set di test fosse presente un unico elemento, non vi sarebbero prefissi attivi su cui basare le previsioni, in quanto non ci sarebbe alcun riferimento con altre sequenze.
Questo evidenzia come la qualità del modello dipenda fortemente dalla quantità e varietà di dati presenti nel set di test.
Per evitare questa problematica, è più opportuno calcolare i prefissi attivi sull'intero dataset, includendo sia i dati di training che quelli di test, ovvero, includendo tutti i dati del contesto di riferimento.

Appreso che risulta conveniente calcolare i prefissi attivi sull'intero dataset prima della suddivisione tra training e test set, emerge un'altra questione.
Poiché le tracce sono ordinate cronologicamente, può capitare che due istanze temporalmente vicine finiscano una nel training set e l'altra nel test set. Se questi prefissi sono eseguiti nello stesso intervallo temporale, esiste la possibilità che i prefissi attivi calcolati per questi campioni siano identici, con il rischio che il test set contenga informazioni già presenti nel training set, causando una sovrastima delle performance del modello.
Tuttavia, ciò avviene solo se anche le classi di appartenenza coincidono e, poiché il dataset è suddiviso in modo stratificato, non è garantito che due istanze di classi

diverse siano così vicine temporalmente, riducendo quindi la probabilità che questo fenomeno si verifichi.
Ciò, dato che lo split del dataset avviene classe per classe, ogni categoria, che rappresenta un'attività della traccia, viene trattata separatamente. Di conseguenza, quando attività diverse avvengono in momenti sovrapposti o ravvicinati all'interno delle tracce, le classi possono intrecciarsi tra loro. Dunque, anche se c'è un ordinamento cronologico, non è detto che due campioni vicini temporalmente appartengano alla stessa classe. Questo riduce la probabilità che prefissi attivi simili o identici vengano inclusi sia nel training che nel test set, diminuendo il rischio di sovrastimare le performance del modello.
Per comprendere meglio la questione, è utile osservare la distribuzione temporale dei dati all'interno delle classi.
Ad esempio, si consideri la classe *RequestForPayementSUBMITTEDbyEMPOLOYEE* che risulta essere la più numerosa del dataset. In Figura 5.4 sono mostrate le distanze degli elementi ricavati considerando il punto di split, ovvero, prendendo il 67° percento dei dati della classe e selezionando 15 elementi a sinistra (per il training) e 15 elementi a destra (per il test). Gli elementi sono stati etichettati come $0, 1, 2, \ldots, 14$ sia per il training che per il test, con il punto di split che funge da riferimento centrale.

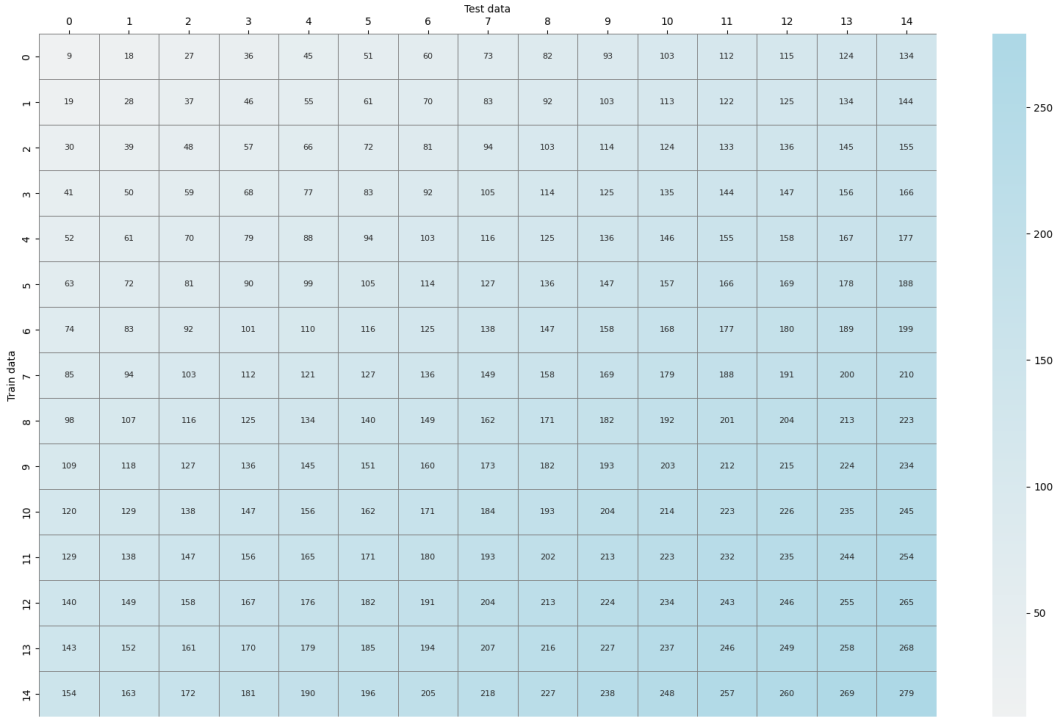


Figura 5.4: Distanza tra gli utlimi 15 elementi di train e i primi 15 elementi di test della classe *RequestForPayementSUBMITTEDbyEMPOLOYEE*

Come è facile osservare, la distanza aumenta con l'indice. Inoltre, si noti come elementi vicini all'interno della classe, ovvero, che corrispondono a numeri immediatamente

progressivi di indice, non sono collocati a distanza 0 l'uno dall'altro. Questo signfica che tra un elemento e l'altro della classe possono presentarsi numerosi elementi appartenenti ad altre classi, il che evidenzia la complessità nell'esecuzione del processo.
A questo punto, diventa fondamentale comprendere quanto siano simili i dati tra il training set e il test set e come vari questa somiglianza.
Per farlo, è possibile analizzare la quantità di tensori identici tra gli elementi. Infatti, ogni tensore rappresenta un nodo, pertanto confrontare i tensori equivale a confrontare i nodi dei prefissi attivi massimi. Questo confronto permette di quantificare la sovrapposizione tra i due set e di valutare eventuali implicazioni sul modello.
La matrice in Figura 5.5 mostra chiaramente come la similarità tra gli elementi di train e test non sia uniforme, ma vari principalmente in funzione della distanza e della intensità.

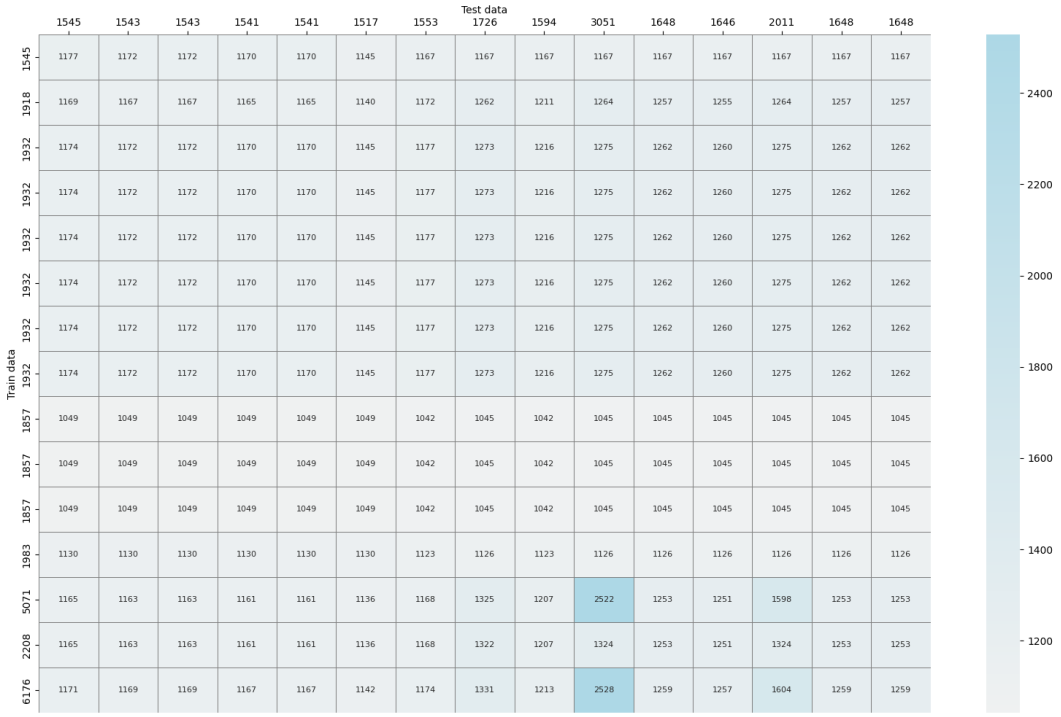


Figura 5.5: Numero di tensori comuni tra gli utlimi 15 elementi di train e i primi 15 elementi di test della classe *RequestForPayementSUBMITTEDbyEMPOLOYEE*

Ovvero, gli elementi più vicini tra loro, sia nel train che nel test, tendono a condividere più tensori identici, suggerendo una maggiore similarità. Man mano che ci si allontana dalla prima metà della matrice, la numerosità di valori comuni tende a diminuire, poiché gli elementi confrontati sono più distanti tra loro temporalmente. Questo riflette il fatto che la vicinanza temporale tra gli elementi è un fattore che influenza la quantità di tensori in comune.
Tuttavia, in alcune aree della matrice si osservano blocchi di celle con valori significa-

tivamente più alti, dove il numero di elementi in comune è molto più alto. Questi picchi di intensità sono dovuti al fatto che, i rispettivi elementi di train e test sono caratterizzati da una quantità maggiore di nodi attivi.
Per rendere il confronto equo e indipendente dall'intensità dei prefissi attivi dei dati, è utile normalizzare le celle della matrice in modo che la misura di similarità sia sempre compresa tra 0 e 1:

$$\text{Similarità} = \frac{C}{\max(N_i, N_j)} \tag{5.1}$$

dove:

- C è il numero di valori in comune,
- N_i è il numero di nodi attivi nella riga i,
- N_j è il numero di nodi attivi nella colonna j.

Il risultato della normalizzazione è riportato in Figura 5.6.

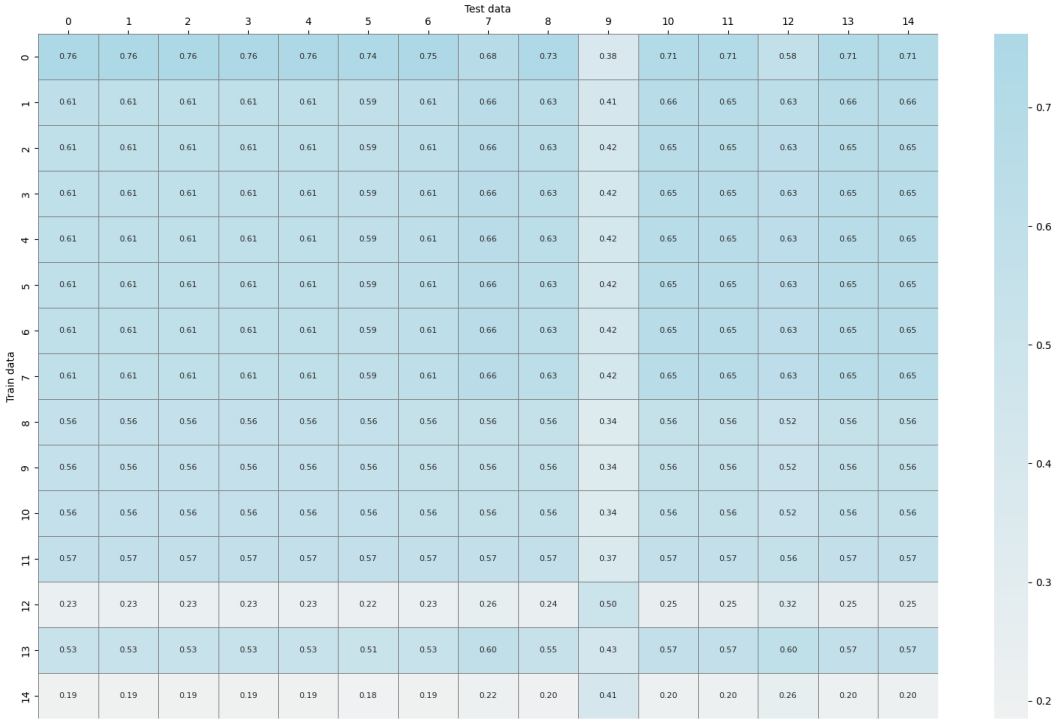


Figura 5.6: Similarità tra gli utlimi 15 elementi di train e i primi 15 elementi di test della classe *RequestForPayementSUBMITTEDbyEMPOLOYEE*

