Università degli Studi di Salerno
Dipartimento di Informatica

Tesi di Laurea Triennale in
Informatica

Enhanced *Gene fusion* detection through Graph learning-based strategies

Relatore
Prof. Rocco Zaccagnino

Candidato
Andrea Gisolfi
Matr. 0512114162

Anno Accademico 2023-2024

"Lo sapete voi quand'è che un uomo muore davvero? Non quando il suo cuore viene raggiunto da un proiettile... e nemmeno quando viene colpito da una malattia incurabile... e nemmeno quando mangia un fungo velenoso! Muore veramente soltanto quando viene dimenticato da tutti."

-Dottor Hillk

Abstract

Gli RNA chimerici sono trascritti generati da eventi di fusione genica e splicing intergenico, combinando sequenze nucleotidiche provenienti da geni diversi. Studi recenti suggeriscono che alcuni RNA chimerici contribuiscano allo sviluppo del cancro e possano servire come biomarcatori diagnostici, specialmente quando espressi selettivamente nelle cellule tumorali, offrendo così informazioni sulle origini del tumore.

Tradizionalmente, il rilevamento di RNA chimerici e fusioni geniche si basa su letture brevi generate da Illumina e strumenti basati sull'allineamento, che identificano letture discordanti tra geni, seguite da fasi di filtraggio e riallineamento. Tuttavia, questo approccio spesso introduce bias ed è limitato da diversi fattori, portando a numerosi falsi positivi.

Metodi di previsione delle fusioni geniche senza allineamento potrebbero superare queste difficoltà e fornire nuove conoscenze sulla rottura genomica cellulare. In questa direzione, il *Machine Learning* potrebbe aprire la strada a nuove soluzioni, grazie al successo nel prevedere elementi regolatori genomici ed eventi di giunzione alternativa dal contesto genomico.

Alla luce di queste sfide, il lavoro di tesi proposto si propone di migliorare la detection delle fusioni geniche utilizzanto strategie *graph-based*, quali l'applicazione delle *Graph Neural Network*. Questo tipo di reti neurali, grazie alla loro capacità di modellare relazioni complesse tra elementi genomici, offrono un potenziale significativo per superare i limiti degli approcci tradizionali.

L'obiettivo di questo lavoro di tesi è quello di progettare una nuova soluzione che sia innovativa, affidabile, e priva dei limiti associati ai metodi basati sull'allineamento, contribuendo così a una comprensione più profonda dei meccanismi di rottura genomica e alla scoperta di nuovi biomarcatori oncologici.

Questa tesi è stata sviluppata in EDGE RESEARCH LAB

Indice

Elenco delle figure

Elenco delle tabelle

Code Listings

Capitolo 1

Introduzione

1.1 Contesto

Gli ultimi due decenni hanno assistito a una vasta ricerca sulle aberrazioni genomiche umane che si ritiene siano fattori causali di una varietà di malattie. Tra le varianti più ampiamente studiate, le *fusioni geniche* sono state di grande interesse a causa delle loro associazioni con la tumorigenesi. Un gene di fusione, chiamato anche *gene chimerico* o *gene ibrido*, è la giustapposizione di due geni altrimenti separati. [1] La fusione può derivare da riarrangiamenti strutturali come traslocazioni e delezioni, read-through della trascrizione di geni vicini o lo splicing trans e cis di pre-mRNA. Molte fusioni geniche sono associate a proprietà oncogeniche e spesso agiscono come mutazioni driver in un'ampia gamma di tipi di *cancro* [2].

La ricerca attiva sulle aberrazioni genomiche e sulle fusioni geniche è stata alimentata dall'avvento e dalle applicazioni diffuse delle tecnologie di *sequenziamento di nuova generazione* (*NGS*) ad alto rendimento, le cui applicazioni hanno rivelato paesaggi complessi di cancro umano. L'*NGS* accelera l'acquisizione di nucleotidi sequenziando simultaneamente decine o centinaia di milioni di target di sequenza e consente quindi un'analisi completa dell'intero genoma a un costo contenuto. Con i rapidi progressi delle tecnologie *NGS*, il costo del sequenziamento è diminuito drasticamente negli ultimi anni e ha reso il sequenziamento dei genomi umani una routine nei centri di sequenziamento del genoma e nelle strutture principali degli istituti. I progressi nelle tecnologie di sequenziamento hanno anche stimolato lo sviluppo del software. Sono emersi rapidamente numerosi nuovi strumenti software per identificare le *varianti strutturali* (*SV*), nonché le fusioni geniche risultanti da queste varianti [1].

1.2 Motivazioni e Obiettivi

La maggior parte degli strumenti bioinformatici di predizione si basa su una fase iniziale di *allineamento*. Vengono quindi identificate le letture discordanti che corrispondono a due geni diversi e viene applicata una serie di passaggi di filtraggio e/o riallineamento per identificare gli *RNA* chimerici candidati. Tuttavia, questo approccio è limitato da diverse combinazioni di fattori limitanti che creano molti *falsi positivi*. Soprattutto, anche se esiste un metodo robusto, riproducibile e imparziale, non è possibile identificare i geni di fusione poco espressi.

Pertanto, lo sviluppo di metodi di previsione dei geni di fusione privi di riferimenti e di allineamenti sarebbe utile e potrebbe fornire nuove conoscenze sul fenomeno della rottura genomica nella cellula. Per ovviare a queste limitazioni, sono stati proposti vari metodi basati sul *Machine Learning* (*ML*), in cui, tuttavia, il ruolo marginale del *ML* non consente di superare fattori limitanti come le diverse condizioni di profondità di sequenziamento, lunghezza delle letture e criteri di filtraggio. Finora, tuttavia, queste tecniche hanno avuto un ruolo di supporto marginale e, inoltre, i set di dati curati manualmente, fondamentali per l'addestramento dei modelli, sono spesso costosi, inaffidabili o semplicemente non disponibili [3].

L'obiettivo di questo lavoro è quello di migliorare la capacità di detection delle sequenze chimeriche attraverso un approccio *graph-based* basato sull'applicazione delle *Graph Neural Network* (GNN), con lo scopo di progettare una soluzione che sia innovativa, affidabile e che non risenta dei limiti delle altre soluzioni esistenti.

1.3 Struttura della Tesi

La struttura della tesi si articola in sei capitoli principali, ciascuno con un ruolo specifico nel guidare il lettore attraverso il percorso di ricerca, dall'introduzione generale al problema fino alla presentazione dei risultati ottenuti e delle conclusioni finali. Di seguito viene fornita una descrizione di ciascun capitolo:

- **Capitolo 1 - "Introduzione".** Questo capitolo introduce il contesto generale del lavoro di ricerca, delineando gli obiettivi principali e le motivazioni alla base dello studio. Viene presentato il problema della previsione delle fusioni geniche, con particolare attenzione alle limitazioni dei metodi attuali. Viene inoltre descritta l'importanza degli *RNA* chimerici come biomarcatori.

- **Capitolo 2 - "Stato dell'arte e analisi del lavoro svolto".** In questo capitolo viene effettuata una revisione della letteratura esistente e delle principali soluzioni proposte per la previsione delle fusioni geniche. Vengono esaminati in dettaglio gli approcci tradizionali basati sull'allineamento delle sequenze, mettendo in luce i loro limiti. Successivamente, vengono analizzati i metodi basati su Machine Learning, con un focus su come queste tecniche potrebbero essere impiegate nel contesto di studio.

- **Capitolo 3 - "Background scientifico e metodologico della ricerca".** Questo capitolo descrive il background teorico e metodologico su cui si fonda il lavoro di tesi. In particolare verrano analizzati concetti relativi alla teoria dei grafi, Machine Learning, bioinformatica e reti neurali con maggiore attenzione su quelle che sono le *GNN*.

- **Capitolo 4 - "A GNN-based Gene fusion detection method".** In questo capitolo si entra nel dettaglio dell'implementazione del modello proposto. Vengono descritte le diverse pipeline di implementazione che hanno portato alla realizzazione di diversi modelli di *GNN*.

- **Capitolo 5 - "Risultati ottenuti".** Questo capitolo presenta i risultati ottenuti durante la fase di valutazione del modelli. Vengono descritti i test effettuati per misurare l'efficacia dei modelli nella previsione delle fusioni geniche, con un'analisi delle metriche di valutazione utilizzate, come accuratezza, precisione e recall. Inoltre, si confrontano le prestazioni dei modelli proposti con altri metodi basati su Machine Learning, sottolineando i vantaggi e i limiti del nuovo approccio.

- **Capitolo 6 - "Conclusioni".** L'ultimo capitolo riassume i principali risultati ottenuti nel corso della tesi e discute le implicazioni delle scoperte fatte per il campo della previsione delle fusioni geniche.

Capitolo 2

Stato dell'arte e analisi del lavoro svolto

Questo capitolo ha l'obiettivo di fornire una panoramica dettagliata dello stato dell'arte, esaminando le soluzioni tradizionali *basate su allineamento* e le prime applicazioni di tecniche di *Machine Learning* in questo campo.

2.1 Previsione delle fusioni geniche: metodi tradizionali

STAR-Fusion. *STAR-Fusion* [4] è uno strumento di previsione della fusione genica che sfrutta la velocità e l'accuratezza dell'allineatore *STAR* (*Spliced Transcripts Alignment to a Reference*). *STAR* è uno strumento di allineamento *RNA-Seq*[1] rapido e accurato con capacità di segnalare letture divise e coppie allineate in modo discordante. *STAR-Fusion* esegue una mappatura rapida delle prove di fusione per fare riferimento alle annotazioni della struttura della trascrizione e filtra i probabili artefatti per segnalare previsioni di fusione accurate. Un aspetto importante di questo tool è che può essere utilizzato in combinazione con altri strumenti per ottenere una visione più completa delle fusioni geniche presenti nei dati di sequenziamento dell'*RNA-Seq*. Tuttavia, può soffrire di un elevato numero di falsi positivi, soprattutto in regioni di splicing complesso, e richiede elevata potenza computazionale per eseguire l'allineamento.

[1] RNA-Seq è una tecnica che utilizza il sequenziamento di nuova generazione per rivelare la presenza e la quantità di molecole di RNA in un campione biologico. (Fonte: *Wikipedia*)

TopHat-Fusion. *TopHat-Fusion* [5] è un algoritmo progettato per scoprire trascrizioni che rappresentano prodotti genici di fusione, che derivano dalla rottura e dalla ri-unione di due cromosomi diversi, o da riarrangiamenti all'interno di un cromosoma; si tratta di una versione migliorata di TopHat, un programma efficiente che allinea le letture *RNA-seq* senza basarsi sull'annotazione esistente. Poiché è indipendente dall'annotazione genica, questo strumento può scoprire prodotti di fusione derivanti da geni noti, geni sconosciuti e varianti di splicing non annotate di geni noti. Anche *TopHat-Fusion* può generare molti falsi positivi, specialmente in regioni ricche di trascrizioni alternative. Inoltre, ha difficoltà nel gestire fusioni complesse con più partner o letture rumorose

EricScript. *EricScript* [6] è un framework computazionale che utilizza una combinazione di quattro processi di allineamento per identificare le firme di trascrizione di fusione.Comprende i seguenti passaggi:

- Mappatura delle letture rispetto al trascrittoma.
- Identificazione di allineamenti discordanti e costruzione del riferimento della giunzione esonica.
- Ricalibrazione del riferimento della giunzione esonica.
- Punteggio e filtraggio delle fusioni geniche candidate.

Il primo allineamento viene utilizzato per identificare allineamenti discordanti e per costruire un riferimento di giunzione esonica. Questo passaggio è seguito dalla mappatura di tutte le letture rispetto a questo nuovo riferimento per rilevare le letture che non sono mappate correttamente. Per stimare con precisione i confini esatti delle giunzioni, esegue un ulteriore riallineamento locale delle letture non mappate correttamente rispetto al riferimento di giunzione esonica. L'ultima procedura di mappatura consente l'identificazione delle letture di spanning, ovvero le letture che si estendono attraverso le giunzioni e producono un elenco di fusioni candidate. Infine, stima un punteggio di probabilità per ciascuna fusione prevista e utilizza diversi filtri euristici per rimuovere gli artefatti di analisi.Nonostante le buone prestazioni, anche questo tool è soggetto a falsi positivi, e la sua precisione dipende fortemente dalla qualità dei dati di input e dall'efficacia dell'algoritmo di filtraggio.