La similarità è quindi calcolata come il numero di valori in comune rispetto al numero massimo di nodi attivi tra la riga e la colonna. Questo approccio rende il confronto indipendente dalla dimensione assoluta delle righe e colonne, evitando che dimensioni

maggiori distorcano la misura di similarità.
Si noti come la similarità tra gli elementi dipenda da due fattori principali: la loro vicinanza temporale e la loro estensione nel tempo.
Analizzando la matrice di similarità, infatti, si osserva che i valori sotto la diagonale principale, che rappresentano prefissi con una maggiore distanza temporale, tendono ad essere più chiari. Ciò indica una diminuzione della similarità man mano che la distanza temporale tra i prefissi aumenta. Tale fenomeno suggerisce che la similarità tra i prefissi diminuisce con il passare del tempo, riflettendo una diminuzione della correlazione tra prefissi più lontani temporalmente.
In aggiunta, la similarità è influenzata dall'estensione temporale dei prefissi di riferimento. Per esempio, osservando la prima colonna della matrice, emerge che, probabilmente, l'elemento 0 dell'insieme di addestramento si estende temporalmente e condivide gli stessi prefissi attivi con un ampio intervallo di elementi nell'insieme di test, dall'indice 0 all'indice 14. Questo suggerisce che alcuni di questi prefissi si stanno sviluppando in parallelo.
In altre parole, un prefisso dell'insieme di addestramento che ha una lunga estensione temporale e che condivide i prefissi con una vasta gamma di elementi dell'insieme di test indica una correlazione più forte con i prefissi testati che si evolvono in modo parallelo.
Nonostante queste osservazioni, è rilevante notare che i valori di similarità non raggiungono mai 1, nemmeno per le altre classi. Questo risultato indica che non esiste una corrispondenza perfetta tra i prefissi analizzati, suggerendo che la scelta di eseguire lo split dopo il calcolo dei prefissi attivi massimi non distorcerà le performance del modello.

5.3 Metriche

Durante gli esperimenti, è stato fondamentale valutare sia l'efficacia che l'affidabilità del lavoro svolto. Per garantire la coerenza con gli studi precedenti, sono state utilizzate le stesse metriche di valutazione adottate nell'articolo di riferimento, ossia l'*Accuracy* e la *Weighted-F1 score*.

L'accuratezza misura il numero di classificazioni corrette, rispetto al numero totale di casi. Dunque, fornisce una misura della correttezza globale delle previsioni del modello e viene calcolata con la seguente formula:

$$\text{Accuratezza} = \frac{\text{Numero di predizioni corrette}}{\text{Numero totale di predizioni}} \tag{5.2}$$

La *F1-score*, invece, è la media armonica di:

- *Precision*, che misura gli elementi correttamente classificati di ogni classe rispetto al numero di elementi predetti della classe. Tale calcolo viene effettuato

utilizzando la seguente formula:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{5.3}$$

- *Recall*, che misura gli elementi correttamente classificati di ogni classe rispetto al numero di elementi veri della classe. Tale calcolo viene effettuato utilizzando la seguente formula:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \tag{5.4}$$

La formula completa per il calcolo della F1-score è la seguente:

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.5}$$

Questa metrica risulta essere più robusta per valutare le performance quando si affrontano situazioni in cui le classi presentano uno sbilanciamento significativo.
Tenendo in considerazione ciò, la weighted-F1 score considera la F1-score per ciascuna classe separatamente e fornisce una media ponderata in base al numero di campioni per classe, offrendo così una valutazione dettagliata delle performance di ogni categoria.
La formula per il suo calcolo è la seguente:

$$\text{Weighted-F1 Score} = \sum_{i=1}^{N} \left(\frac{\omega_i \cdot F1_{class_i}}{\sum_{i=1}^{N} \omega_i} \right) \tag{5.6}$$

dove N è il numero di classi del dataset, ω_i è il numero di elementi associati alla classe i e $F1_{class_i}$ è l'F1-score calcolata per la classe i.

5.4 Scelta dei parametri

La metodologia descritta nel Capitolo 3 prevede l'uso di due algoritmi che richiedono la configurazione di parametri: l'*Infrequent Inductive Miner* (IIM) [Leemans et al., 2014], impiegato per estrarre il modello di processo da un log degli eventi, e la DGCNN.

L'IIM costruisce il modello dopo aver eliminato i comportamenti meno frequenti, applicando una soglia di rumore. La soglia è stata modificata in incrementi del 10%, da 0% a 100%, come indica l'articolo [Chiorrini et al., 2023]. Per ciascun valore della soglia, è stato valutato il modello risultante in base alla sua capacità di riprodurre accuratamente i case presenti nel log degli eventi, misurata tramite la metrica di fitness. In particolare, è stato selezionato il modello con la soglia di rumore più bassa che garantisse un valore di fitness almeno del 90%.
In altre parole, tra tutte le soglie testate, è stata scelta quella che permettesse al modello di rappresentare con precisione almeno il 90% dei case nel log, minimizzando

al contempo il rumore e i comportamenti infrequenti nel modello finale e mantenendo così un adeguato livello di generalizzazione.
Per quanto riguarda i parametri della DGCNN, è stato impostato un unico 1-D Convolutional Layer, seguito da un dense layer. Il numero di neuroni ($num\_neurons$) di questi layers, che definisce quante unità di elaborazione vengono impiegate per catturare le informazioni rilevanti del grafo, sono stati impostati a 64 o 128, permettendo alla rete di apprendere rappresentazioni più o meno complesse in funzione del valore impostato.
L'algoritmo di ottimizzazione utilizzato è *ADAM* [Kingma and Ba, 2017] e la rete è stata addestrata per 100 epoche con early stopping. In alcuni casi, in base all'andamento delle performance, la rete è stata riaddestrata per un totale di 150 o 200 epoche.
Come funzione di loss è stata impiegata la *categorical cross entropy*, con una dimensione di batch di 64 e una percentuale di dropout fissa dello 0.1.
In aggiunta, sono stati variati i seguenti parametri:

- il numero di nodi selezionati dallo strato SortPooling (k): 5, 7, 9, 30, 50, 100, 150, 200;
- il numero di strati convoluzionali (n): 3, 5, 7, 9;
- il learning rate iniziale (lr): 10^{-4}, 10^{-5}.

5.4.1 Motivazioni nella scelta delle configurazioni

Si noti che, rispetto all'articolo di riferimento [Chiorrini et al., 2023], nella presente sperimentazione sono stati variati un numero maggiore di parametri, con l'aggiunta di configurazioni che prevedono valori più elevati. Questa scelta è stata dettata dalla necessità di adattare il modello alla complessità dei grafi in input, caratterizzati da dimensioni significativamente superiori rispetto a quelle considerate nello studio originario. L'aumento delle dimensioni dei grafi ha richiesto, infatti, l'impiego di una rete più capace di catturare e gestire la complessità delle interazioni tra i nodi, giustificando l'adozione di configurazioni più ampie e complesse per i parametri della rete.

Per avviare l'analisi, i parametri della rete sono stati inizialmente configurati seguendo le impostazioni suggerite nell'articolo di riferimento, con l'obiettivo di stabilire un punto di partenza solido e di confrontare le prestazioni del modello con quelle ottenute nello studio originale.
In particolare, inizialmente, sono stati scelti i seguenti valori:

- {5, 7, 9, e 30} per il numero di nodi selezionati dallo strato SortPooling (k);
- {3, 5, 7, e 9} per il numero di strati convoluzionali (n);
- {10^{-4}, 10^{-5}} per il learning rate;

- {64, 128} per il numero di neuroni.

Questa configurazione iniziale è in linea con i valori proposti nello studio originale, che comprendevano, ad esempio, 3, 5, 7, e 30 per k, e 2, 3, e 7 per n.
Tuttavia, in base alle caratteristiche specifiche della metodologia e in base alla complessità delle feature inter-case, si è deciso di modificare alcuni parametri. I nuovi valori esplorati sono stati scelti con lo scopo di migliorare l'adattamento del modello alla dimensione e alla complessità dei dati.
In particolare, l'inclusione di valori aggiuntivi per k e n come 9 è stata motivata dalla necessità di catturare una maggiore varietà di strutture nei grafi. Inoltre, l'aumento del numero di strati convoluzionali e dei nodi selezionati permette al modello di gestire meglio la complessità delle interazioni tra i nodi, ottimizzando così le prestazioni complessive.
Anche per quanto riguarda il learning rate, sono stati scelti valori pari a 10^{-4} e 10^{-5}, anziché 10^{-3} e 10^{-4}, come suggerito nello studio originario. Questa decisione è stata presa a causa della maggiore complessità dei grafi in input, caratterizzati da un numero di nodi significativamente superiore. Infatti, un learning rate più basso è stato ritenuto necessario per garantire aggiornamenti più stabili e graduali dei pesi della rete, riducendo il rischio di saltare oltre i minimi locali e migliorando la convergenza.
Per ragioni simili, in aggiunta al valore 64, è stato deciso di testare anche il valore di 128 per il numero di neuroni. Questa scelta è giustificata dal fatto che, maggiore è il numero di neuroni, maggiore è la capacità della rete di rappresentare e apprendere caratteristiche più complesse e dettagliate dei dati. Questo è particolarmente utile in questo caso, dove il dataset contiene strutture intricate e variazioni nei dati che dipendono dalle feature inter-case e che richiedono una maggiore capacità di gestire e processare queste informazioni in modo più efficace.
Per analizzare come influiscono questi parametri sulle performance del modello, sono stati realizzati diversi grafici che mostrano l'andamento dell'accuratezza e della weighted F1-score sul set di test in funzione della variazione delle configurazioni.

Analizzando il grafico di Figura 5.7 che rappresenta l'accuratezza media e la weighted F1-score media in funzione del parametro k, che indica il numero di nodi considerati nel sort pooling della DGCNN, è possibile osservare alcuni trend significativi. Poiché i prefissi attivi contengono moltissimi nodi, la scelta del valore di k diventa cruciale per il bilanciamento tra la capacità del modello di apprendere informazioni rilevanti e la possibilità di includere nodi irrilevanti o ridondanti.
Come si osserva dal grafico, quando k è basso, il modello esclude una parte significativa delle informazioni disponibili e non è in grado di rappresentare efficacemente le complesse interazioni tra i nodi, il che porta a performance subottimali, sia in termini di accuratezza che di weighted F1-score. Aumentando il valore di k, invece, il modello ha accesso a un numero maggiore di nodi, il che gli permette di catturare una

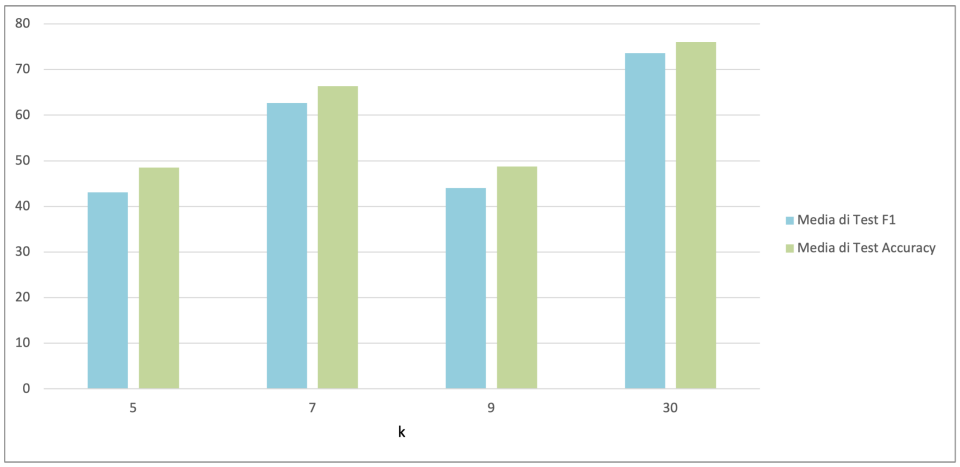


Figura 5.7: Accuratezza e weighted F1-score medie in funzione del parametro k della rete

porzione più ampia e significativa del grafo. Tale scelta è particolarmente importante nel contesto delle feature inter-case, dove l'ampiezza e la complessità dei prefissi attivi richiedono un k elevato per poter sfruttare appieno le informazioni contenute nei nodi. Di conseguenza, le performance migliorano con l'incremento di k, poiché il modello riesce a catturare sia le caratteristiche locali che globali, necessarie per una rappresentazione accurata dei grafi.

In Figura 5.8 è mostrato l'andamento dell'accuratezza e della weighted F1-score in funzione del numero di layers del modello DGCNN. Il grafico evidenzia che l'aumento del numero di strati influisce positivamente sulle performance. Questo comportamento può essere spiegato considerando le caratteristiche strutturali e la complessità dei grafi in input. Con un numero limitato di layers, probabilmente, il modello non ha

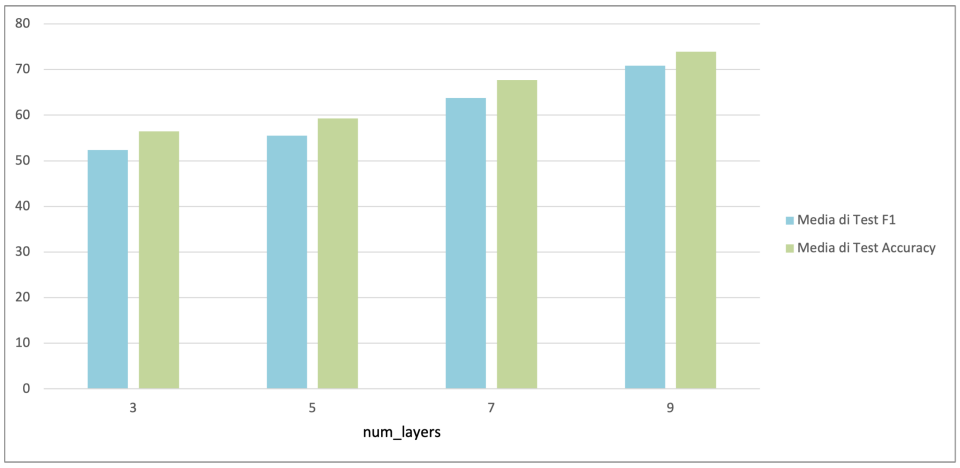


Figura 5.8: Accuratezza e weighted F1-score medie in funzione del parametro n_layers della rete

sufficiente profondità per catturare tutte le caratteristiche strutturali complesse dei grafi. Infatti, poiché i prefissi attivi sono composti da molti nodi e presentano una struttura intricata, un modello superficiale e con pochi layers non riesce a estrarre le informazioni necessarie per una rappresentazione accurata. Questo si traduce in una limitata capacità di generalizzazione, con performance inferiori in termini di accuratezza e F1 score.
Al contrario, strati più profondi permettono alla DGCNN di esplorare meglio le connessioni tra i nodi nei grafi, raccogliendo informazioni non solo dai nodi immediatamente vicini ma anche da quelli più distanti. Questo è particolarmente importante per rappresentare correttamente i grafi ampi derivanti dai prefissi attivi, che contengono una vasta gamma di interazioni tra i nodi. L'aumento del numero di layers consente quindi al modello di catturare pattern più complessi e di creare rappresentazioni più efficaci, il che si traduce in un miglioramento delle performance, come evidenziato dall'incremento dell'accuratezza e dell'F1 score nel grafico.

Tuttavia, un aspetto importante da considerare è che k deve sempre essere maggiore o uguale al numero di strati (n_layers) presenti nel modello. Questo vincolo nasce dal fatto che ogni strato aggiunge un livello di astrazione, rappresentando l'informazione raccolta dai nodi e dalle loro connessioni a livelli sempre più complessi. Se k fosse minore di n_layers, il modello non avrebbe un numero sufficiente di nodi per applicare la fase di pooling correttamente, riducendo di conseguenza la capacità di aggregare informazioni significative dal grafo. Se il numero di nodi selezionati nel sort pooling è minore del numero di layers, si rischia di perdere informazioni fondamentali, riducendo troppo il grafo. Inoltre, ogni layer della rete serve a catturare diverse caratteristiche dei dati e, se i nodi sono troppo pochi, la capacità della rete di rappresentare il grafo originario in modo utile per il task di classificazione potrebbe essere compromessa.
Considerando quanto detto, appare evidente che l'aumento del numero di strati nel modello impone un vincolo sempre più severo sui valori ammissibili di k. Per esempio, quando $n\_layers = 9$, il parametro k dovrà assumere valori pari o superiori a 9. Pertanto, qualsiasi configurazione in cui $k < 9$ sarà automaticamente esclusa, poiché non rispetta la condizione minima necessaria per una fase di pooling efficace.
Questo signfica che, all'aumentare del parametro n_layers, il grafico in Figura 5.8 mostra l'accuratezza e la weighted F1-score medie calcolate su un numero ridotto di combinazioni di parametri. Tale aspetto potrebbe introdurre un bias nella valutazione complessiva, poiché la media viene calcolata su una base di risultati più limitata.
Questo aspetto introduce l'importanza di valutare le performance combinando le configurazioni di k e n_layers come mostrato in Figura 5.9. Dal grafico si evince come il vincolo imposto dalla relazione tra k e n_layers influisca sulla scelta ottimale di questi parametri. Infatti, le combinazioni che non rispettano il vincolo $k \geq n\_layers$ vengono automaticamente escluse, e ciò si riflette nelle performance ottenute. In particolare, si noti come nel grafico di Figura 5.9, $n\_layers = 9$ non è sempre il

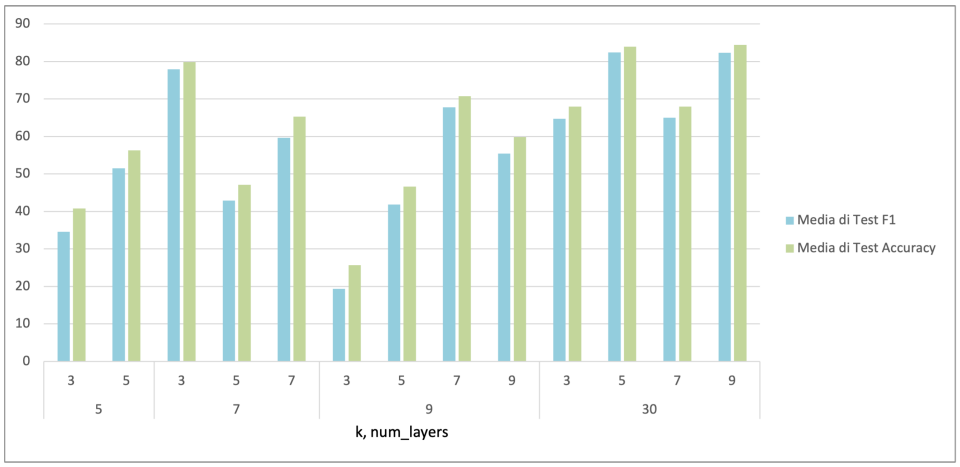


Figura 5.9: Accuratezza e weighted F1-score medie in funzione dei parametri n_layers e k della rete

parametro che garantisce le migliori performance, contrariamente a quanto si osserva nel grafico di Figura 5.8 dove le performance ottimali si registrano unicamente con $n\_layers = 9$. Questo suggerisce che l'interazione tra k e n_layers svolge un ruolo cruciale nell'ottimizzazione del modello.
Ciò significa che non basta fissare n_layers a 9 per ottenere le migliori performance. Infatti, la scelta di k deve essere attentamente calibrata in base alla struttura del grafo e alla profondità del modello, poiché la complessità delle interazioni tra i nodi gioca un ruolo decisivo.
Al contrario, si noti come la configurazione con $k = 30$ garantisce performance più alte per quasi tutti i valori di n_layers. Questo potrebbe indicare che la rete ha bisogno di selezionare più nodi nello strato di sort pooling per mantenere più informazioni sui nodi dei prefissi attivi.

Durante l'ottimizzazione dei parametri per la Deep Graph Convolutional Network (DGCNN), la combinazione del learning rate con il valore di k e il numero di layers ha dimostrato di influenzare significativamente i risultati come emerge in Figura 5.10. In alcuni casi, un learning rate più basso ha portato a risultati migliori, grazie a un apprendimento più stabile e controllato, che ha permesso una migliore convergenza verso soluzioni ottimali. Tuttavia, in altre situazioni, un learning rate più alto ha prodotto performance superiori, facilitando un apprendimento più veloce.
Questa variabilità suggerisce che non esiste una regola fissa per la scelta del learning rate ideale, ma piuttosto è fondamentale adottare una strategia di tuning dinamica che consideri la combinazione di questi parametri. Pertanto, è risultato consigliabile mantenere un approccio flessibile e utilizzare entrambi i learning rate per ottimizzare il modello.

Un'analisi simile emerge anche per il parametro $num\_neurons$ come mostrato nel

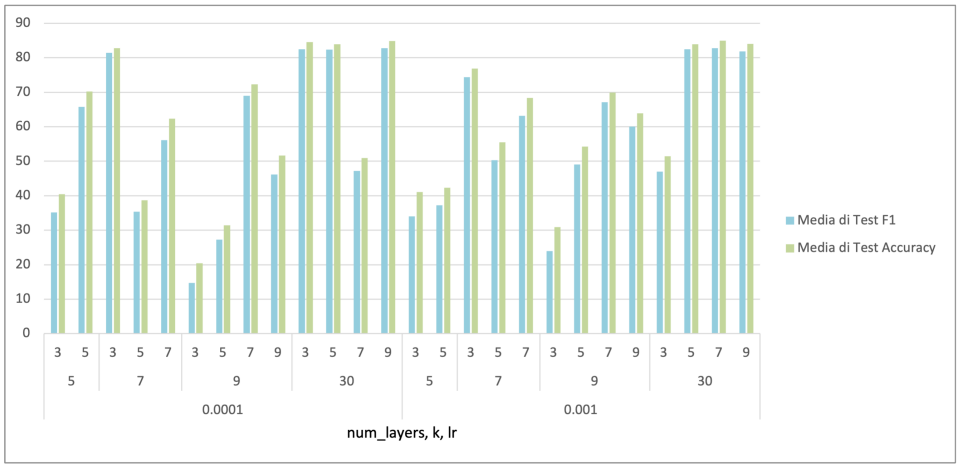


Figura 5.10: Accuratezza e weighted F1-score medie in funzione dei parametri *n_layers*, *k* e *lr* della rete

grafico di Figura 5.11. Infatti, la combinazione di *num_neurons* con il valore di *k* e il numero di layer influisce variabilmente sui risultati del modello. In alcuni casi, l'utilizzo di $num\_neurons = 64$ ha portato a performance superiori, mentre, in altri casi, $num\_neurons = 128$ ha prodotto risultati migliori. Nonostante questa variabilità, l'analisi generale mostra che $num\_neurons = 64$ tende a garantire performance più elevate nella maggior parte dei casi.
Questa osservazione suggerisce che il parametro *num_neurons* può influenzare i risultati in modo variabile, a seconda della combinazione con altri parametri. Di conseguenza, è vantaggioso mantenere entrambe le variazioni nelle future configurazioni in modo da valutare in modo più completo le performance del modello.

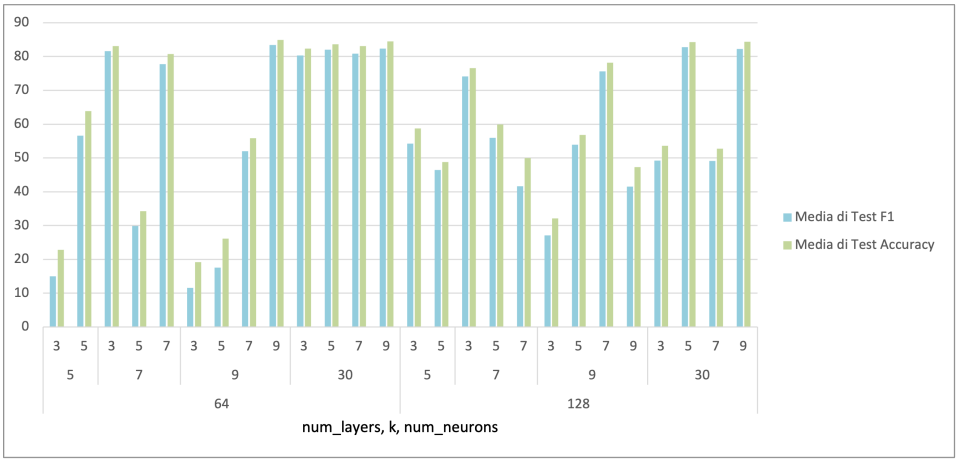


Figura 5.11: Accuratezza e weighted F1-score medie in funzione dei parametri *n_layers*, *k* e *num_neurons* della rete

Dalle valutazioni effettuate e osservando che le performance del modello migliora-

no con l'aumento del valore di k, è stato ritenuto opportuno testare valori ancora più elevati. Infatti, poichè i prefissi attivi contengono in media circa 2000 nodi, è stato deciso di incrementare ulteriormente il valore di k a 50, 100, 150 e 200, che corrispondono rispettivamente al $2,5\%$, 5%, $7,5\%$ e 10% di 2000. Questo approccio mira a verificare se una selezione più ampia di nodi nello strato SortPooling possa migliorare ulteriormente le prestazioni del modello, offrendo una rappresentazione più dettagliata e completa dei prefissi attivi massimi.
Tuttavia, osservando il grafico di Figura 5.12, i risultati mostrano che l'aumento di k non porta a miglioramenti nelle performance.