FusionMap. *FusionMap* [7] è un tool che allinea le letture di fusione direttamente al genoma senza una conoscenza pregressa delle potenziali regioni di fusione; questo tool è in grado di rilevare eventi di fusione in dataset sia single-end che paired-end da studi *RNA-Seq* o *DNA-Seq* e caratterizzare le

giunzioni di fusione a risoluzione di coppia di basi. Il tool fa un uso sostanziale di un allineatore di base per rilevare e allineare le letture di fusione al riferimento del genoma. L'allineamento di base è un'implementazione di un metodo *GSNAP* (*Genomic Short-read Nucleotide Alignment Program*) modificato. *GSNAP* è un allineatore basato su tabella hash che consente più disallineamenti e indel lunghi. Come gli altri strumenti basati sull'allineamento, *FusionMap* può produrre risultati falsi positivi, specialmente in regioni ripetitive o con trascritti complessi.

2.1.1 Considerazioni generali sui metodi basati sull'allineamento

I metodi basati sull'allineamento sono stati fondamentali nella ricerca delle fusioni geniche, ma sono caratterizzati da alcune problematiche comuni, tra cui:

- **Falsi positivi**: Gli algoritmi spesso confondono varianti di *splicing* o *trascrizioni alternative* con fusioni reali.
- **Lentezza e complessità computazionale**: L'allineamento di dati *RNA-seq* di un genoma di riferimento è molto intensivo in termini di risorse computazionali.
- **Difficoltà nel gestire letture chimeriche**: Quando le letture *RNA-seq* derivano da trascritti chimerici o sequenze molto diverse, l'allineamento può essere impreciso, creando problemi di interpretazione dei risultati.

2.2 Approcci basati su Machine Learning

Il campo del *ML* [8] si occupa dello sviluppo e dell'applicazione di algoritmi informatici che migliorano con l'esperienza. I metodi di apprendimento automatico sono stati applicati a un'ampia gamma di settori della genetica e della genomica. Il *ML* è forse più utile per l'interpretazione di grandi insiemi di dati genomici ed è stato utilizzato per annotare un'ampia varietà di elementi di sequenza genomica. Ad esempio, i metodi di *ML* possono essere utilizzati per "imparare" a riconoscere la posizione dei siti di inizio trascrizione (*TSS*) in una sequenza genomica. Allo stesso modo, gli algoritmi possono essere addestrati per identificare siti di splice, promotori, enhancer o nucleosomi posizionati. In generale, se si può compilare un elenco di elementi di sequenza di un determinato tipo, è probabile che un modello di

ML possa essere addestrato a riconoscere tali elementi. Inoltre, i modelli che riconoscono un singolo tipo di elemento genomico possono essere combinati, insieme alla logica sulla loro posizione relativa, per costruire sistemi di *ML* in grado di annotare i geni.

Di seguito viene riportato un metodo *basato sulle frasi* sfrutta tecniche di *NPL* (*Natural Language Processing*), trattando le sequenze nucleotidiche come dati "testuali", per estrarre modelli semantici dalle letture.

Sequence-based method. Questo approccio si basa sulla definizione di un modello in grado di analizzare e classificare liste di *k-mers*. [9] Le letture sono rappresentate come insiemi di frasi composte da *k-mers*, per scoprire le strutture semantiche nascoste all'interno dei dati genomici. Più precisamente, data una lettura di lunghezza n e un valore k, si estrae innanzitutto l'elenco dei suoi n - k + *1 k-mers*. Successivamente, tale elenco viene diviso in segmenti consecutivi di n *k-mers*. I *k-mers* di un segmento vengono poi uniti in una frase utilizzando un carattere di spazio come "separatore", producendo così un insieme di frasi.

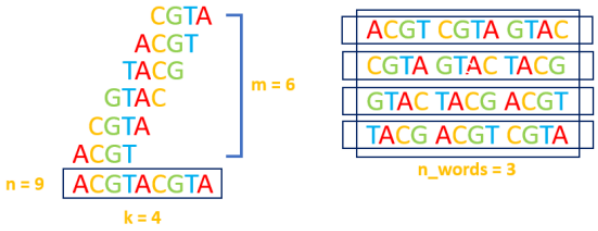


Figura 2.1: Esempio di creazione di un insieme di frasi.

Questa rappresentazione basata su frasi viene a sua volta sfruttata da un modello basato su *DL* (*Deep Learning*) per il rilevamento di letture chimeriche, costruito come un insieme di due sottomodelli: *Gene classifier* e *Fusion classifier*.
L'obiettivo del *Gene classifier* è classificare una frase nel gene da cui è stata generata. Questo modello viene addestrato utilizzando tutte le frasi derivate da letture non chimeriche estratte dai trascritti di un insieme di geni di riferimento. Per addestrare il *Fusion classifier*, viene generato un insieme di letture chimeriche e non chimeriche dallo stesso insieme di geni di riferimento utilizzato per l'addestramento del *Gene classifier*. Quindi, per ogni lettura vengono generate tutte le frasi di lunghezza *n_words*, che vengono poi fornite come input al *Gene classifier* addestrato, che comprende uno

strato di embedding e diversi *strati di classificazione*. Gli output dello strato di embedding per tutte le frasi generate vengono raggruppati in un'unica *matrice di embedding*, che costituisce l'input per il *Fusion classifier*. Il *Fusion classifier* utilizza poi tali matrici di embedding per distinguere tra letture che derivano dalla fusione di due geni o da un singolo gene.

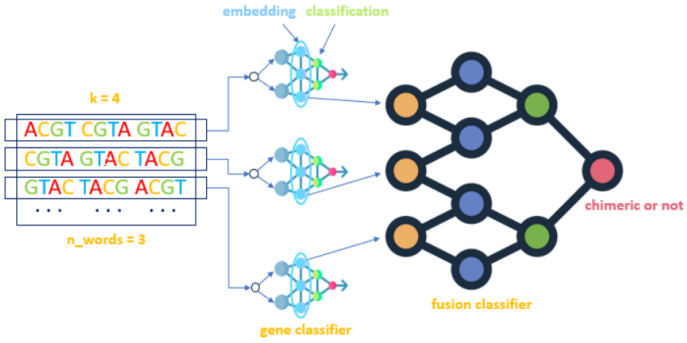


Figura 2.2: Architettura del modello

Entrambi i classificatori sono basati sulle reti neurali, le cui architetture sono state messe a punto attraverso un'ampia sperimentazione, che ha rivelato che l'utilizzo delle matrici di incorporamento generate dal *Gene classifier* come input del *Fusion classifier*, piuttosto che i suoi output di classificazione, migliora significativamente le prestazioni. Questo miglioramento è dovuto alla capacità degli embeddings di catturare in modo più efficace la rappresentazione interna del modello, anziché affidarsi a un risultato di classificazione semplificato. Questo approccio offre un metodo efficiente e relativamente semplice per individuare i punti di fusione, sebbene sia limitato nella sua capacità di cogliere relazioni più complesse tra regioni distanti delle sequenze.

Capitolo 3

Background scientifico metodologico della ricerca

In questo capitolo verrà esposto un quadro generale su aspetti di base che caratterizzano metodologie e tecnologie adoperate in questo lavoro di tesi; in particolare vengono descritti concetti relativi alla *teoria dei grafi*, *Bioinformatica*, *ML* e sulle reti neurali, con maggiore attenzione alle *GNN*. Inoltre, verranno presentate due metodologie di codifica utilizzate nel lavoro di tesi per trasformare le sequenze geniche in dati numerici utilizzati per addestrare il modello, il *One-Hot encoding* e *DNABERT*.

3.1 Teoria dei grafi

Un grafo *non orientato* $G = (V, E)$ è dato da una coppia distinta di insiemi, dove $V = \{v_1, v_2, ..., v_n\}$ è l'insieme degli n vertici o nodi del grafo e $E = \{e_1, e_2, ..., e_n\} \subseteq V \times V$ è l'insieme degli m archi non orientati di G.

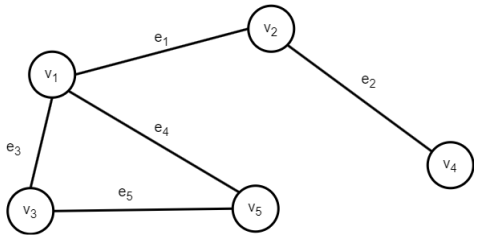


Figura 3.1: Esempio di un grafo non orientato.

Ogni arco non orientato $e_k = (v_i, v_j)$ di G corrisponde ad una coppia non

ordinata di nodi v_i, v_j di G. I nodi v_i, v_j sono gli estremi dell'arco e_k. La presenza di un arco tra una coppia di nodi indica una relazione tra i nodi stessi. Di seguito sono riportate alcune definizioni di base per i grafi non orientati:

- Un arco $e = (v, v) \in E$ è detto *loop*;
- Un arco $e = (u, v) \in E$ si dice *incidente* su u e v;
- Due nodi $u, v \in V$ sono detti *adiacenti* $\iff u, v \in E$;
- Due archi $e_1, e_2 \in E$ sono detti *adiacenti* $\iff e_1 = (u, v)$ e $e_2 = (v, w)$;
- L'insieme dei nodi $N(u) = \{v \in V | v$ è adiacente a $u\}$ è detto *intorno* di u in G;
- L'insieme degli archi $\delta(u) = \{e \in E | e$ incide su $u\}$ è detto *stella* di u in G
- $|\delta(u)|$ è detto *grado* del nodo u
- Un *cammino* in *G = (V, E)* è una sequenza di nodi $v_1, v_2, ..., v_k$ tale che ogni nodo è adiacente al nodo successivo, cioè $(v_{i-1}, v_i) \in E$ per *i = 1, 2, ..., k.*
- Un grafo *G = (V, E)* è *connesso* se per ogni coppia di nodi u e v, esiste un cammino fra u e v.

Un grafo *G = (V, E)* è un grafo *orientato* se, dato $V = \{v_1, v_2, ..., v_n\}$, l'insieme degli archi $E = \{e_1, e_2, ..., e_n\}$ è formato da coppie ordinate di nodi. Per un grafo orientato *G = (V, E)* si ha che $e_k = (v_i, v_j) \neq e_h = (v_j, v_i)$ con $e_k, e_h \in E$.

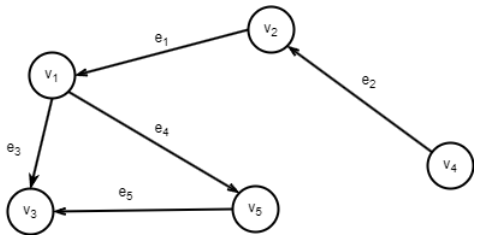


Figura 3.2: Esempio di grafo orientato.

L'arco e_k si dice *uscente* da v_i ed *entrante* in v_j; v_i e v_j sono rispettivamente

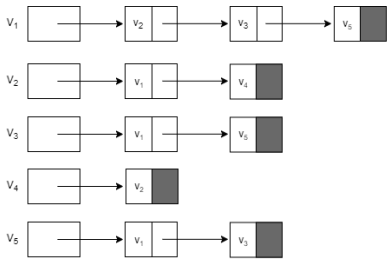
la *coda* e la *testa* di e_k. Molte delle definizioni di base per i grafi non orientati valgono anche per i grafi orientati, di seguito sono riportate le altre definizioni di base per i grafi orientati:

- $Fs(v) = \{u \in V | (v, u) \in E\}$ è detto *stella uscente* di v;
- $Bs(v) = \{u \in V | (u, v) \in E\}$ è detto *stella entrante* di v;
- $S(v) = Fs(v) \cup Bs(v)$ è detto *stella* di v;

3.1.1 Rappresentazioni di un Grafo

Condideriamo due modi per rappresentare un grafo:

- **liste di adiacenza**: un vettore Adj con indici da 1 a n in cui $Adj[i]$ contiene un puntatore alla lista dei nodi adiacenti di i;
- **matrice di adiacenza**: una matrice quadrata $A = (a_{i,j})$ di dimensione $n \times n$ tale che $a_{i,j} = 1$ se $(i, j) \in E$ e $a_{i,j} = 0$ altrimenti.