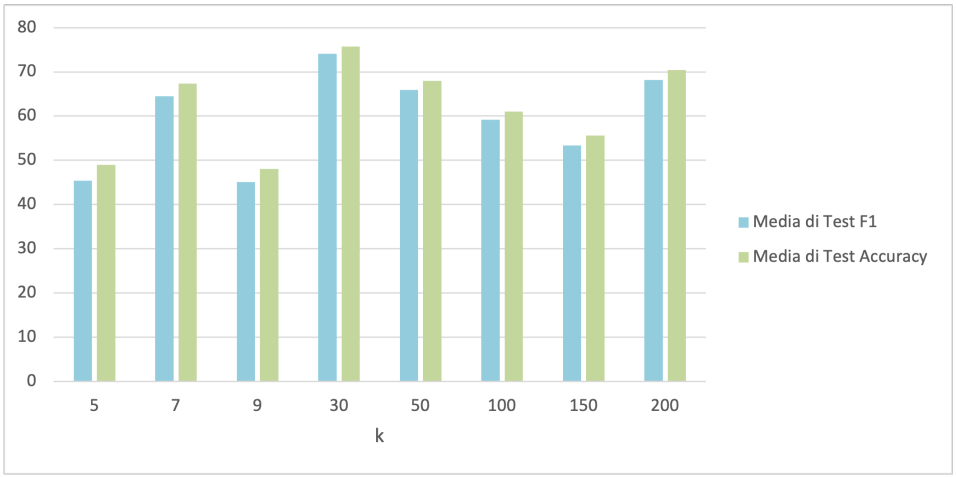


Figura 5.12: Accuratezza e weighted F1-score medie in funzione del parametro k della rete

Questo potrebbe essere dovuto al fatto che il modello avesse già raggiunto una capacità adeguata per rappresentare le caratteristiche dei dati con i valori di k testati in precedenza. Infatti, valori molto elevati di k possono comportare l'inclusione di nodi che non sono direttamente utili per il compito specifico di classificazione introducendo rumore e riducendo la qualità delle informazioni trattate dallo strato di pooling.
Inoltre, un valore di k troppo alto potrebbe causare un eccesso di complessità nel modello, il quale potrebbe imparare dettagli specifici dei dati di training e generalizzare male sui dati di test.
In definitiva, sembra che il valore $k = 30$ permetta al modello di sfruttare al meglio la complessità dei grafi, bilanciando l'inclusione di informazioni utili senza aggiungere troppo rumore.

Anche in questo caso, è utile valutare come i parametri si influenzano tra loro per comprendere le dinamiche del modello.
Come mostrato in Figura 5.13, quando si sceglie un valore molto elevato per k, come $k = 200$, le prestazioni del modello tendono a essere ottimali con un numero maggiore

di layer (come $n = 9$). Questo potrebbe accadere perché un valore elevato di k arricchisce la rappresentazione globale del grafo, offrendo al modello una quantità abbondante di informazioni sui vicini. Con più layer, dunque, il modello può apprendere e sfruttare queste informazioni dettagliate in modo più efficace.

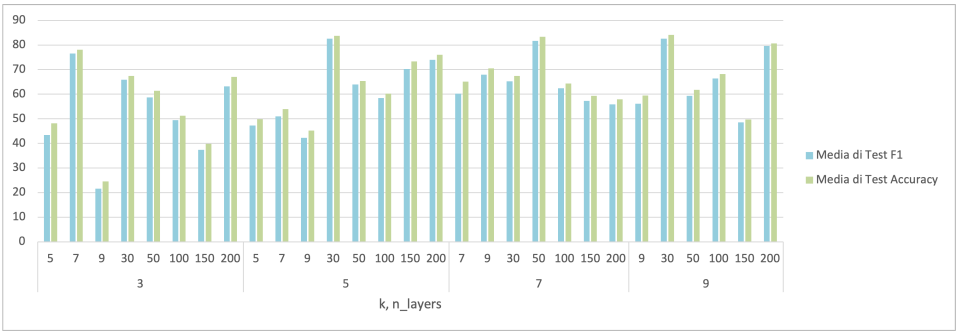


Figura 5.13: Accuratezza e weighted F1-score medie in funzione del parametro k della rete

D'altro canto, incrementando k, può essere utile considerare di ridurre il numero di layer per evitare un eccessivo aumento della complessità del modello. Infatti, con un valore maggiore di k, il modello è già in grado di catturare informazioni più ricche e potrebbe non essere necessario un numero eccessivo di layer per ottenere buone rappresentazioni. Ad esempio, prestazioni buone sono state ottenute con $k = 30$ e $n = 9$, così come con $k = 50$ e $n = 7$, suggerendo che con valori più alti di k potrebbe non essere indispensabile aumentare il numero di layer in modo proporzionale.

5.5 Risultati

Nell'approccio che considera le feature inter-case, le performance migliori sono state ottenute con la combinazione dei parametri $k = 30$, $n\_layers = 9$, $lr = 10^{-5}$ e $num\_neurons = 128$.
I risultati ottenuti con questa configurazione, confrontati con la metodologia *Multi-BIG-DGCNN* proposta da [Chiorrini et al., 2023], sono dettagliati nella Tabella 5.3. Per la metodologia *Multi-BIG-DGCNN*, i parametri ottimali sono $k = 7$, $n\_layers = 3$, $lr = 10^{-3}$ e $num\_neurons = 64$. I risultati evidenziano che l'approccio inter-case riesce a migliorare la capacità di classificazione della Next Activity, con un impatto sulla weighted F1-score, che passa da $83,90\%$ a $84,16\%$.
Questo aumento suggerisce che il modello, considerando anche le relazioni tra le tracce, riesce a bilanciare meglio le performance tra le diverse categorie, migliorando la sua capacità di identificare correttamente anche quelle meno rappresentative. Ovvero, una weighted F1-score più alta, suggerisce non solo che il modello sta dimostrando di performare bene sulle classi maggioritare, ma che sta anche evitando di trascurare

quelle minoritarie.

Metrica	Metodologia	Risultato
F1-score weighted	Multi-BIG-DGCNN	83.90%
	Multi-inter-case-BIG-DGCNN	**84.16%**

Tabella 5.3: Risultati

Inoltre, è importante notare la differenza significativa rispetto ai parametri ottimali nei due approcci.
In particolare, l'aumento del parametro k, da 7 a 30, riflette la necessità di gestire grafi con una maggiore complessità strutturale dovuta all'aggiunta delle feature inter-case. Come emerge osservando la Tabella 5.1, la media della lunghezza dei prefissi è 9. In questo contesto, il valore di 7 suggerito nell'articolo [Chiorrini et al., 2023] risulta relativamente vicino alla lunghezza media dei prefissi, indicando una certa coerenza con le caratteristiche dei dati.
D'altra parte, il parametro $k = 30$ eccede la lunghezza della traccia più estesa che è pari a 21. Questo implica che, con un valore così elevato di k, si include inevitabilmente un'ampia gamma di nodi, tra cui quelli relativi ai prefissi attivi. Le feature inter-case sono numerose e complesse, pertanto, durante i vari stadi di convoluzione, filtraggio e ordinamento dei nodi, è probabile che nei 30 nodi selezionati emergano le caratteristiche più rilevanti e significative relative ai prefissi attivi. Questo processo di selezione aiuta a focalizzarsi sugli aspetti più importanti e ridurre l'influenza del rumore e delle feature meno rilevanti, migliorando così l'efficacia e la robustezza del modello.
Per quanto riguarda il numero di layers (n_layers), il metodo *Multi-BIG-DGCNN* utilizza 3 layer, mentre l'approccio *Multi-inter-case-BIG-DGCNN* ne impiega 9. Questo incremento riflette una maggiore complessità nel modellare le relazioni tra le feature inter-case, consentendo al modello di apprendere rappresentazioni più sofisticate e approfondite delle relazioni tra le tracce.
Un'altra differenza significativa riguarda il learning rate (lr). Per *Multi-BIG-DGCNN*, il tasso di apprendimento ottimale è 10^{-3}, mentre per *Multi-inter-case-BIG-DGCNN* è 10^{-5}. Questo decremento significativo indica la necessità di una maggiore precisione e stabilità nella fase di ottimizzazione quando si lavora con un set di dati più complesso, come quello che include le feature inter-case. Infatti, un valore più basso di lr può aiutare a prevenire aggiornamenti eccessivi dei pesi e a migliorare la convergenza del modello durante l'addestramento.
Infine, il numero di neuroni per layer ($num\_neurons$) varia tra i due approcci: 64 per *Multi-BIG-DGCNN* e 128 per *Multi-inter-case-BIG-DGCNN*. Questo incremento consente al modello di *Multi-inter-case-BIG-DGCNN* di gestire un numero maggiore

di feature e di rappresentare in modo più dettagliato le informazioni derivanti dalle tracce.
Queste differenze nei parametri non solo riflettono le esigenze specifiche dei due approcci, ma indicano anche come l'approccio inter-case richieda una configurazione di modello più complessa e adattata alla natura dei dati.

5.5.1 Analisi dei risultati per classe

In Figura 5.14 è rappresentata la matrice di confusione che mostra le prestazioni di un modello di classificazione alla luce delle relazioni tra le classi predette (colonne della matrice) e le classi effettive (righe della matrice). Le celle lungo la diagonale principale, che va da in alto a sinistra verso il basso a destra, mostrano le predizioni corrette.

Dall'analisi della matrice, emerge come alcune attività, come *END*, *RequestPayement*, *PayementHandled*, *RequestForPayementAPPROVEDbyADMINISTRATOR*, *PermitAPPROVEDbyADMINISTRATOR*, *RequestForPayementREJECTEDbyPRE-APPROVER* e *PermitSUBMITTEDByEMPLOYEE* mostrano valori elevati lungo la diagonale, suggerendo che il modello riesce a predire queste classi con buona accuratezza.
Tuttavia, dall'analisi del log e del modello di processo riportato in fondo al capitolo, si osserva che, ad esempio, la classe *PermitSUBMITTEDByEMPLOYEE* è facilmente riconoscibile, poiché segue quasi sempre l'attività *START*. Questa correlazione chiara tra le due attività aiuta il modello a identificarla con maggiore precisione, contribuendo così a un'accuratezza elevata nella previsione di questa classe.

Le classi *PermitAPPROVEDbyBUDGETOWNER* e *PermitFINAL-APPROVED bySUPERVISOR* tendono a confondersi a vicenda e questo fenomeno può essere spiegato osservando la rete di Petri, che mostra come questi eventi possano verificarsi in parallelo. Dunque, la loro sovrapposizione temporale e la loro similarità possono contribuire alla difficoltà del modello nel distinguerle correttamente.

La classe *RequestForPaymentSUBMITTEDbyEMPLOYEE*, invece, viene spesso classificata come *PermitSUBMITTEDbyEMPLOYEE* o *PermitFINAL-APPROVEDby SUPERVISOR*. Questa confusione si spiega considerando le transazioni fantasma: *RequestForPaymentSUBMITTEDbyEMPLOYEE* può talvolta apparire come una delle prime attività dopo *Start*, sostituendo le altre due. Questa sovrapposizione temporale e contestuale contribuisce alla difficoltà del modello nel discriminare tra queste attività.