	1	2	3	4	5
1	0	1	1	0	1
2	1	0	0	1	0
3	1	0	0	0	1
4	0	1	0	0	0
5	1	0	1	0	0

Figura 3.3: Lista di adiacenza (sinistra) e matrice di adiacenza (destra) del grafo in 3.1.

Con le *liste di adiacenza* vi sono due rappresentazioni di ogni arco, lo spazio di memoria occupato è proporzionale a $m + n$ poichè ogni arco compare *2* volte e il costo delle operazioni è:

- Verificare se *(u, v)* è un arco richiede tempo $O(\delta(u))$;
- Elencare tutti gli archi richiede tempo $\Theta(m + n)$;
- Elencare tutti gli archi incidenti su un fissato nodo v richiede tempo $\Theta(\delta(u))$.

Con la *matrice di adiacenza* vi è una doppia rappresentazione di ogni arco, lo spazio di memoria occupato è proporzionale a n^2 e il costo delle operazioni è:

- Verificare se *(u, v)* è un arco richiede tempo $\Theta(1)$;
- Elencare tutti gli archi richiede tempo $\Theta(n^2)$;
- Elencare tutti gli archi incidenti su un fissato nodo v richiede tempo $\Theta(n)$.

In relazione al lavoro di tesi svolto, per ogni grafo è stata costruita la matrice di adiacenza, la quale è stata poi data in input al modello di GNN per la previsione delle fusioni.

3.2 Machine Learning

Negli ultimi dieci anni, l'*intelligenza artificiale* (*IA*) [10] è diventata un argomento popolare sia all'interno che all'esterno della comunità scientifica; un'abbondanza di articoli in riviste tecnologiche e non, hanno trattato i temi del *ML*, del *DL* e dell'*IA*. Tuttavia, c'è ancora confusione intorno ad *IA*, *ML* e *DL*. I termini sono fortemente associati, ma non sono intercambiabili. Il *ML* è un campo che si concentra sull'aspetto dell'apprendimento dell'*IA* sviluppando algoritmi che rappresentino al meglio un insieme di dati. A differenza della programmazione classica, in cui un algoritmo può essere codificato esplicitamente utilizzando caratteristiche note, il *ML* utilizza sottoinsiemi di dati per generare un algoritmo che può utilizzare combinazioni di caratteristiche e pesi nuove o diverse da quelle che possono essere derivate da principi primi.
Esistono quattro metodi di apprendimento comunemente utilizzati, ciascuno utile per risolvere compiti diversi:

- **Apprendimento supervisionato**. L'apprendimento supervisionato è quello in cui un agente apprende usando dei *dati etichettati*. Le etichette determinano la *variabile dipendente*, ovvero quello che l'agente dovrà apprendere. Quindi, per ogni osservazione, oltre ai dati di input è noto anche il valore della variabile dipendente. I problemi di apprendimento supervisionato più comuni sono quelli che riguardano problemi di *classificazione* o *regressione*. La regressione è un task in cui l'obiettivo è predire il valore di una variabile numerica, mentre una classificazione è un task in cui l'obiettivo è predire il valore di una variabile categorica.

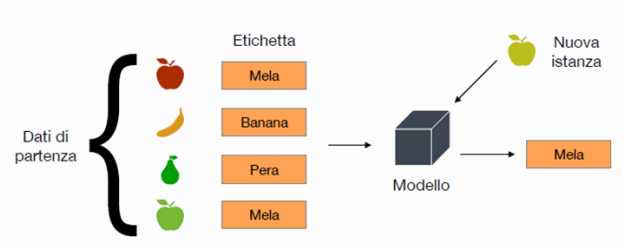


Figura 3.4: Esempio di un problema di classificazione.

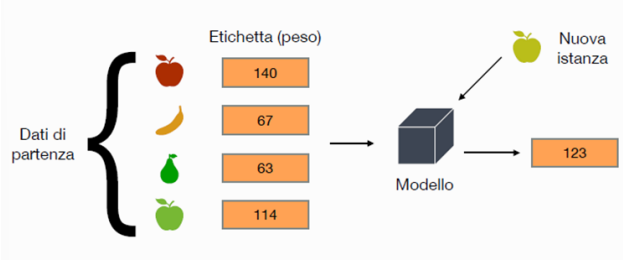


Figura 3.5: Esempio di un problema di regressione.

- **Apprendimento non supervisionato**. L'apprendimento non supervisionato è quello in cui un agente apprende usando *dati non etichettati*. L'agente sarà quindi in grado di imparare senza conoscere il valore reale della variabile dipendente. Generalmente, agenti non supervisionati vengono usati per problemi più complessi di quelli supervisionati. Un problema di apprendimento non supervisionato molto comune è quello del *clustering*. Il clustering è una task in cui l'obiettivo è raggruppare oggetti in gruppi che abbiano un certo grado di omogeneità ma che, al tempo stesso, abbiano un certo grado di eterogeneità rispetto agli altri gruppi.

Figura 3.6: Esempio di un problema di clustering.

- **Apprendimento semisupervisionato**. L'apprendimento semisupervisionato [10] può essere considerato come una "via di mezzo" tra l'apprendimento supervisionato e quello non supervisionato ed è particolarmente utile per i set di dati che contengono sia dati etichettati che non etichettati. Questo modello viene utilizzato per classificare il resto dei dati non etichettati del set di dati.

- **Apprendimento per rinforzo**. L'apprendimento per rinforzo [10] è la tecnica di addestramento di un algoritmo per un compito specifico in cui non esiste una singola risposta corretta, ma si desidera un risultato complessivo. È probabilmente il tentativo più vicino a modellare l'esperienza di apprendimento umana, perché apprende anche da tentativi ed errori piuttosto che dai soli dati.

3.2.1 One-Hot Encoding

I modelli di Machine Learning [11] agiscono sui dati, richiedono istanze di input x, per produrre un output. In questo contesto, x è generalmente un insieme di attributi, noti anche come *caratteristiche*, che descrivono una particolare istanza di punto campione. Se si utilizzano n caratteristiche per descrivere un particolare campione, x diventa un punto campione in uno spazio dati *n-dimensionale*: $x = [x_1, x_2, ..., x_n]$.

Si possono distinguere due tipi di caratteristiche: *numeriche* e *categoriali*. Le caratteristiche numeriche sono solitamente rappresentate da numeri a virgola mobile o interi e si presentano naturalmente in molti camp., La rappresentazione delle variabili categoriali è meno ovvia, poiché non esiste un modo naturale per eseguire calcoli numerici sulle categorie. L'approccio più comune per convertire le caratteristiche categoriali in un formato adatto all'uso come input per un modello di Machine Learning è la codifica *One-Hot*. Ogni categoria viene rappresentata da un vettore in cui un solo elemento è impostato a *1* (attivo) e tutti gli altri a *0* (inattivi). Nel caso di sequenze di *DNA*, i nucleotidi corrispondono alle categorie, ed essendo *4* nucleotidi *(A,G,T,C)* è possibile rappresentarli tramite vettori binari di dimensione *4*, come mostrato in *Figura* 3.7.
In generale, per variabili di cardinalità *d*, i vettori avrebbero *d-dimensioni*. Una proprietà interessante della codifica one-hot è che le categorie sono rappresentate come concetti indipendenti.

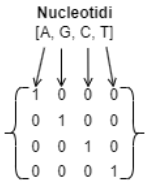


Figura 3.7: Rappresentazione grafica della codifica.

3.3 Reti neurali

Le reti neurali [12] sono costituite da un insieme di neuroni e archi e traggono le loro origini dall'analisi dei circuiti. A ciascun arco che collega i neuroni possono essere applicati pesi diversi. A ogni neurone viene applicata una *funzione di attivazione* a un segnale di ingresso pesato per generare un segnale di uscita. I neuroni sono ulteriormente suddivisi in uno strato di ingresso, uno strato nascosto e uno strato di uscita, *Figura 3.8*. Gli strati nascosti eseguono il livello di astrazione necessario per passare dallo strato di ingresso a quello di uscita. Il numero di strati nascosti definisce se il sistema è un sistema di apprendimento superficiale (con uno o pochi strati nascosti) o profondo (con molti strati nascosti).

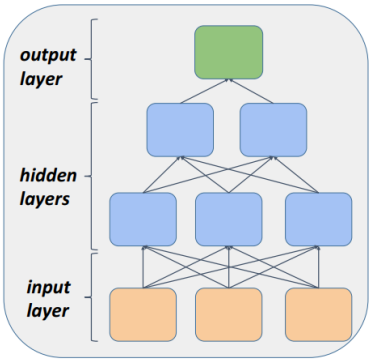


Figura 3.8: Struttura di una rete neurale.

Esiste un compromesso intrinseco tra il numero di strati nascosti e il tempo necessario per addestrare il modello. Per questo motivo, sebbene il concetto di base della rete neurale non sia nuovo, ha trovato una nuova applicazione grazie ai recenti progressi della potenza di calcolo. Il tipo più elementare è noto come rete neurale *feedforward* (*FNN*), in quanto l'informazione si propaga semplicemente dallo strato di ingresso allo strato nascosto e infine

allo strato di uscita. Lo stato attuale del sistema non è definito da nessuno stato passato; pertanto, rappresenta un sistema senza memoria. Un altro tipo di rete neurale è rappresentato dalle reti neurali *ricorrenti* (*RNN*). Si tratta di una classe di reti neurali dedicate alle serie di dati temporali, in quanto tengono conto della relazione sequenziale intrinseca osservata nei dati da un punto temporale all'altro. La forma di base di una *RNN* è mostrata nella *Figura 3.9*, dove ogni stato corrente (al tempo t) è definito da una combinazione dello stato precedente del sistema e dell'ingresso corrente. I pesi per ogni arco possono essere determinati in base a quanto indietro guardare, in modo simile a una costante temporale.

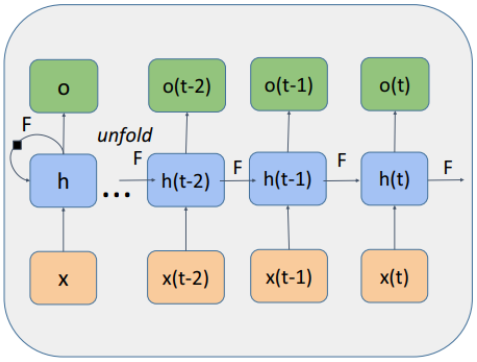


Figura 3.9: Struttura di una rete neurale ricorrente.

Esistono molti altri tipi di reti neurali, tuttavia, il lavoro di tesi utilizza le *GNN* come strumento principale; pertanto, la prossima sezione introduce il concetto di rete neurale a grafo, spiegando il suo funzionamento e mostrando i vari tipi di queste reti.

3.3.1 Graph Neural Network

Il progresso del deep learning [13] ha mostrato un grande potenziale nelle applicazioni di classificazione del parlato, delle immagini e dei video. In queste applicazioni, i modelli di deep learning vengono addestrati da insiemi di dati nello spazio euclideo con dimensioni e sequenze fisse. Tuttavia, la rapida crescita delle richieste di analisi di insiemi di dati in spazi non euclidei richiede ulteriori ricerche. In generale, trovare le relazioni tra gli elementi degli insiemi di dati e rappresentare tali relazioni come grafi ponderati costituiti da vertici e archi è un modo valido per analizzare gli insiemi di dati in uno spazio non euclideo. Tuttavia, l'analisi dei dataset basati su grafi ponderati è un problema impegnativo per i modelli di deep learning esistenti.

Per risolvere questo problema, le *GNN* sfruttano strategie spettrali e spaziali per estendere e implementare le operazioni di convoluzione nello spazio non euclideo.

In alcuni scenari reali, i dati non possono essere mappati nello spazio euclideo, che è definito in R^n , ovvero i dati dello spazio euclideo possono essere modellati e rappresentati come un insieme di punti in uno spazio lineare *n-dimensionale*. Tuttavia, è difficile utilizzare uno spazio lineare *n-dimensionale* per codificare alcuni dati, come quelli dei social network, a causa delle dimensioni dinamiche. Pertanto, è necessario estendere l'insieme delle strutture di dati che utilizziamo per l'apprendimento automatico dagli spazi euclidei agli spazi non euclidei. Per gestire i dati dello spazio non euclideo, le *GNN* sono in grado di estrarre e combinare le caratteristiche dei dati spaziali locali multiscala con elevate capacità di rappresentazione, estendendo così i modelli di deep learning alla presentazione di dati dello spazio non euclideo. Essendo una struttura di dati non euclidea unica, i grafi hanno attirato l'attenzione sulla classificazione dei nodi, sulla previsione dei collegamenti e sull'analisi dei cluster. Grazie all'elevata interpretabilità, il *GNN* è diventato di recente un metodo di analisi dei grafi ampiamente utilizzato. Di seguito vengono presentati alcuni dei modelli tipici di *GNN*, soffermandosi sulla loro stuttura e funzionamento.

Graph Convolution Network

Il *Graph Convolution Network* (*GCN*) [13] è un modello *GNN* di base sul quale si basano diversi studi che mirano a creare nuovi modelli di *GNN*.

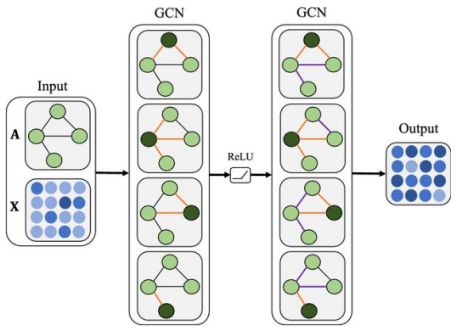


Figura 3.10: Struttura di un GCN.

La *Figura 3.10* illustra la struttura di *GCN*. L'input del modello è un

intero grafo. Nel livello di convoluzione *1*, viene eseguita un'operazione di convoluzione sui vicini di ogni nodo e il nodo viene aggiornato con il risultato della convoluzione. Quindi, il *GCN* applica la *funzione di attivazione* (ad esempio, *ReLU*) ai risultati della convoluzione. Dopo il livello di attivazione, l'output viene spinto in un altro livello di convoluzione e in un altro di attivazione, che costituisce il secondo ciclo. Il processo viene quindi ripetuto finché l'output non si avvicina ai requisiti di precisione. In questo modo, il *GCN* può aumentare la profondità degli strati di convoluzione in modo da soddisfare i requisiti di precisione per ogni caso specifico. Il *GCN* dispone di una funzione di uscita locale, utilizzata per convertire lo stato del nodo (compresi lo stato nascosto e la caratteristica del nodo) in tag relativi al compito. In generale, esistono due compiti diversi per una *GCN*. Il primo è un compito di classificazione a livello di nodo, il secondo è un compito di classificazione a livello di grafo.

Graph Attention Network

Le *Graph Attention Networks* (*GAT*) [14] sono una delle architetture *GNN* più popolari e sono considerate lo stato dell'arte per l'apprendimento delle rappresentazioni con i grafi. In un *GNN*, ogni nodo aggiorna iterativamente il proprio stato interagendo con i suoi vicini. Le varianti di *GNN* si differenziano principalmente per il modo in cui ogni nodo aggrega e combina le rappresentazioni dei suoi vicini con le proprie. In *GAT*, ogni nodo aggiorna la propria rappresentazione partecipando ai suoi vicini utilizzando la propria rappresentazione come query. Ciò generalizza la media standard o il max-pooling dei vicini, consentendo a ogni nodo di calcolare una media ponderata dei suoi vicini e di selezionare (in modo soft) i suoi vicini più rilevanti.

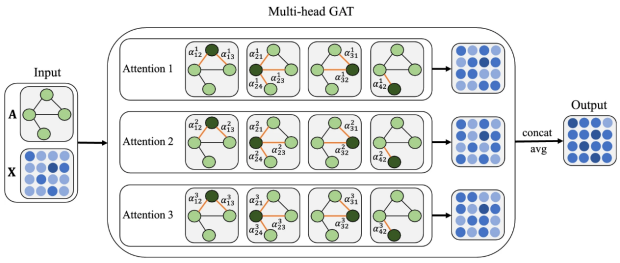


Figura 3.11: Struttura di un GAT multi-head.

Graph Sample and Aggregate

Un altro modello di rete neurale a convoluzione di grafi è il *Graph Sample and Aggregate* (*GraphSAGE*) [13]. Come variazione rispetto al *GNN*, l'ordine di ingresso delle funzioni di aggregazione non ha alcun impatto sul risultato di *GraphSAGE*, il che significa che *GraphSAGE* può gestire nodi vicini disordinati.
I tre passaggi chiave di *GraphSAGE* sono i seguenti:

- un numero fisso di nodi vicini viene ottenuto tramite campionamento;
- la funzione di aggregazione viene utilizzata per ottenere le informazioni di aggregazione dei nodi vicini in modo da poter ottenere le caratteristiche dell'elemento centrale;
- le informazioni di aggregazione dei nodi vicini vengono utilizzate per classificare o prevedere il contenuto o le etichette dell'elemento centrale.

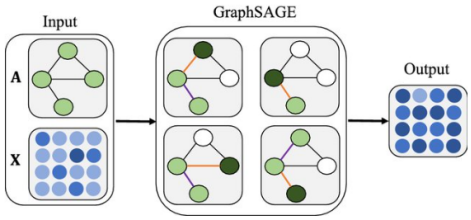


Figura 3.12: Struttura di un GraphSAGE.

3.3.2 DNABERT

Decifrare il linguaggio del *DNA* non codificante è uno dei problemi fondamentali nella ricerca sul genoma. Il codice regolatore dei geni è altamente complesso a causa dell'esistenza di polisemia e di relazioni semantiche distanti, che i precedenti metodi informatici spesso non riescono a catturare, soprattutto in scenari con dati scarsi.

Negli ultimi anni, sono stati sviluppati molti strumenti computazionali, la maggior parte dei quali si concentrano sulla caratteristica sequenziale del *DNA* e tentano di catturare la dipendenza tra gli stati applicando modelli basati su reti neurali ricorrenti (*RNN*), come reti *Long Short-Term Memory* (*LSTM*) e *Gated Recurrent Units* (*GRU*). Tuttavia, i modelli *RNN*

(*LSTM, GRU*), sebbene in grado di apprendere la dipendenza a lungo termine, soffrono notevolmente del gradiente di scomparsa e del problema della bassa efficienza quando elaborano sequenzialmente tutti gli stati passati e comprimono le informazioni contestuali in un collo di bottiglia con lunghe sequenze di input. Inoltre, la maggior parte dei modelli esistenti richiede enormi quantità di dati etichettati, con conseguenti prestazioni e applicabilità limitate in scenari con dati scarsi, in cui ottenere dati di alta qualità con etichette è costoso e richiede molto tempo.
Per risolvere tali limitazioni, è stato sviluppato un metodo di apprendimento profondo chiamato *DNABERT* [15], basato sul modello *Bidirectional Encoder Representations from Transformers* (*BERT*). *DNABERT* applica i *Transformer*, un'architettura basata sull'attenzione che ha raggiunto prestazioni all'avanguardia nella maggior parte delle attività di elaborazione del linguaggio naturale. *DNABERT* prende innanzitutto un set di sequenze rappresentate come token *k-mer* come input. Ogni sequenza è rappresentata come una matrice M incorporando ogni token in un vettore numerico. Formalmente, il modello cattura le informazioni contestuali eseguendo il meccanismo di *auto-attenzione multi-head* su M.

3.4 Bioinformatica

La *Bioinformatica* [16] è una disciplina poliedrica che combina molti campi scientifici, tra cui la biologia computazionale, la statistica, la matematica, la biologia molecolare e la genetica.

Questa disciplina consente ai ricercatori biomedici di sfruttare le tecnologie computazionali esistenti ed emergenti per archiviare, estrarre, recuperare e analizzare senza soluzione di continuità i dati provenienti dalle tecnologie genomiche e proteomiche. Ciò si ottiene creando modelli di dati unificati, standardizzando le interfacce dei dati, sviluppando vocabolari strutturati, generando nuovi metodi di visualizzazione dei dati e catturando metadati dettagliati che descrivono i vari aspetti del disegno sperimentale e dei metodi di analisi.

Con la continua evoluzione della ricerca biologica, la necessità di incorporare nuove competenze provenienti da diverse discipline non è mai stata così grande se vogliamo comprendere e realizzare le potenzialità di imprese grandiose come il Progetto Genoma Umano[1] (*The Human Genome Project).*

L'enorme quantità di dati generati attraverso gli sforzi di sequenziamento dell'intero genoma, le tecnologie di microarray, la mappatura dei polimorfismi

[1] https://www.genome.gov/human-genome-project

a singolo nucleotide (*SNP*), la proteomica e molti altri sforzi hanno creato un abisso di conoscenze per la maggior parte dei biologi con formazione tradizionale su come interpretare e utilizzare la ricchezza di informazioni.

La bioinformatica è il campo scientifico in cui biologia, informatica e tecnologia dell'informazione si fondono per formare un'unica disciplina. L'obiettivo finale del campo è quello di consentire la scoperta di nuove conoscenze biologiche e di creare una prospettiva globale dalla quale si possano discernere i principi unificanti della biologia.

3.4.1 DNA e RNA

James Watson e Francis Crick [16] hanno descritto per la prima volta la struttura dell'*acido desossiribonucleico* (*DNA*) nel 1953. Il *DNA* è un polimero composto principalmente da quattro molecole, *adenina* (*A*), *guanina* (*G*), *timina* (*T*) e *citosina* (*C*), disposte a doppia elica antiparallela con le coppie di basi legate chimicamente per tenere unite le due eliche. Esso contiene le informazioni genetiche che determinano le caratteristiche di un individuo, come il colore degli occhi, l'altezza, la predisposizione a determinate malattie e così via.

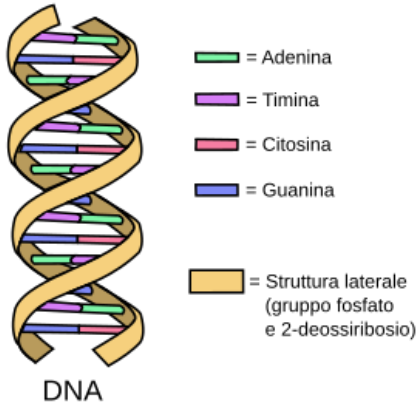


Figura 3.13: Struttua del DNA.

L'*acido ribonucleico* (*RNA*) è una molecola simile al *DNA* ma più piccola e generalmente a singola elica. L'*RNA* ha diverse funzioni all'interno delle cellule, incluso il trasferimento delle informazioni genetiche dal *DNA* alle proteine che svolgono le funzioni biologiche. Si tratta di una molecola a singolo filamento che contiene tre delle quattro coppie di basi del *DNA*, *G,*

A, C e una quarta base, l'*uracile*, *U*. Tutto l'*RNA* in una cellula è prodotto dalla *trascrizione* del *DNA*; il processo inizia con l'apertura e lo svolgimento di una piccola porzione della doppia elica del *DNA* per esporre le basi su ogni filamento di *DNA*. Uno dei due filamenti della doppia elica del *DNA* funge quindi da stampo per la sintesi di una molecola di *RNA*. La catena di *RNA* prodotta dalla trascrizione, il *trascritto*, è quindi allungata di un nucleotide alla volta e ha una sequenza nucleotidica che è esattamente complementare al filamento di *DNA* utilizzato come stampo. [17]

3.4.2 Fusioni geniche

Le fusioni geniche [18] sono geni ibridi risultanti dalla fusione di due geni precedentemente separati. Le fusioni geniche si formano in seguito a un riarrangiamento cromosomico che comprende *traslocazione*, *inversione*, *delezione* o *duplicazione in tandem*.