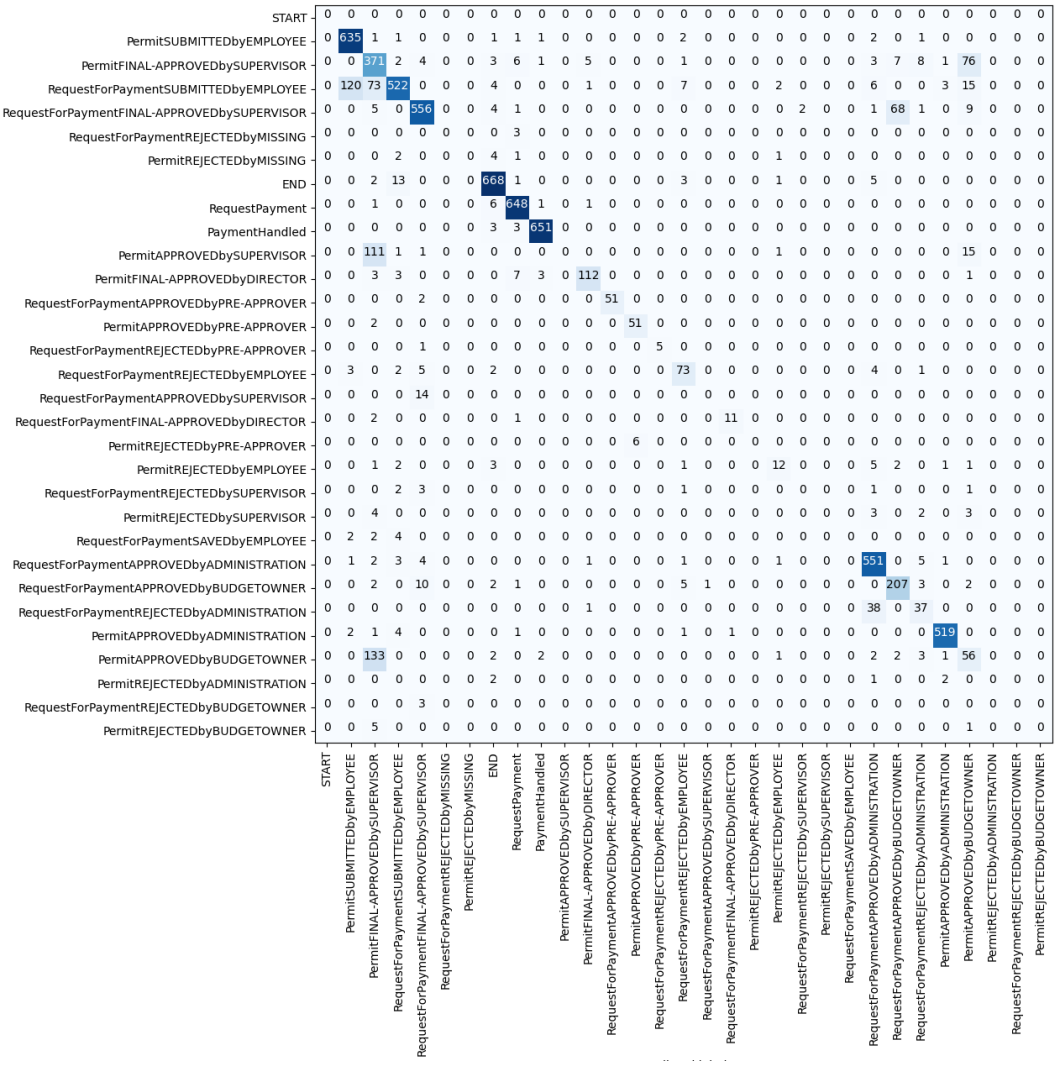


Figura 5.14: Matrice di confusione

Inoltre, classi *PermitAPPROVEDByBUDGETOWNER* e *PermitApprovedBySUPERVISOR* sono spesso confuse con la classe *PermitFINAL-APPROVEDBySUPERVISOR* probabilmente a causa di vari motivi legati alla loro similarità semantica e comportamentale. Infatti, come emerge dall'analisi del log e della rete di Petri, queste attività rappresentano fasi vicine all'interno del flusso di processo, aumentando la probabilità di confusione.

Si noti anche come la classe *PermitFINAL-APPROVEDBySUPERVISOR* sia rappresentata in modo preponderante nel dataset. In questo modo, il modello tenderà a predire con maggiore frequenza questa classe, confondendola con le altre due fasi intermedie di approvazione. In generale, la differenza funzionale tra un'approvazione intermedia e una finale, dal punto di vista del processo, può risultare molto sottile. Questa sfumatura, se non chiaramente rappresentata nei dati, complica ulteriormente la capacità del modello di riconoscere il passaggio tra le attività, portando ad un'alta

probabilità di errore nel distinguere tra queste attività simili.
Un ragionamento analogo vale per la classe *RequestForPaymentREJECTEDbyADMINISTRATION*, che spesso viene confusa con *RequestForPaymentAPPROVEDbyADMINISTRATION*. Queste due attività fanno parte di una fase decisionale simile ma con esiti differenti, dunque, i segnali distintivi necessari per una corretta classificazione possono risultare sottili e facilmente confusi se non catturati in modo adeguato nei dati di input o nelle caratteristiche utilizzate dal modello.
Inoltre, ancora una volta, *RequestForPaymentAPPROVEDbyADMINISTRATION* è rappresentata più frequentemente rispetto all'altra, per cui, il modello tenderà a predirla più spesso, portando a una sovrastima delle approvazioni a scapito delle previsioni corrette per i rifiuti.

Dal punto di vista interpretativo, le attività che mostrano valori elevati lungo la diagonale indicano le classi in cui il modello ottiene le prestazioni migliori. Sebbene questo fenomeno si riscontri spesso nelle classi maggioritarie, ci sono eccezioni, come nel caso di *RequestForPaymentFINAL-APPROVEDbySUPERVISOR* e *RequestForPaymentREJECTEDbyPRE-APPROVER*, dove il modello riesce comunque a performare bene nonostante la loro minore frequenza.

In definitiva, la presenza di errori nelle predizioni tra attività simili, come approvazioni e rifiuti da parte di diversi ruoli, evidenzia la facilità con cui il modello confonde attività che possono apparire simili nella struttura o nel contenuto. Queste problematiche, legate alla similarità semantica e di ruolo, alla sovrapposizione temporale e alla mancanza di caratteristiche chiare nel dataset, suggeriscono la necessità di affinare ulteriormente il modello. Migliorare la definizione delle caratteristiche e fornire informazioni di contesto più dettagliate, ad esempio sulle risorse che eseguono le varie attività, potrebbe contribuire a migliorare la capacità di distinguere tra le varie classi e ridurre le confusioni.

Per capire meglio questo aspetto, si osservi la matrice di confusione di Figura 5.15 che mostra i risultati raggruppati per tipologia di attività.
Come è interessante notare, le diverse tipologie di attività, come le approvazioni e i rifiuti, sono categorizzate in modo efficace, con una chiara distinzione tra i vari gruppi.
L'analisi mostra che, al massimo, ci può essere una certa confusione tra le attività *PermitFINAL-APPROVED* e *PermitAPPROVED*, così come tra *RequestForPaymentSUBMITTED* e *PermitSUBMITTED*. Tuttavia, queste ultime rappresentano comunque delle approvazioni e delle sottomissioni, rientrando pertanto in categorie simili.

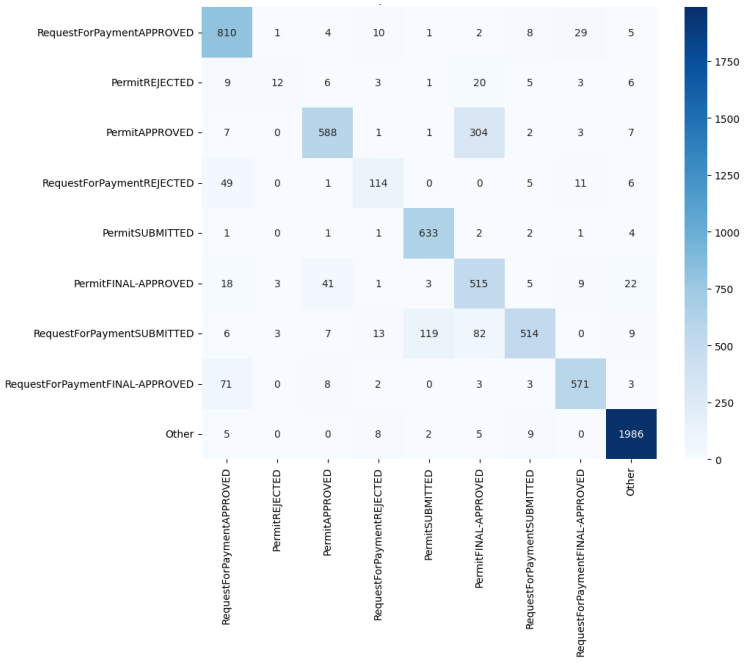
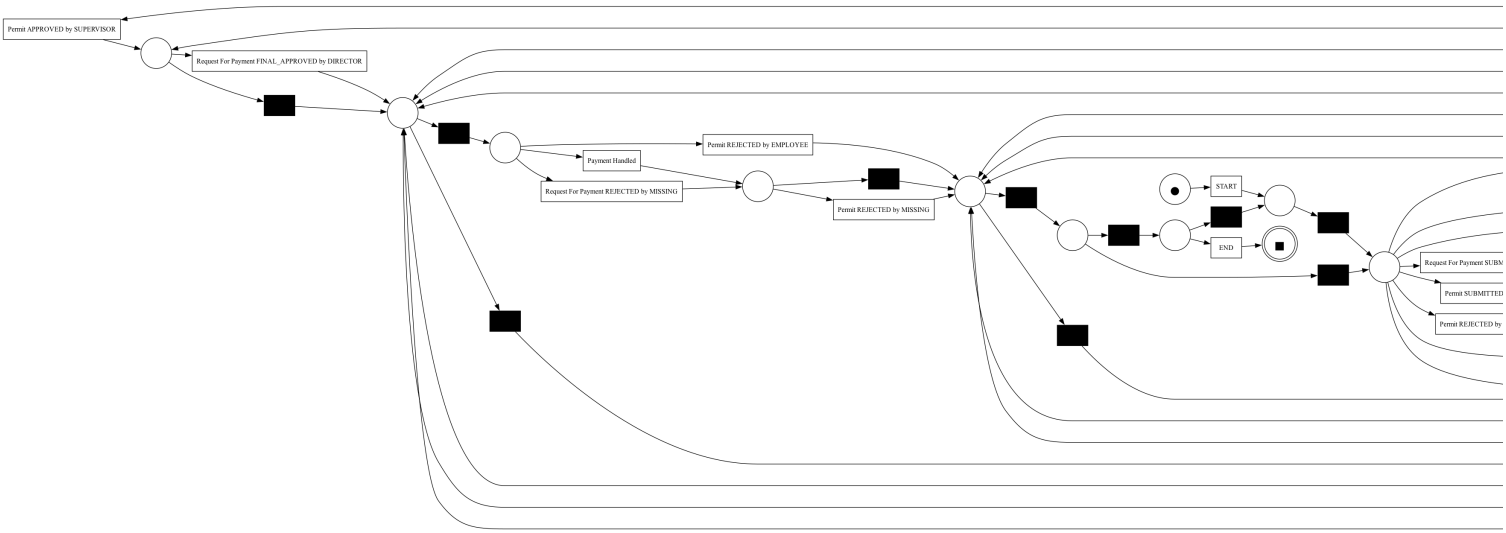


Figura 5.15: Matrice di confusione raggruppata per tipologia di attività

Analizzando invece la matrice di confusione raggruppata per ruoli mostrata in Figura 5.16, si nota che la classificazione non risulta sempre chiara. Le attività associabili a ciascun ruolo mostrano una certa confusione, evidenziando che le distinzioni tra le diverse categorie non sono sempre nette. Gli unici ruoli che emergono come ben classificati sono *ADMINISTRATOR* e *DIRECTOR*, probabilmente perché svolgono funzioni privilegiate all'interno del processo.
Ciò suggerisce che la classificazione potrebbe essere migliorata, rendendo necessario un approfondimento delle caratteristiche distintive di ciascun ruolo. La mancanza di chiarezza nella classificazione evidenzia la complessità dei processi analizzati e la necessità di ulteriori raffinamenti nel modello.