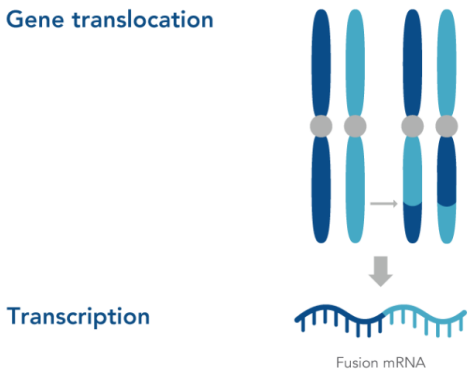


Figura 3.14: Esempio di fusione genica avvenuta per traslocazione.

È interessante notare che le fusioni geniche sono firme citogenetiche caratteristiche di molti tipi di cancro e sono state utilizzate con successo come strumenti diagnostici. Spesso, le fusioni geniche danno origine a trascritti di fusione genica e a prodotti proteici chimerici, che sono stati utilizzati come bersagli per il trattamento.

Mentre il termine "*fusione genica*" si riferisce a eventi di fusione a livello del *DNA*, "*RNA chimerico*" si riferisce a qualsiasi trascritto composto da esoni di geni parentali diversi, compresi i trascritti di fusione genica. Contrariamente a quanto si crede, la trascrizione di fusioni geniche non è l'unico mezzo di generazione di *RNA chimerici*. Gli *RNA chimerici* possono anche derivare dalla trasplicazione di due *mRNA* precursori separati

e dallo splicing alternativo di un trascritto readthrough, altrimenti noto come *cis-splicing* di geni adiacenti (*cis-SAGe*).

È interessante notare che diversi studi indicano la presenza di *RNA chimerici* che imitano esattamente i trascritti di fusione genica comune in assenza della fusione genica corrispondente. Queste osservazioni hanno stimolato l'ipotesi secondo la quale gli *RNA chimerici* possono essere generati prima dal trans-splicing e poi dal riarrangiamento del genoma per formare la fusione genica corrispondente.

3.4.3 Grafo di De Bruijn

Ai fini del lavoro di tesi, è stato utilizzato un particolare tipo di grafo molto utilizzato nell'ambito bioinformatico, il grafo di *De Bruijn*.
Nel 1946, il matematico olandese Nicolaas de Bruijn [19] si interessò al "*problema delle superstringhe*" ovvero trovare una "*superstringa*" circolare più breve che contenga tutte le possibili "*sottostringhe*" di lunghezza k (**k-mers**) su un dato alfabeto.

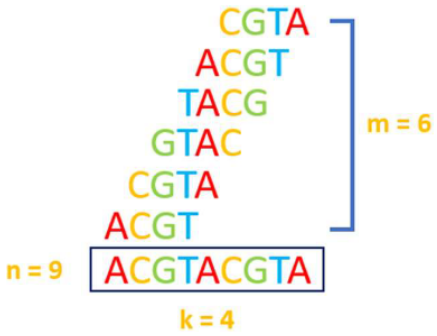


Figura 3.15: Esempio di estrazione dei k-mers.

Esistono n^k *k-mers* in un alfabeto contenente n simboli: per esempio, dato l'alfabeto composto da A, T, G e C, ci sono $4^3 = 63$ trinucleotidi. Ma come si può costruire una simile superstringa per tutti i k-mers nel caso di un valore arbitrario di k e di un alfabeto arbitrario? De Bruijn ha risposto a questa domanda prendendo in prestito la soluzione di Eulero del problema dei *ponti di Königsberg*[2]. Si costruisce un grafo B per il quale ogni possibile *(k - 1)-mer* è assegnato a un nodo; si collega un *(k - 1)-mer* con un arco

[2] https://www.scientificamerican.com/article/how-the-seven-bridges-of-koenigsberg-spawned-new-math/

diretto a un secondo *(k - 1)-mer* se esiste un *k-mer* il cui prefisso è il primo e il cui suffisso è il secondo. Gli archi del grafo di De Bruijn rappresentano tutti i possibili k-mer, e quindi un *ciclo euleriano*[3] in B rappresenta una superstringa più breve che contiene ogni *k-mer* esattamente una volta.

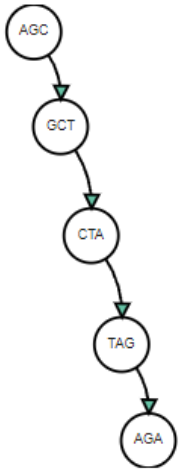


Figura 3.16: Esempio di grafo di De Bruijn sulla sequenza AGCTAGAT con k-mers = 4.

Il grafo di De Bruijn [20] è stato introdotto per la prima volta nel contesto della bioinformatica da *Idury* e *Waterman* (1995) ed è stato successivamente applicato al contesto dell'assemblaggio del genoma da *Pevzner*(*2001*). Da questa applicazione iniziale, il grafo è diventato onnipresente nell'assemblaggio e nell'analisi dei dati di sequenziamento di nuova generazione.

[3]Un ciclo euleriano in un grafo è un cammino che passa per tutti gli archi una ed una sola volta. (Fonte: *Wikipedia*)

Capitolo 4

A GNN-based Gene fusion detection method

Nel presente capitolo, verrà fornita una descrizione dettagliata della fase implementativa della soluzione proposta per la detection delle sequenze chimeriche. Saranno presentati tutti i vari step eseguiti durante il processo, offrendo una panoramica completa dell'implementazione.

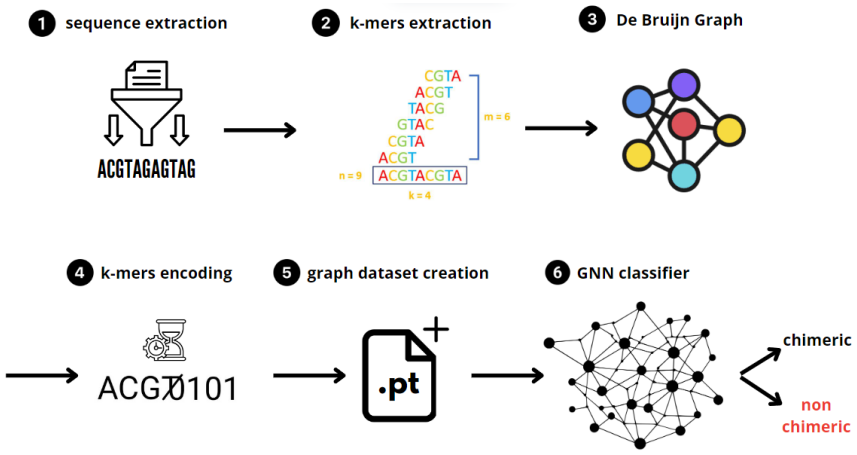


Figura 4.1: Pipeline dell'implementazione.

Il processo implementativo è stato suddiviso in due pipeline parallele, ognuna delle quali utilizza una diversa tecnica di rappresenzazione delle sequenze geniche. La prima pipeline sfrutta le capacità del modello *DNABERT* per generare delle rappresentazioni numeriche, dette *embeddings*, delle sequenze.

L'altra pipeline, invece, basata su un approccio più semplice, utilizza il *One-Hot encoding* per tradurre i nucleotidi in vettori binari fornendo una rappresentazione meno complessa ma allo stesso tempo più efficiente in termini compitazionali.

La scelta di questa suddivisone nasce dalla necessità di confrontare due metodolgie di sviluppo complementari e valutare quale delle due si comporti meglio al fine di distiguere sequenze chimerche da sequenze non chimercihe. Nelle sezioni successive verranno descritte nel dettaglio entrambe le pipeline, illustrando i passi principali e le tecniche adottate.

Tutto il codice dell'implemetazione è disponibile online, alla seguente repository https://github.com/gisolfi02/GeneFusion.

4.1 Pipeline basata su DNABERT

Il primo passo della pipeline è quello relativo all'estrazione delle sequenze geniche. A tal proposito sono stati utilizzati due datset distinti, uno contenete sequenze chimeriche e un altro contentente sequenze non chimeriche. Tali dataset sono stati generati artificialmente utilizzando il tool *Fusim*[1] utilizzando *100* geni distinti e successivamente salvati in due file in formato *FASTQ*[2].

Per poter estrarre le sequenze dai dataset è stata implementata una funzione *estrai_sequenze_geni* che prende in input un file in formato *FASTQ* ed estrae le sequenze ignorando l'identificativo della sequenza. Inoltre, poichè il dataset contenente le sequenze chimeriche presentava un numero eccessivamente maggiore rispetto a quello contenente le sequenze non chimeriche, è stato posto un limite al numero di sequenze estraibili, pari al numero massimo di sequenze non chimeriche, per ridurre il rischio di overfitting dovuto allo sbilanciamento delle classi durante l'addestramento del modello.

```
1 def get_kmer(sequence, k=4):
2     kmers = []
```

[1] Fusim è un programma per simulare trascrizioni di fusione da un genoma di riferimento. (Fonte: *Fusim*)

[2] Il formato FASTQ è un formato basato su testo per memorizzare sia una sequenza biologica che i relativi punteggi di qualità. (Fonte: *Wikipedia*)

```
    for i in range(0, len(sequence)):
        if len(sequence[i:i + k]) != k:
            continue
        kmers.append(sequence[i:i + k])
    return kmers


def get_debruijn_edges(kmers):
    edges = set()
    for k1 in kmers:
        for k2 in kmers:
            if k1 != k2:
                if k1[1:] == k2[:-1]:
                    edges.add((k1, k2))
                if k1[:-1] == k2[1:]:
                    edges.add((k2, k1))
    return edges
```

Code Listing 4.1: Funzioni get_kmers e get_debruijn_edges

Successivamente, per ogni sequenza sono stati generati i *k-mers* utilizzando la funzione *get_kmers*, listato 4.1, la quale prende in input la sequenza, la dimensione k dei *k-mer* e genera tutte le possibili sottosequenze di dimensione k scorrendo lungo la sequenza. Ottenuta la lista di *k-mers* sono stati creati quelli che sono gli archi del grafo di *De Bruijn* attraverso la funzione *get_debruijn_edges*,. La funzione costruisce gli archi del grafo collegando ogni k-mer a un altro se le loro sottosequenze si sovrappongono, in pratica, crea un arco da k_1 a k_2 se la parte finale di k_1 coincide con la parte iniziale di k_2, e viceversa.

A partire dagli archi viene costruita la matrice di adiacenza del grafo utilizzando la funzione *create_adjaceny_matrix* che prende i input gli archi e restituisce l'iniseme ordinato dei nodi e la matrice di adiacenza. Questa matrice descrive le connessioni tra i nodi del grafo, indicando quali *k-mer* sono connessi tra loro.

```
BERT_model = BertForSequenceClassification.from_pretrained("./
    bert_model")
model_name = "zhihan1996/DNA_bert_6"
tokenizer = BertTokenizer.from_pretrained(model_name)

def get_feature_matrix(nodes):
    BERT_model.eval()
```

```
7
8       features_matrix = []
9       for node in nodes:
10          # Sposta gli input sulla GPU
11          inputs = tokenizer(node, return_tensors="pt",
    max_length=128, padding='max_length', truncation=True).to(
    device)
12
13          # Estrai embedding BERT
14          with torch.no_grad():  # Non serve il calcolo del
    gradiente
15              outputs = BERT_model(**inputs, output_hidden_states
    =True)
16
17          # Ottieni gli embedding dal livello nascosto (hidden
    states)
18          hidden_states = outputs.hidden_states
19          last_hidden_state = hidden_states[-1]
20          embedding = last_hidden_state.mean(dim=1)  # Puoi usare
     mean pooling sull'intera sequenza
21          # Sposta il risultato sulla CPU per appendere
22          features_matrix.append(embedding.squeeze(0).cpu().numpy
    ())
23
24      return features_matrix
```

Code Listing 4.2: Funzione get_feature_matrix

Sempre utilizzando gli archi del grafo viene costruita la matrice delle feauture del grafo attraverso la funzione *get_feature_matrix*, mostrata nel listato 4.2, la quale utilizza un modello *DNABERT* precedentemente addestrato per ottenere gli embedding di ciascun nodo. Per ogni nodo del grafo, il tokenizer prepara i dati in modo compatibile con il modello, includendo padding e troncamento. Successivamente, il modello *DNABERT* elabora il testo e restituisce le rappresentazioni interne. La funzione estrae la rappresentazione dell'ultimo livello e calcola la media di tutte le posizioni del vettore per ottenere una singola rappresentazione del nodo. Infine, ogni vettore viene convertito in un array e inserito in una matrice di caratteristiche, che viene restituita come risultato.

```
1   def create_graph_data(sequences, chimeric):
2       data_list = []
3       k = 0
4       for seq in sequences:
5           kmers = get_kmer(seq, k=6)  # ottengo i kmer della
    sequenza
```

```
        edges = get_debruijn_edges(kmers)  # creo gli archi del
     grafo di De Bruijn
        nodes, adjacency_matrix = create_adjacency_matrix(edges
    )  # creo la matrice di adiacenza del grafo
        features_matrix = get_feature_matrix(nodes)  # creo la
    matrice delle feature dei nodi
        features_matrix = np.array(features_matrix)

        # Crea una lista vuota per memorizzare gli indici degli
     archi
        edge_index = []

        # Crea un dizionario che mappa ogni nodo al suo indice
    numerico
        node_index = {node: i for i, node in enumerate(nodes)}

        # Itera su ciascun arco nel grafo
        for edge in edges:
            # Ottieni gli indici dei nodi connessi dall'arco
            i, j = node_index[edge[0]], node_index[edge[1]]
            # Aggiungi una coppia di indici all'elenco degli
    indici degli archi
            edge_index.append([i, j])

        # Converti l'elenco degli indici degli archi in un
    tensore di PyTorch
        edge_index = torch.tensor(edge_index, dtype=torch.long)
    .t().contiguous().to(device)

        # Sposta il tensore delle feature su GPU
        x = torch.tensor(features_matrix, dtype=torch.float).to
    (device)

        # Crea un tensore di PyTorch con le etichette per ogni
    grafo
        y = torch.tensor([1 if chimeric else 0], dtype=torch.
    long).to(device)

        # Crea un oggetto Data che rappresenta un grafo
        data = Data(x=x, edge_index=edge_index, y=y)
        data_list.append(data)

        k += 1
        print(f'{k:4d} Graph created:')
        print(data)
    return data_list
```

Code Listing 4.3: Funzione create_graph_data

Il listato 4.3 mostra l'implementazione della funzione *create_graph_data* che automatizza i passaggi descritti per ciascuna sequenza e produce un dataset di grafi. In dettaglio, la funzione prende in input una lista di sequenze e un valore booleano che indica se le sequenze sono chimeriche o meno, e per ciascuna sequenza esegue i seguenti passaggi:

1. Estrae i *k-mers* di dimensione *6*
2. Crea gli archi del grafo di *De Bruijn*
3. Crea la matrice di adiacenza del grafo
4. Crea la matrice delle feature del grafo
5. Crea una lista contente gli indici degli archi e la converte in un tensore[3] *PyTorch*[4]
6. Converte la matrice delle feature in un tensore *PyTorch*
7. Crea un tensore contenente l'etichetta che specifica se la sequenza è chimerica o non
8. Crea un oggetto di tipo *Data* di *PyTorch Geometric*[5] con i tre tensori creati
9. Aggiunge l'oggetto di tipo *Data* ad una lista di oggetti *Data*

Una volta effettuati questi passi per tutte le sequenze, la funzione restituisce la lista di oggetti. Per i due dataset di sequenze (chimeriche e non chimeriche), vengono create due liste separate di grafi, che vengono successivamente viene memorizzate in due file in formato *.pt*, pronti per essere utilizzati nell'addestramento e nella validazione del modello di *Graph Neural Network*.

4.1.1 Addestramento modello DNABERT

Al fine di migliorare la capacità di classificare le sequenze geniche in chimeriche o non chimeriche, è stato addestrato un modello *DNABERT* utilizzato

[3] Nell'ambito del machine learning, un tensore è un insieme multidimensionale di numeri che funziona come un dispositivo matematico di contabilità. (Fonte: *IBM*)

[4] PyTorch è un framework di deep learning open source basato su software usato per creare reti neurali. (Fonte: *IBM*)

[5] PyTorch Geometric è una libreria costruita su PyTorch per scrivere e addestrare facilmente reti neurali grafiche (GNN) per un'ampia gamma di applicazioni relative a dati strutturati. (Fonte: *PyTorch Geometric*)

successivamente nella pipeline dalla funzione *get_feature_matrix* per estrarre gli embedding dai nodi del grafo.

Il primo passo dell'addestramento è stato l'estrazione delle sequenze dai dataset e la loro successiva suddivisione in *k-mers* di lunghezza *6*. Inoltre, alla sequenza k-merizzata è stata associata anche una *label* che la identifica in chimerica o non chimerica. Tutte le sequenze k-merizzate e le rispettive label sono poi state combinate in un unico *DataFrame* utilizzato per l'addestramento.

Una volta ottenuto il *DataFrame* è stato utilizzato un tokenizzatore per convertire le sequenze k-merizzate in un formato numerico compresibile al modello *DNABERT*. Per eseguire questa operazione è stato utilizzato il tokenizzatore del modello *DNABERT* pre-addestrato "*zhihan1996/DNA_bert_6*".

```
class DNADataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: val[idx] for key, val in self.encodings.
    items()}
        item['labels'] = self.labels[idx]
        return item

    def __len__(self):
        return len(self.labels)
```

Code Listing 4.4: Classe DNADataset

Le sequenze tokenizzate e le loro label sono state successivamente convertite in un formato compatibile con *PyTorch*, realizzando una classe *DNADataset*, mostrata nel listato 4.4, che gestisce i dati i li prepara per l'addestramento del modello.

Per addestrare il modello, è stata utilizzata la classe *Trainer* della libreria *transformers*, che facilita il processo di training. Per prima cosa, viene caricato il modello *DNABERT* "*zhihan1996/DNA_bert_6*" utilizzato per la classificazione di sequenze, con *num_labels=2* per distinguere tra sequenze chimeriche e non chimeriche. Il modello è stato addestrato per *15* epoche,

utilizzando dei mini batch di dimensione *16*. Inoltre, è stata definita una funzione *compute_metrics* che calcola l'accuratezza del modello durante la fase di valutazione.

Al termine dell'addestramento, il modello è stato salvato all'interno di una directory "*bert_model*" per poter essere ricaricato nella fase di estrazione degli embedding nella pipeline.

4.2 Pipeline basata sul One-Hot Encoding

Anche la pipeline basata sul *One-Hot Encoding* esegue gli stessi passi della pipeline precedentemente descritta. La principale differenza rispetto alla pipeline *DNABERT* emerge nella fase di creazione della matrice delle feature. Mentre la pipeline *DNABERT* utilizza un modello pre-addestrato per generare embedding complessi dei nodi, la pipeline *One-Hot* utilizza un approccio più diretto, basato sull'encoding *One-Hot*.

```
onehot = {
    'A': [1, 0, 0, 0],
    'C': [0, 1, 0, 0],
    'G': [0, 0, 1, 0],
    'T': [0, 0, 0, 1],
    'N': [0, 0, 0, 0],  # Per nucleotidi sconosciuti o gap
}

def get_feature_matrix(nodes):
    features_matrix = []

    # Itera attraverso i nodi, ciascun nodo pu  essere un k-mer (sequenza) o un singolo nucleotide
    for node in nodes:
        # Se il nodo    un k-mer, converti ciascun nucleotide in one-hot encoding e concatenali
        onehot_representation = []
        for nucleotide in node:
            onehot_representation.append(onehot.get(nucleotide, onehot['N']))

        # Aggiungi la rappresentazione one-hot completa del nodo alla matrice delle feature
        features_matrix.append(onehot_representation)
    # Converte la lista in una matrice numpy
    return np.array(features_matrix)
```

Code Listing 4.5: Funzione get_feature_matrix One-Hot Encdoding

Il listato 4.5 mostra la funzione *get_feature_matrix* che utilizza il *One-Hot Encodiding* per generare gli embeddings. Per ogni *k-mer*, viene convertito ciascun nucleotide nella sua rappresentazione one-hot utilizzando il dizionario definito. La funzione concatena le rappresentazione dei nucleotidi di un kmer in un'unica struttura e aggiunge quest'ultima in una matrice.

```
    def create_graph_data(sequences, chimeric):
    data_list = []
    k = 0
    for seq in sequences:
        kmers = get_kmer(seq, k = 6)  #ottengo i kmer della
    sequenza
        edges = get_debruijn_edges(kmers)  #creo gli archi del
    grafo di De Bruijn
        nodes, adjacency_matrix = create_adjacency_matrix(edges
    )  #creo la matrice di adiacenza del grafo
        features_matrix = get_feature_matrix(nodes)  #creo la
    matrice delle feature dei nodi, che continene per ogni riga
     l'encoding BERT del nodo

        # Crea una lista vuota per memorizzare gli indici degli
     archi
        edge_index = []

        # Crea un dizionario che mappa ogni nodo al suo indice
    numerico
        node_index = {node: i for i, node in enumerate(nodes)}

        # Itera su ciascun arco nel grafo
        for edge in edges:
            # Ottieni gli indici dei nodi connessi dall'arco
            i, j = node_index[edge[0]], node_index[edge[1]]
            # Aggiungi una coppia di indici all'elenco degli
    indici degli archi
            edge_index.append([i, j])

        # Converti l'elenco degli indici degli archi in un
    tensore di PyTorch
        edge_index = torch.tensor(edge_index, dtype=torch.long)
    .t().contiguous()

        node_features = []
        for onehot_list in features_matrix:
            # Ogni k-mer ha una lista di one-hot separati (uno
    per nucleotide)
            # Li appiattiamo lungo la prima dimensione (
```

```
    concatenazione lungo l'asse orizzontale)
30          flattened = torch.cat([torch.tensor(vec, dtype=
    torch.float) for vec in onehot_list], dim=0)
31          node_features.append(flattened)
32
33      # Convertiamo la lista di feature dei nodi in un
    tensore
34      x = torch.stack(node_features)
35
36      # Converte la matrice delle feature in un tensore di
    PyTorch
37      #x = torch.tensor(features_matrix, dtype=torch.float)
38
39      # Crea un tensore di PyTorch con le etichette per ogni
    nodo
40      y = torch.tensor([1 if chimeric else 0], dtype=torch.
    long)
41
42      # Crea un oggetto Data che rappresenta un grafo
43      data = Data(x=x, edge_index=edge_index, y=y)
44      data_list.append(data)
45
46      k += 1
47      print(f'{k:4d} Graph created: ')
48      print(data)
49  return data_list
```

Code Listing 4.6: Funzione create_graph_data One-Hot Encdoding

La funzione *create_graph_data*, mostrata nel listato 4.6, viene modifcata per adattarsi alla nuova funzione *get_feature_matrix*. In particolare, ogni riga della matrice delle feature, poichè composta da una lista di vettori, viene appiattita e i vettori vengono concatenati per creare una singola rappresentazione numerica.
Anche in questo caso, per i due dataset di sequenze vengono create liste separate e salvate in due file in formato *.pt*.