Permit APPROVED by SUPERVISOR
Request For Payment FINAL_APPROVED by DIRECTOR
Payment Handled
Permit REJECTED by EMPLOYEE
Request For Payment REJECTED by MISSING
Permit REJECTED by MISSING
START
END
Request For Payment SUBM
Permit SUBMITTED
Permit REJECTED by

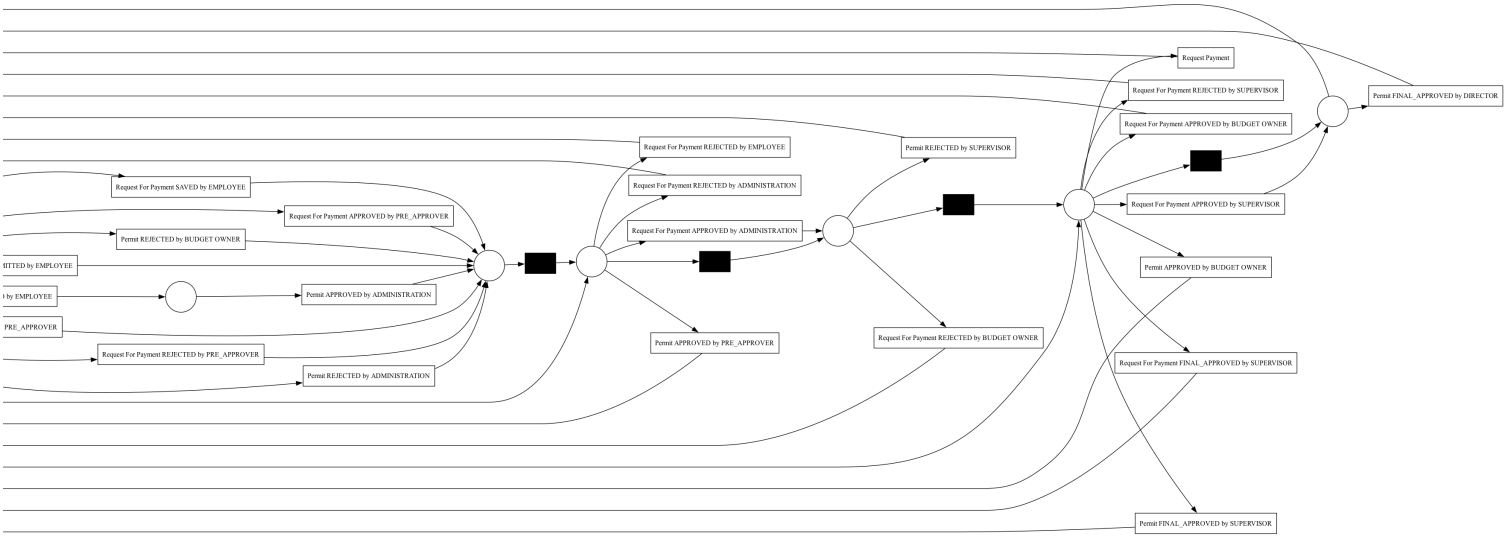

Request Payment
Request For Payment REJECTED by SUPERVISOR
Request For Payment APPROVED by BUDGET OWNER
Permit FINAL_APPROVED by DIRECTOR
Request For Payment REJECTED by EMPLOYEE
Permit REJECTED by SUPERVISOR
Request For Payment REJECTED by ADMINISTRATION
Request For Payment SAVED by EMPLOYEE
Request For Payment APPROVED by PRE_APPROVER
Request For Payment APPROVED by ADMINISTRATION
Request For Payment APPROVED by SUPERVISOR
Permit REJECTED by BUDGET OWNER
Permit APPROVED by BUDGET OWNER
Permit APPROVED by ADMINISTRATION
Request For Payment REJECTED by BUDGET OWNER
Permit APPROVED by PRE_APPROVER
Request For Payment REJECTED by PRE_APPROVER
Request For Payment FINAL_APPROVED by SUPERVISOR
Permit REJECTED by ADMINISTRATION
Permit FINAL_APPROVED by SUPERVISOR

Figura 5.16: Matrice di confusione raggruppata per tipologia di risorse

5.5.2 Analisi dei risultati per lunghezza di prefisso

Dopo aver analizzato le performance per classe, risulta altrettanto utile considerare l'analisi dei risultati per prefisso in modo da comprendere come le prestazioni del modello cambino in base alla quantità di informazioni disponibili prima della previsione.
Osservando il grafico di Figura 5.17, si nota che le metriche rimangono piuttosto alte per gran parte delle lunghezze di prefisso. Tuttavia, si evince un calo significativo delle performance per i prefissi di lunghezza 3, nonostante vi sia ancora un numero elevato di campioni disponibili, come evidenziato dalla linea rossa tratteggiata.
Questo indica che, pur avendo una quantità sufficiente di dati per fare previsioni, il modello fatica a mantenere alte prestazioni con prefissi così corti, probabilmente a causa della mancanza di contesto informativo sufficiente per fare previsioni accurate.
Un altro elemento che emerge dall'analisi è un calo graduale nelle prestazioni, sia in termini di accuratezza che di weighted F1 score, al diminuire del numero di campioni, specialmente per prefissi di lunghezza superiore a 10. Questa leggera decrescita delle performance può essere attribuita al fatto che, con un numero ridotto di dati, il modello ha meno opportunità di generalizzare bene e quindi la sua capacità predittiva ne risente.

Foccalizzandosi sull'analisi della matrice di confusione relativa ai prefissi di lunghezza 3 (Figura 5.18), emerge un quadro interessante. Dall'esame della matrice, si nota che gran parte delle classi più difficili da predire nella matrice complessiva (Figura 5.14) appartengono quasi interamente a questa categoria. Questo suggerisce che le difficoltà nella classificazione sono particolarmente pronunciate nelle fasi iniziali del processo. In effetti, come evidenziato anche dalla rete di Petri, all'inizio il processo presenta

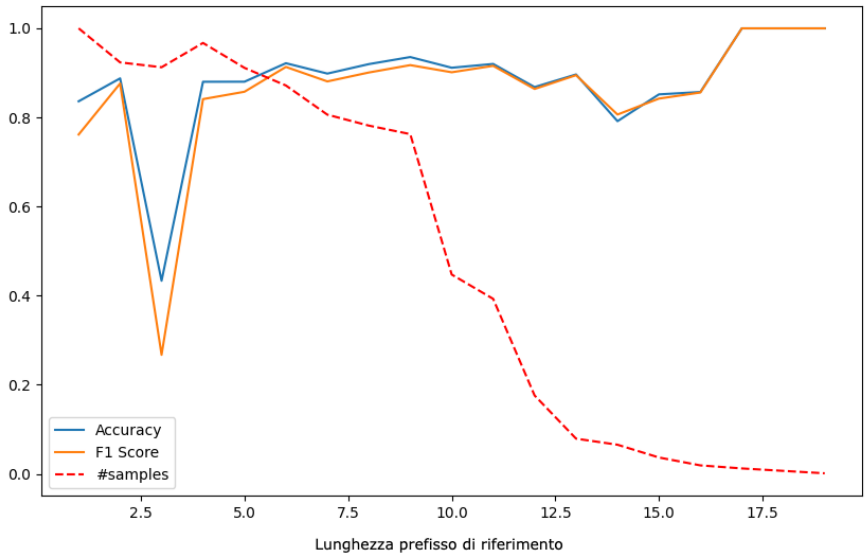


Figura 5.17: Accuratezza e weighted F1-score al variare della lunghezza del prefisso

un numero molto elevato di attività concorrenti che aumentano la complessità di predizione a scapito delle classi minoritarie.
Gli Instance Graph associati a queste fasi mostrano una struttura intricata, caratterizzata da numerosi parallelismi tra una o più attività. Questi parallelismi rendono difficile per il modello riconoscere chiaramente le transizioni tra le varie classi, poiché molteplici eventi possono verificarsi simultaneamente o in rapida successione. Tale complessità si riflette nelle prestazioni del modello, che fatica a distinguere accuratamente le attività concorrenti nelle prime fasi del processo.

In Figura 5.19 e 5.20 sono mostrate rispettivamente le distribuzione delle classi per lunghezza del prefisso e le predizioni effettuate in base a questa lunghezza. Nella prima figura, si nota che, per i prefissi di lunghezza 1, la maggior parte delle classi si concentra su un numero molto ristretto di categorie, con una predominanza netta di una classe specifica, che evidenzia un comportamento fortemente sbilanciato. Questa concentrazione è visibile dalla colorazione più intensa nella matrice per quella lunghezza.

La seconda figura, che rappresenta le predizioni, mostra una distribuzione simile: poiché nel train gli elementi sono concentrati quasi esclusivamente su una delle classi maggioritarie, il modello tende a confondere gli altri esempi predicendo principalmente quella classe, portando a predizioni ripetitive e poco diversificate.
Questo evidenzia ancora una volta la mancanza di chiarezza nelle fasi iniziali degli instance graph, dove le attività possono diramarsi e intrecciarsi in modo complesso. Nelle prime fasi, infatti, le informazioni sono meno definite e le attività possono prendere direzioni diverse, rendendo difficile per il modello distinguere tra le diverse classi.

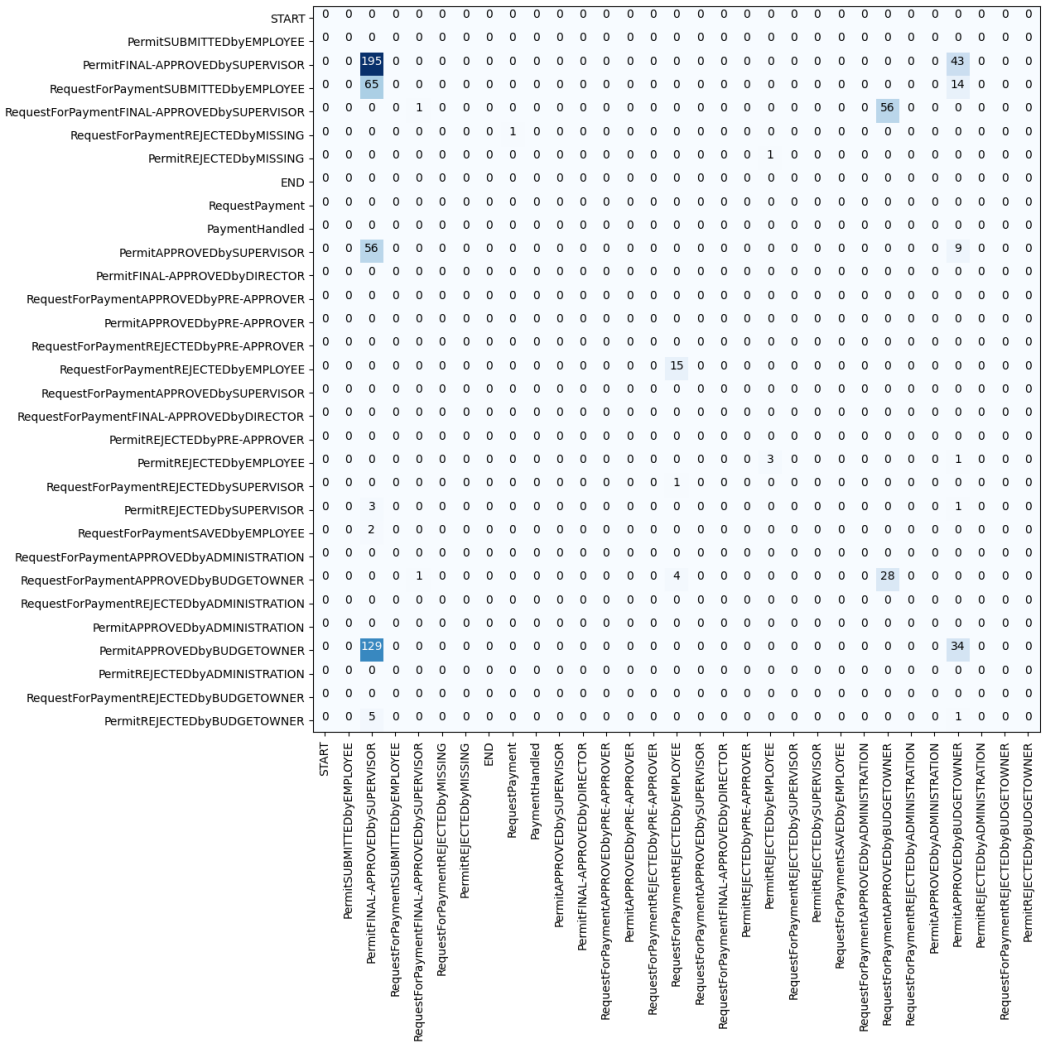


Figura 5.18: Matrice di confusione per i prefissi di lunghezza 3

Tale fenomeno si riflette anche nei prefissi di lunghezza 3 come se gli errori iniziali generati dalle predizioni sui prefissi di lunghezza 1 influenzano il resto del processo predittivo. Infatti, il modello continua a commettere errori anche nei prefissi successivi. La difficoltà nel distinguere chiaramente tra attività simili o intrecciate nelle fasi iniziali si propaga, riducendo ulteriormente la precisione delle predizioni man mano che i prefissi diventano più lunghi, fino a che il flusso delle attività non risulta più definito.

Quando il processo si trova nelle prime fasi, il modello fatica a catturare correttamente la relazione tra le varie attività. In questa fase, le informazioni sono ancora troppo ambigue e le attività possono prendere direzioni diverse, rendendo difficile per il modello identificare chiaramente quale classe predire.

In questa fase, inoltre, le feature inter-case e i prefissi attivi possono contribuire a creare ulteriore confusione, complicando ulteriormente il processo predittivo.

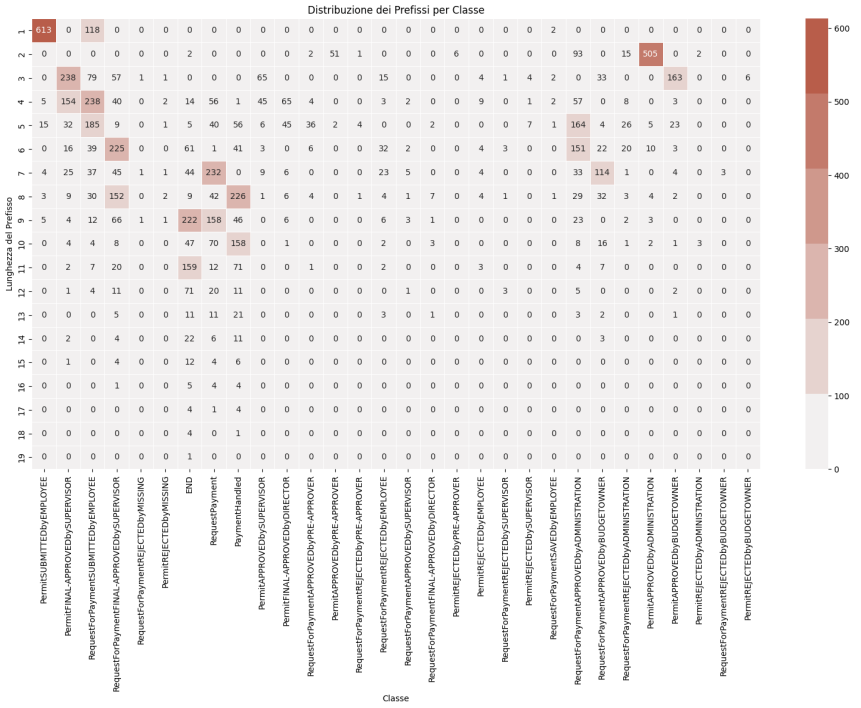


Figura 5.19: Distribuzione degli elementi di test per classe e per lunghezza di prefisso

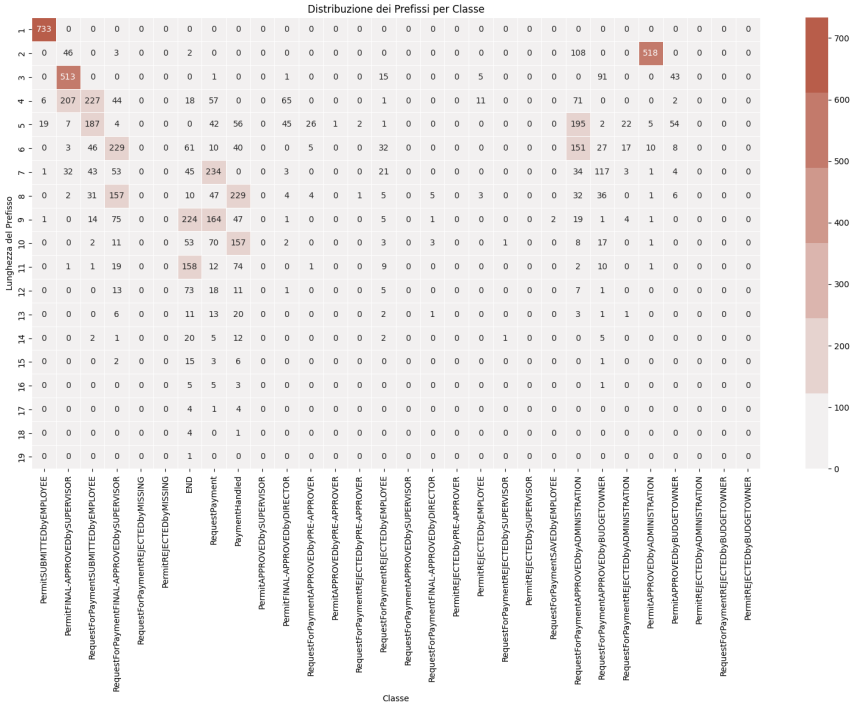


Figura 5.20: Predizione degli elementi di test per classe e per lunghezza di prefisso

In realtà, anche nell'articolo [Chiorrini et al., 2023] si osserva un andamento simile (Figura 5.21) dove le prime fasi del processo risultano particolarmente difficili da

predire e modellare correttamente con un impatto negativo soprattutto, ancora una volta, sui prefissi di lunghezza 3.

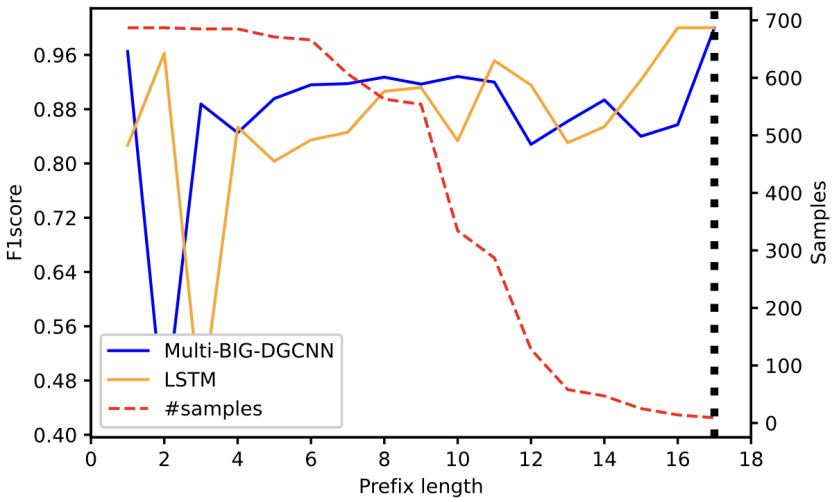


Figura 5.21: Accuratezza e weighted F1-score in funzione della lunghezza del prefisso, secondo i risultati riportati in [Chiorrini et al., 2023]

L'analisi delle prestazioni, mostrata in Figura 5.17 e 5.21, evidenzia una coerenza nei risultati, ma con importanti differenze. Nelle aree in cui le performance sono elevate nel secondo grafico, si osservano valori ancora più alti nel primo, indicando un potenziamento delle capacità predittive del modello. Tuttavia, dove le performance sono più basse nel secondo caso, si registra un ulteriore abbassamento nel primo.
Questa situazione può essere spiegata attraverso le caratteristiche delle feature inter-case. Quando l'interazione tra le attività è chiara, l'inclusione di prefissi che operano in parallelo al contesto di riferimento può migliorare le prestazioni. Al contrario, nei casi in cui le interazioni sono più complesse, come nel caso dei prefissi di lunghezza 3, queste feature possono introdurre rumore, compromettendo l'accuratezza delle predizioni. Pertanto, pur mantenendo una coerenza nei risultati, la struttura delle attività e la loro interazione influenzano in modo significativo l'efficacia delle feature inter-case.
In aggiunta, la coerenza nell'andamento dei risultati potrebbe essere in parte attribuita alla definizione di prefisso (6), la quale etichetta la prossima attività come la successiva in ordine numerico all'interno dell'instance graph.
Tale approccio ignora i parallelismi esistenti tra i nodi, riducendo la complessità delle interazioni che si possono verificare nel flusso delle operazioni. In particolare, questa definizione implica che le attività siano trattate come sequenziali, quando in realtà possono presentarsi in modo parallelo, rendendo difficile per il modello cogliere le relazioni dinamiche e interattive tra di esse.
Tale limitazione può influenzare negativamente la capacità del modello di rappresentare in modo accurato la realtà del processo, poiché non considera le variazioni di

comportamento che possono emergere quando più attività si svolgono simultaneamente.
Di conseguenza, l'incapacità di integrare questi parallelismi nella definizione di prefisso potrebbe contribuire nelle prime fasi, ostacolando la capacità del modello di effettuare predizioni corrette e affidabili.

Capitolo 6

Conclusioni e Sviluppi Futuri

In questa tesi è stato introdotto un approccio innovativo che fa uso di prefissi attivi e mira a sfruttare le caratteristiche inter-case per migliorare la previsione della Next Activity. I risultati ottenuti hanno evidenziato alcuni miglioramenti nelle performance del modello, sebbene non siano così marcati come inizialmente previsto.

Un elemento cruciale da considerare è che l'integrazione di feature inter-case richiede la costruzione di grafi di dimensioni considerevoli. Questo aumento dimensionale può portare al fenomeno noto come *curse of dimensionality*, che si riferisce a problematiche che emergono quando il numero di variabili in un dataset aumenta in modo significativo.
In questo contesto, l'aumento dei nodi e delle connessioni rende il modello più complesso e richiede una quantità maggiore di dati per ottenere stime affidabili. Questo, a sua volta, aumenta la probabilità di introdurre rumore nei dati, compromettendo le performance predittive del modello.
In particolare, l'espansione della dimensionalità può complicare il processo di apprendimento e portare a un fenomeno noto come overfitting. In questo caso, il modello si adatta eccessivamente ai dati di addestramento, assimilando fluttuazioni e rumori che compromettono la sua capacità di fare previsioni accurate su nuovi dati.
Questo fenomeno è a maggior ragione accentuato nell'analisi di grafi arricchiti con feature inter-case, poiché l'aumento dimensionale richiederebbe una quantità esponenzialmente maggiore di dati per mantenere la qualità delle stime e garantire che il modello possa apprendere in modo efficace le relazioni significative senza cadere nel rischio di overfitting.

Un aspetto specifico di questo problema è dato dalla considerazione di prefissi attivi. Poiché i prefissi attivi rappresentano porzioni di sequenze di eventi che partono sempre dallo start, c'è il rischio di includere attività non necessariamente parallele ai prefissi di riferimento. Questa inclusione può generare ulteriore rumore, poiché tali attività possono influenzare il modello senza fornire informazioni utili per la previsione della prossima attività. In altre parole, analizzare un intero prefisso attivo può sovraccaricare il modello con dati non rilevanti, contribuendo all'overfitting.
Per affrontare questi problemi, si potrebbe adottare un approccio alternativo. Invece

di considerare i prefissi attivi nella loro interezza, sarebbe utile analizzare solo le ultime attività di ciascun prefisso attivo. Questo permetterebbe di focalizzarsi solo sulle attività più rilevanti, riducendo così la complessità e l'espansione dimensionale. Tale strategia potrebbe migliorare la capacità di generalizzazione del modello, consentendo di apprendere relazioni più significative e riducendo il rischio di catturare il rumore presente nei dati di addestramento. Inoltre, questo approccio potrebbe contribuire ad una rappresentazione più chiara delle interazioni tra attività e risorse, facilitando l'interpretazione dei risultati.