4.3 Pipeline del modello di Graph Neural Network

Dopo la costruzione dei dataset di grafi utilizzando la pipeline *DNABERT* o *One-Hot Encoding*, si passa alla fase di addestramento di un modello di *Graph Neural Network* per la classificazione delle sequenze. A tale scopo, sono stati implementati diversi modelli di *GNN* per valutare quale modello si presta meglio nella rilevazione delle sequenze chimeriche.

```
class GCN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels,
    out_channels, dropout = 0.5):
        super(GCN, self).__init__()

        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
        self.dropout = torch.nn.Dropout(dropout)
        self.lin = torch.nn.Linear(hidden_channels,
    out_channels)

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.
    batch

        # Passaggio attraverso i livelli GCN
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.dropout(x)
        x = self.conv2(x, edge_index)
        x = F.relu(x)
        x = self.dropout(x)



        # Pooling globale (media) per ottenere un vettore per
    ogni grafo
        x = global_mean_pool(x, batch)

        # Classifier
        x = self.lin(x)
        return x

class GAT(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels,
    out_channels, heads = 8):
        super(GAT, self).__init__()
        self.gat1 = GATv2Conv(in_channels, hidden_channels,
    heads=heads)
        self.gat2 = GATv2Conv(hidden_channels * heads,
    hidden_channels, heads=1)
        self.lin = torch.nn.Linear(hidden_channels,
    out_channels)

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.
    batch

```

```
        # Passaggio attraverso i livelli GAT
        x = F.dropout(x, training=self.training, p=0.6)
        x = self.gat1(x, edge_index)
        x = F.elu(x)
        x = F.dropout(x, training=self.training, p=0.6)
        x = self.gat2(x, edge_index)
        x = F.elu(x)

        x = global_mean_pool(x, batch)  # Pooling globale (
    media) per ottenere un vettore per ogni grafo

        x = self.lin(x)  # Classifier

        return x

class SAGE(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels,
    out_channels):
        super(SAGE, self).__init__()
        self.sage1 = SAGEConv(in_channels, hidden_channels)
        self.sage2 = SAGEConv(hidden_channels, hidden_channels)
        self.lin = torch.nn.Linear(hidden_channels,
    out_channels)

    def forward(self, data):
        x, edge_index, batch = data.x, data.edge_index, data.
    batch

        # Passaggio attraverso i livelli SAGE
        x = F.dropout(x, training=self.training, p=0.6)
        x = self.sage1(x, edge_index).relu()
        x = F.relu(x)
        x = F.dropout(x, training=self.training, p=0.6)
        x = self.sage2(x, edge_index)
        x = F.relu(x)

        x = global_mean_pool(x, batch)  # Pooling globale (
    media) per ottenere un vettore per ogni grafo

        x = self.lin(x)  # Classifier

        return x
```

Code Listing 4.7: Modelli GCN - GAT - GraphSAGE

Nel listato 4.7 è mostrata l'implementazione dei modelli, che sono:

- **Graph Convolutional Network**: il modello è stato implementato utilizzando la classe *GCNConv* messa a disposizione dalla libreria

PyTorch Geometric. É compopsto da due livelli di convoluzione e un livello finale di classificazione

- **Graph Attention Network**: Per questo modello è stata utilizzata la classe *GATv2Conv*, sempre di *PyTorch Geometric*. É composto da due livelli convoluzione con meccanismi di attenzione e un livello finale di classificazione

- **GraphSAGE**: implementato utilizzando la classe *SAGEConv* messa a disposizione dalla libreria *PyTorch Geometric*. Anche questo modello è costituito da due livelli di convoluzione e un livello finale di classificazione.

Ogni modello prende in input:

- *in_channels*: il numero di feature per nodo, che sono *768* nel caso di *DNABERT* e *24* nel caso del *One-Hot Encoding*,

- *hidden_channels*: il numero di neuroni nello strato nascosto, *128* per *DNABERT* e *16* per *One-Hot Encoding*,

- *out_channels*: l'output finale, uguale a *1* in quanto si tratta di una probabilità che indica se la sequnza è chimericha o meno.

Il processo di addestramento parte dal caricamento di una delle due coppie di dataset realizzati dalle pipeline precedenti. Caricati i dataset, questi vengono concatenati in un unico grande dataset, che viene poi suddiviso in tre parti utilizzate per il **training**, la **validazione** e il **testing**, con una proporzione di *80%, 10% e 10%* rispettivamente. Questa suddivisione viene effettuata in modo casuale per ogni ciclo di addestramento.

Ogni suddivisione è stata poi caricata in un *DataLoader*, il quale permette di iterare sui dati a piccoli batch. Successivamente, viene scelto uno dei modelli da utilzzare e lo si inizzializza con i parametri specificati.
Il modello viene addestrato per *200* epoche, utilizzando una funzione di perdita *Binary Cross-Entropy With Logit Loss* e un ottimizzatore *Adam* con un tasso di apprendimento pari a *0.01*. In ogni epoca, i dati vengono passati attraverso il modello, calcolando le perdite e aggiornando i pesi per minimizzare l'errore.

Dopo l'addestramento, si procede alla validazione e al testining del modello utilizzando i rispettivi dataset. Inoltre, durante la fase di testing

vengono calcolate diverse metriche per valutare le prestazioni del modello: *accuracy, precision, F1-score, recall* e *ROC-AUC*. Queste metriche vengono poi salvate in un file *.txt* per tenere traccia delle prestazioni di ciascun modello e facilitare il confronto.

Per garantire la robustezza e l'affidabilità dei modelli, l'addestramento di ciascun modello è stato eseguito più volte utilizzando i diversi dataset, *DNA-BERT* o *One-Hot*, con differenti suddivisoni di questi ultimi. Questa strategia ha permesso di monitorare l'andamento delle prestazioni del modello, ottenendo una stima più affidabile, riducendo la variabilità dei risultati dovuta alla selezione casuale dei dati e fornendo una visione più completa della sua capacità di generalizzare.

Capitolo 5

Risultati ottenuti

Nel presente capitolo verranno presentati e analizzati i risultati ottenuti dall'addestramento dei modelli per entrambe le pipeline sviluppate. Per ciascuna pipeline, ci concentreremo sui risultati del miglior modello, selezionato in base alle metriche di *accuratezza, precisione, F1-score, recall e AUC*. Infine, confronteremo questi risultati con quelli ottenuti dal metodo *sequence-based* , al fine di valutare i vantaggi e le eventuali limitazioni dei modelli basati su grafi rispetto all'approccio sequenziale.

5.1 Risultati One-Hot Encoding

Tra i tre diversi modelli implementati per la rilevazione delle fusioni geniche, il *Graph Attention Network* (*GAT*) ha ottenuto le prestazioni migliori utilizzando il dataset ottenuto dalla pipeline basta sul *One-Hot encoding*. La Tabella 5.1 mostra le misure di performance del modello.

Metrica	Valore
Accuracy	93.19
Precision	91.75
F1 score	93.19
Recall	94.68

Tabella 5.1: Metriche ottenute dal modello GAT

L'accuracy indica che il modello ha classificato correttamente il *93.19%* delle sequenze totali. La precision indica invece la percentuale di sequenze classificate come *chimeriche* che risultano esserlo effettivamente; un valore del *91.75%* indica che il modello commette pochi errori di *falso positivo*. Il recall

rappresenta la percentuale di sequenze chimeriche che il modello è riuscito ad identificare; un valore pari al *94.68%* significa che il modello genera pochi *falsi negativi*. Infine, l'F1 score è la media armonica tra precision e recall; un punteggio di *93.19%* indica un buon equilibrio, suggerendo che il modello è sia accurato nel predire le sequenze positive sia capace di identificare la maggior parte delle sequenze positive.
Nella Figura 5.1 è riportata la matrice di confusione che fornisce una rappresentazione visuale dei risultati del modello. In questa matrice di dimensione 2×2 ogni riga rappresenta la classe reale, mentre ogni colonna rappresenta la classe predetta dal modello.

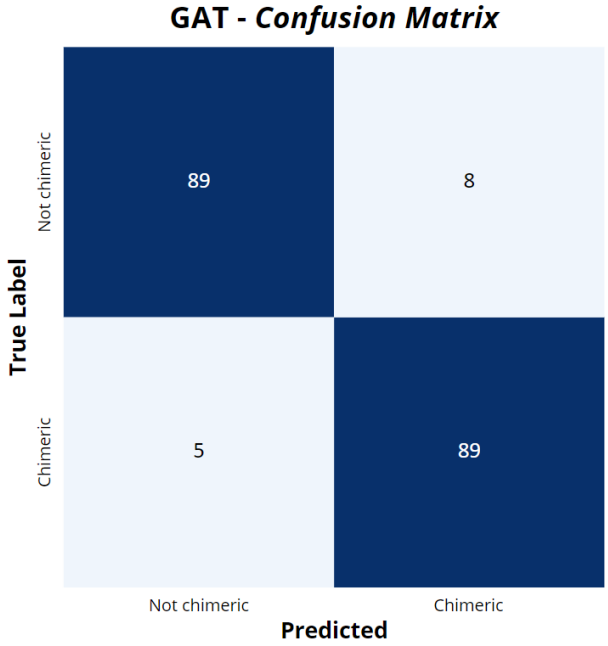


Figura 5.1: Matrice di confusione sui dati di test del modello GAT

Possiamo osservare che il modello riesce a raggiungere buoni risultati in entrambe le classi, con un numero significativo di predizioni corrette. La diagonale principale della matrice presenta valori elevati, indicando una buona capacità del modello di discriminare tra le due classi. Nonostante ciò, il modello presenta *8* casi di *falsi positivi*, che seppur pochi, in un contensto reale potrebbero rappresentare un problema e quindi sarebbe necessario ridurre al minimo questo valore.
Oltre alla matrice di confusione, la Figura 5.2 mostra il grafico della curva

ROC (*Receiver Operating Characteristic*) e l'*AUC* (*Area Under the Curve*). Questo grafico fornisce una visualizzazione delle performance del modello di classificazione.

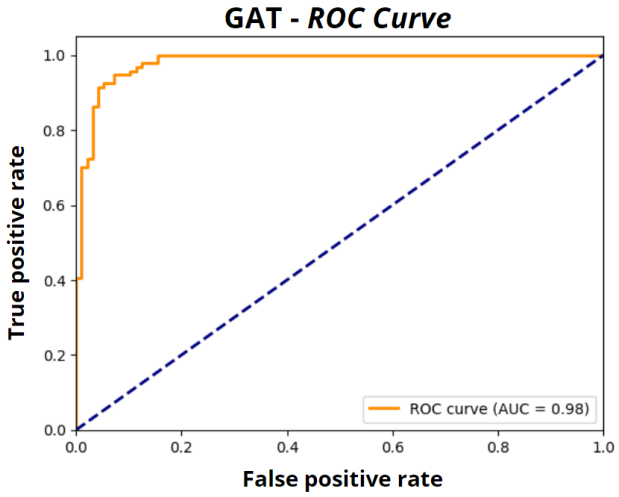


Figura 5.2: Curva ROC sui dati di test del modello GAT

La curva *ROC* è una rappresentazione grafica della capacità di discriminazione di un modello al variare della soglia di classificazione. Graficamente, la curva ROC è disegnata tracciando il *tasso di vero positivo* (*TPR*) sull'asse delle ordinate e il *tasso di falso positivo* (*FPR*) sull'asse delle ascisse. La *TPR* rappresenta la sensibilitò del modello, ovvero la capacità di identificare correttamente gli esempi positivi, mentre la *FPR* rappresenta la proporzione di falsi positivi rispetto agli esempi negativi. Una curva *ROC* ideale si avvicina all'angolo in alto a sinistra, indicando una sensibilità elevata e una bassa proporzione di falsi positivi. L'*AUC* è invece l'area sottesa dalla curva *ROC*. Essa misura la capacità di discriminazione complessiva del modello, fornendo un valore compreso tra *0* e *1*. Un valore di AUC pari a *1* indica una perfetta capacità di discriminazione, mentre un valore pari a *0.5* indica una capacit'a di discriminazione casuale.
Dalla figura possiamo osservare che il modello ha ottenuto un'ottima performance di classificazione. La curva *ROC* si avvicina all'angolo in alto a sinistra del grafico, indicando che il modello ha una buona capacità di discriminazione tra le classi. Inoltre, l'*AUC* ha un valore elevato, pari a

0.98, il che conferma che il modello ha una capacità di classificazione molto accurata.