Un ulteriore potenziale sviluppo futuro del presente lavoro potrebbe concentrarsi sull'integrazione di informazioni relative alle risorse coinvolte nei processi analizzati. Questo approccio si basa sull'idea di arricchire il modello con una rappresentazione più dettagliata delle interazioni tra le risorse e le attività, al fine di migliorare la previsione della prossima attività.
Nel contesto di questa nuova proposta, si potrebbe considerare un grafo in cui, per ogni prefisso di riferimento, vengono utilizzate due tipologie di nodi: uno per rappresentare le risorse e l'altro per rappresentare le attività. Ogni nodo risorsa sarebbe collegato all'attività specifica che sta svolgendo in quel momento, creando così una rete di interazioni più complessa e informativa. Questa rappresentazione permetterebbe di catturare le dinamiche operative e i ruoli delle diverse risorse in relazione alle attività, contribuendo a una comprensione più profonda delle sequenze di eventi.
Come osservato nel capitolo dedicato agli esperimenti, la classificazione delle risorse è spesso problematica, con rischi di confusione tra diverse tipologie. Al contrario, le attività tendono a essere più facilmente classificabili e distinguibili. Pertanto, l'inclusione di un grafo che separa chiaramente risorse e attività potrebbe facilitare l'apprendimento del modello, migliorando la sua capacità di generalizzazione e riducendo l'errore di classificazione.
Tuttavia, è importante notare che questa strategia di integrazione è attuabile solo se il dataset in uso comprende informazioni dettagliate sulle risorse. Inoltre, è fondamentale che non sussista il vincolo per cui una risorsa è limitata a svolgere un'unica attività in un determinato contesto. Se tale vincolo fosse presente, l'encoding delle sole attività potrebbe rivelarsi sufficiente per la modellazione, rendendo non necessaria l'aggiunta di ulteriori complessità nel grafo.
In sintesi, l'inclusione delle informazioni sulle risorse potrebbe rappresentare un significativo passo avanti nell'ottimizzazione della previsione delle attività, offrendo una rappresentazione più ricca e informativa del processo analizzato. Questa direzione di ricerca non solo potrebbe contribuire a migliorare le performance del modello, ma anche a chiarire le interrelazioni tra risorse e attività, rendendo i risultati più interpretabili e applicabili in scenari reali.

In conclusione, sebbene l'approccio basato su prefissi attivi mostri potenzialità

significative, è fondamentale affrontare le sfide legate all'aumento della dimensionalità e alla possibile introduzione di rumore. Ciò implica la necessità di esplorare tecniche di riduzione della dimensionalità e di raffinare le strategie di pre-elaborazione dei dati, al fine di ottimizzare ulteriormente i risultati e rendere il modello più robusto e affidabile.

Bibliografia

[Aalst, 2012] Aalst, W. (2012). Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems*, 3:7.1–7.17.

[Chiorrini et al., 2023] Chiorrini, A., Diamantini, C., Genga, L., and Potena, D. (2023). Multi-perspective enriched instance graphs for next activity prediction through graph neural network. *Journal of Intelligent Information Systems*, 61(1):5–25.

[Di Francescomarino and Ghidini, 2022] Di Francescomarino, C. and Ghidini, C. (2022). *Predictive Process Monitoring*, pages 320–346. Springer International Publishing, Cham.

[Diamantini et al., 2016] Diamantini, C., Genga, L., Potena, D., and van der Aalst, W. (2016). Building instance graphs for highly variable processes. *Expert Systems with Applications*, 59:101–118.

[Efrén et al., 2023] Efrén, Vidal, Rama-Maneiro, J. C., and Lama, M. (2023). Deep learning for predictive business process monitoring: Review and benchmark. *IEEE Transactions on Services Computing*, 16(1):739–756.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[Leemans et al., 2014] Leemans, S. J. J., Fahland, D., and van der Aalst, W. M. P. (2014). Discovering block-structured process models from incomplete event logs. In Ciardo, G. and Kindler, E., editors, *Application and Theory of Petri Nets and Concurrency*, pages 91–110, Cham. Springer International Publishing.

[Scardapane et al., 2023] Scardapane, S., Silvestri, S., Zuccarello, E., Saggese, A., Piccialli, A., Uncini, A., and Vannucci, G. (2023). Tsunami: Lightweight continuous learning framework for real-time prediction of customer behavior in retail. *International Journal of Data Science and Analytics*, 12(1).

[van Dongen, 2020] van Dongen, B. (2020). Bpi challenge 2020: Prepaid travel costs. 4TU.Centre for Research Data.

[Zhang et al., 2018] Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative*

Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence, AAAI'18/IAAI'18/EAAI'18. AAAI Press.

Ringraziamenti

Questa tesi rappresenta il traguardo di un percorso accademico che mi ha arricchito non solo dal punto di vista delle conoscenze, ma anche a livello umano. Ci sono molte persone che hanno contribuito, in diversi modi, a rendere possibile questo risultato, e vorrei cogliere l'occasione per esprimere la mia sincera gratitudine.

In primo luogo, un sentito ringraziamento va al mio relatore, Domenico Potena, per la guida costante, i preziosi consigli e la pazienza dimostrata durante tutto il percorso. Grazie a lui non solo ho potuto completare questo lavoro con successo, ma mi è stata offerta l'incredibile opportunità di proseguire gli studi con il dottorato, un passo importante per la mia carriera accademica e professionale. La sua fiducia e il suo sostegno sono stati fondamentali per questo percorso.

Un ringraziamento va anche agli invidiosi e a coloro che mi dicevano che non ce l'avrei fatta. Le loro parole, sebbene inizialmente difficili da accettare, hanno avuto un effetto motivante su di me, spingendomi a dimostrare il contrario e a superare le mie stesse aspettative. Grazie a loro, ho scoperto una forza interiore che non sapevo di avere e ho imparato a trasformare le critiche in opportunità di crescita.

Vorrei inoltre ringraziare i miei colleghi e amici, che hanno reso questi anni di studio un'esperienza indimenticabile, offrendomi non solo collaborazione, ma anche supporto e incoraggiamento nei momenti critici.

Grazie alle amiche del mio gruppo perchè le chiacchierate spensierate e i momenti di svago sono stati essenziali per il mio benessere e mi hanno permesso di ricaricare le energie quando gli impegni sembravano insormontabili.

Un ringraziamento speciale va al gruppo del musical, che mi ha accolto con calore. Fin dal primo momento, mi hanno fatto sentire parte di una grande famiglia, offrendomi l'opportunità di esprimermi e di crescere in un ambiente stimolante e positivo.
Grazie a loro, ho potuto affrontare la mia timidezza e superare i miei limiti, buttandomi in nuove esperienze che non avrei mai pensato di poter affrontare. Ogni prova, ogni esibizione è stata un'occasione per scoprire nuove parti di me stessa e per costruire relazioni significative. La loro pazienza e il loro incoraggiamento mi hanno dato la forza di esprimere la mia creatività e di immergermi completamente nel progetto, riscoprendo le mie passioni.

Questa esperienza non è stata solo un arricchimento personale, ma mi ha anche fornito strumenti utili per affrontare la vita, in particolare nel mio percorso universitario. Le abilità comunicative e relazionali che ho sviluppato nel gruppo del musical mi hanno aiutato a interagire con i miei compagni di studio e a presentare le mie idee con maggiore sicurezza. Sono profondamente grata a tutti voi per aver reso questo percorso così bello e per avermi dato la possibilità di crescere, non solo come artista, ma anche come persona. La vostra accoglienza e il vostro sostegno rimarranno sempre nel mio cuore.

Innanzitutto, un ringraziamento speciale va ad Alessandro, dottorando incaricato di supervisionare il mio lavoro di tesi. Nonostante ci conosciamo solo da pochi mesi, è diventato un punto di riferimento fondamentale. Si è sempre mostrato disponibile e mi ha sempre supportato anche nei momenti di difficoltà, quando le cose non andavano per il verso giusto. Il rapporto che si è instaurato tra di noi ha reso il lavoro non solo più semplice, ma anche decisamente più piacevole. La sua pazienza, i suoi consigli e la sua presenza costante sono stati essenziali per portare a termine questo percorso, e per questo gli sono sinceramente grata.

Grazie alle mie colleghe e compagne di università Alice, Arianna e Federica, che mi hanno accolto nel loro gruppo di lavoro con grande disponibilità e amicizia. Grazie a loro, ho trovato un ambiente collaborativo e stimolante, dove ho potuto affrontare con più serenità le sfide di questo percorso. La loro compagnia ha reso l'esperienza universitaria non solo più produttiva, ma anche più piacevole.

In particolare, un ringraziamento speciale va ad Alice, con cui ho svolto la prima parte del tirocinio e diversi progetti di gruppo in coppia. Inizialmente non mi ero legata subito a lei, ma con il tempo ho scoperto la sua personalità e ho imparato ad apprezzarla profondamente. Alice mi è sempre stata vicina nei momenti di sconforto, offrendo completo sostegno quando ne avevo più bisogno. La stimo molto per la sua tenacia e determinazione, sia nelle sfide della vita che nel lavoro universitario. Grazie a lei ho scoperto il vero valore del lavoro di gruppo, trovando un'affinità caratteriale che ci ha permesso di lavorare insieme nei progetti collettivi in modo sinergico e produttivo. La sua presenza è stata fondamentale in questo percorso.

Un ringraziamento speciale va al mio ragazzo, Leonardo, e alla sua famiglia.
Leonardo è stato per me un'ancora in momenti di grande incertezza e un amico prezioso quando le difficoltà sembravano insormontabili. Nei momenti in cui mi sentivo sola e persa, sapevo di poter contare su di lui. La sua disponibilità ad ascoltarmi, senza giudizio e con comprensione, è stata una fonte di conforto inestimabile. Le sue parole incoraggianti e il suo sostegno incondizionato mi hanno dato la forza di affrontare le sfide e di non lasciarmi sopraffare dalle paure.
A differenza di quanto lui a volte possa pensare, lo stimo immensamente. Leonardo

ha una capacità unica di farmi sentire apprezzata e rispettata, e mi ha insegnato l'importanza di difendere le proprie opinioni e i propri spazi, mantenendo sempre la gentilezza e il rispetto verso gli altri. Questo equilibrio che riesce a mantenere è qualcosa che ammiro profondamente in lui.
Inoltre, la sua attitudine a vedere le cose da prospettive diverse mi ha aiutato a crescere e a riflettere sulle mie scelte. Grazie a Leonardo, ho imparato a non avere paura di esprimere me stessa e a trovare la forza per affrontare le sfide con determinazione. La sua presenza nella mia vita ha reso questo percorso non solo più facile, ma anche molto più arricchente e significativo.

Desidero ringraziare anche la sua famiglia, che mi ha accolto con calore e affetto. Un ringraziamento particolare va a Francesca, la mamma di Leonardo, che mi ha sostenuto come una seconda mamma. La sua presenza nella mia vita è stata un vero e proprio dono. Nei momenti di difficoltà, Francesca si è sempre dimostrata una presenza inestimabile. La sua disponibilità ad ascoltarmi, unita alla profondità dei suoi consigli, ha saputo offrirmi un conforto sincero e una rinnovata sicurezza. Ogni volta che avevo bisogno di un punto di vista esterno o semplicemente di qualcuno che mi comprendesse, lei era lì, pronta a supportarmi con empatia e saggezza.

Infine, il ringraziamento più speciale va alla mia famiglia, in particolare ai miei genitori, Graziella e Francesco, e ai miei fratelli, Elisa, Anna e Claudio.
Nonostante io non sia molto brava ad esternare il mio affetto, sono grata per non avermi mai fatto mancare nulla, sia in termini materiali che affettivi. La loro presenza costante e il loro sostegno incondizionato hanno rappresentato una vera e propria ancora in questo percorso di studi.

Mamma e papà, mi avete sempre incoraggiato a perseguire i miei sogni e a non fermarmi di fronte alle difficoltà. Il vostro esempio di determinazione mi ha spronata a dare il massimo in ogni situazione. Avete instillato in me un senso profondo del dovere, insegnandomi che il lavoro e l'impegno sono valori fondamentali nella vita. Questa lezione mi ha accompagnato lungo tutto il mio cammino, permettendomi di affrontare le sfide con resilienza e tenacia, senza arrendermi mai.

Essere la sorella maggiore non è sempre facile, ma sono davvero fortunata ad avere fratelli come Elisa, Anna e Claudio. In particolare, voglio esprimere la mia gratitudine per il rispetto e la comprensione che mi hanno dimostrato, specialmente nei momenti più difficili. Quando mi trovavo a studiare per gli esami, spesso avevo bisogno di silenzio per potermi concentrare. Anche se sapevo quanto fosse difficile garantire un ambiente tranquillo in una casa abitata da sei persone, loro si sono sempre mostrati disponibili e comprensivi. La loro pazienza nel rispettare i miei spazi, anche quando questo significava rinunciare a qualcosa per loro, mi ha permesso di affrontare le mie paure e di concentrarmi sui miei studi senza sentirmi sola o

sopraffatta. Questo sostegno incondizionato è stato fondamentale, non solo per il mio rendimento scolastico, ma anche per il mio benessere emotivo.
Le risate condivise e le esperienze vissute insieme hanno arricchito non solo la mia vita, ma anche il mio percorso accademico. La vostra fiducia in me e il vostro amore incondizionato mi hanno dato la forza per superare ogni ostacolo. Non sarei qui senza di voi, e per questo vi sono eternamente grata.