5.2 Risultati DNABERT

Per quanto riguarda la pipeline basata su *DNABERT*, il modello che ha ottenuto i risultati migliori è stato il *Graph Convolutional Network* (*GCN*). La Tabella 5.2 mostra i valori delle metriche ottenuti dal modello.

Metrica	Valore
Accuracy	96.34
Precision	95.70
F1 score	96.22
Recall	96.74

Tabella 5.2: Metriche ottenute dal modello GCN

Possiamo osservare che il modello ha avuto risultati migliori rispetto al modello della pipeline basata sul *One-Hot Encoding*. Questo suggerisce che *DNABERT* riesce a catturare meglio le informazioni presenti nelle sequenze nucleotidiche.
Di seguito, la Figura 5.3 rappresenta la matrice di confusione del modello. Anche in questo caso, possiamo osservare che il modello ha ottimi risultati nella classificazione, con un margine di errore ridotto ripetto all'altro modello. Infatti, il modello persenta soltanto un caso di *falso positivo*, rispetto agli *8* del modello predecente, suggerendo che il modello riesce a classificare meglio le sequenze chimeriche.
Infine, nella Figura 5.4 è illustrato il grafico della curva *ROC* e *AUC*. In questo caso, la curva *ROC* si avvicina maggiormenente all'angolo in alto a sinistra, indicando che il modello ha una migliore capacità di discriminazione delle due classi. Inoltre, il valore dell'*AUC* è uguale al valore ottenuto dal modello precedente, quindi entrambi i modelli una capacità di classificazione simile.

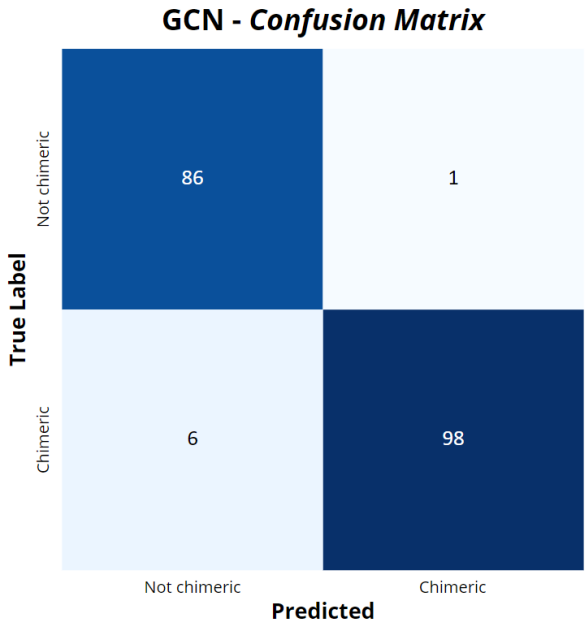


Figura 5.3: Matrice di confusione sui dati di test del modello GCN

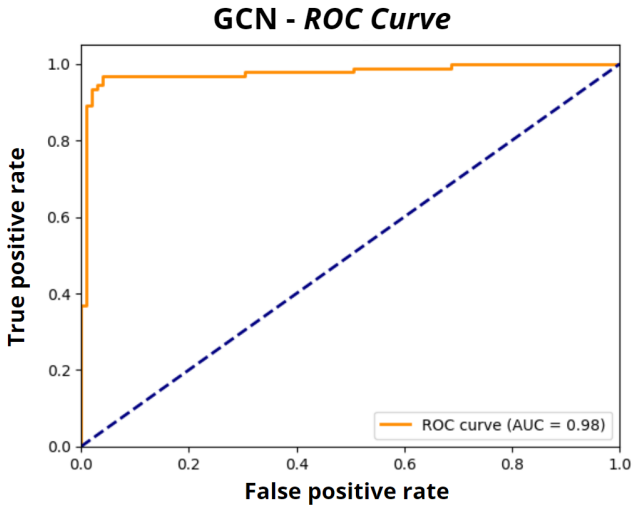


Figura 5.4: Curva ROC sui dati di test del modello GCN

5.3 Confronto con il metodo sequence-based

In questa sezione saranno presentati i risultati ottenuti dal metodo *sequence-based* discusso nel Capitolo 2 e verranno confrontati con quelli dei modelli realizzati in questo lavoro di tesi.
La Tabella 5.3 riporta i valori delle metriche ottenute dal modello *Fusion classifier* utilizzato nel metodo *sequence-based*.

Metrica	Valore
Accuracy	81.82
Precision	78.64
F1 score	81.82
Recall	79.29

Tabella 5.3: Metriche ottenute dal Fusion classifier

Come si evince dalla tabella, il modello *Fusion classifier* ha ottenuto delle prestazioni nettamente inferiori rispetto ai modelli basati sulle *Graph Neural Network*. Questi risultati suggeriscono che i modelli proposti sono in grado di identificare con maggiore precisione le sequenze chimeriche e non chimeriche, riducendo al minimo il numero di falsi positivi e negativi, garantendo di conseguenza maggiore affidabilità e robustezza, che rappresentano dei requistti chiave in un contesto reale.

Capitolo 6

Conclusioni

Negli ultimi decenni, la Bioinformatica è diventata un pilastro fondamentale nella ricerca genomica, grazie alla sua capacità di integrare biologia molecolare, informatica e statistica per analizzare e interpretare dati biologici complessi. In ambito oncologico, la rilevazione delle fusioni geniche riveste un ruolo cruciale. Le fusioni geniche, eventi in cui segmenti di *DNA* provenienti da geni diversi si fondono dando origine a *RNA chimerici*, sono spesso coinvolte nella *tumorigenesi* e rappresentano importanti biomarcatori per la diagnosi e il monitoraggio del cancro. Tuttavia, i metodi di rilevamento tradizionali, basati principalmente sull'allineamento delle sequenze, presentano diverse limitazioni, tra cui un elevato tasso di falsi positivi e una difficoltà nell'identificazione di fusioni complesse, che ne riducono l'affidabilità.
In questo lavoro di tesi, allo scopo di superare le limitazioni dei sistemi esistenti, vengono proposte delle soluzioni *graph-based* attraverso l'utilizzo delle *GNN*, un'alternativa promettente per la rilevazione delle fusioni geniche. L'analisi sperimentale ha dimostrato che i modelli proposti ottengono delle prestazioni notevoli, indicando una maggiore affidabilità, nonchè un basso tasso di errori, quali falsi negativi o posistivi, rispetto ai metodi tradizionali.

6.1 Limitazioni

Nonostate i risultati promettenti, questo lavoro presenta alucne limitazioni che meritano attenzione. Per prima cosa, la complessità computazionale delle *GNN* richiede delle risorse elevate. In particolare, i modelli che usano *DNABERT* per estrarre gli embedding richiedono un costo computazionale in termini di memoria e tempo di elaborazione elevato. Questa caratteristica

rende questi modelli difficili da utilizzare in ambienti con risorse limtate, come piccoli laboratori, che non dispongono di potenti unità di elaborazione.

Un'altra limitazione della soluzione propsota è la mancanza di una tecnica di interpretabilità dei risultati e di come le caratteristiche genetiche influiscano su tali risultati. Sebbene le Graph Neural Networks siano efficaci nel catturare strutture complesse, le loro previsioni risultano spesso difficili da spiegare in modo intuitivo. In ambito clinico, la mancanza di interpretabilità può ostacolare l'adozione di queste tecnologie, poiché i medici e i biologi potrebbero preferire modelli che forniscano informazioni chiare sulle ragioni delle previsioni fatte.

6.2 Sviluppi futuri

Alla luce delle conclusioni tratte, lo studio apre diverse oppurtunità di sviluppo per il futuro. Un primo aspetto riguarda l'ottimizzazione del carico computazionale dei modelli, attraverso l'applicazione di tecniche di compressione, come la quantizzazione, che potrebbero ridurre la complessità dei modelli, rendedoli così più efficienti.

Un altro aspetto è la valutazione dei modelli su campioni biologici reali provenienti da pazienti. Valutare le prestazioni dei modelli su questi campioni permetterebbe di analizzare la loro efficacia in scenari clinici concreti.

Per favorire l'adozione delle *GNN* in ambito clinico, è cruciale che le decisioni del modello siano trasparenti e interpretabili. Un interessante sviluppo futuro riguarda l'implementazione di tecniche di *explainability*, cioè di interpretabilità, che possano spiegare quali regioni delle sequenze genomiche influenzano maggiormente le previsioni del modello.

In conclusione, questo lavoro di tesi ha mostrato come le *GNN* rappresentano un progresso significativo nella rilevazione delle fusioni geniche. Sebbene siano necessari ulteriori sviluppi e validazioni, la seguente tesi rappresenta un passo avanti verso una diagnostica più affidabile.

Bibliografia

[1] Qingguo Wang, Junfeng Xia, Peilin Jia, William Pao, and Zhongming Zhao. Application of next generation sequencing to human gene fusion detection: computational tools, features and perspectives. *Briefings in Bioinformatics*, 14(4):506–519, 08 2012.

[2] Natasha S. Latysheva and M. Madan Babu. Discovering and understanding oncogenic gene fusions through data intensive computational approaches. *Nucleic Acids Research*, 44(10):4487–4503, 04 2016.

[3] P Bonizzoni, De Felice, Y Pirola, R Rizzi, R Zaccagnino, and R Zizza. Identification of chimeric rnas: a novel machine learning perspective. *Unimib.it*, 03 2024.

[4] Brian J. Haas, Alex Dobin, Nicolas Stransky, Bo Li, Xiao Yang, Timothy Tickle, Asma Bankapur, Carrie Ganote, Thomas G. Doak, Nathalie Pochet, Jing Sun, Catherine J. Wu, Thomas R. Gingeras, and Aviv Regev. Star-fusion: Fast and accurate fusion transcript detection from rna-seq. *bioRxiv*, 2017.

[5] Daehwan Kim and Steven L Salzberg. Tophat-fusion: an algorithm for discovery of novel fusion transcripts. *Genome Biology*, 12:R72, 2011.

[6] Matteo Benelli, Chiara Pescucci, Giuseppina Marseglia, Marco Severgnini, Francesca Torricelli, and Alberto Magi. Discovering chimeric transcripts in paired-end RNA-seq data by using EricScript. *Bioinformatics*, 28(24):3232–3239, 10 2012.

[7] Huanying Ge, Kejun Liu, Todd Juan, Fang Fang, Matthew Newman, and Wolfgang Hoeck. FusionMap: detecting fusion genes from next-generation sequencing data at base-pair resolution. *Bioinformatics*, 27(14):1922–1928, 05 2011.

[8] Maxwell W. Libbrecht and William Stafford Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 16:321–332, 05 2015.

[9] Rocco Zaccagnino, Gerardo Benevento, Paola Bonizzoni, Clelia De Felice, Delfina Malandrino, Yuri Pirola, Raffaella Rizzi, and Rosalba Zizza. Advancing machine learning from sentence to graph-based representations for chimeric read detection.

[10] Rene Y. Choi, Aaron S. Coyner, Jayashree Kalpathy-Cramer, Michael F. Chiang, and J. Peter Campbell. Introduction to Machine Learning, Neural Networks, and Deep Learning. *Translational Vision Science Technology*, 9(2):14–14, 02 2020.

[11] Cedric Seger. An investigation of categorical variable encoding techniques in machine learning: Binary versus one-hot and feature hashing, 2018.

[12] Solveig Badillo, Balazs Banfai, Fabian Birzele, Iakov I. Davydov, Lucy Hutchinson, Tony Kam-Thong, Juliane Siebourg-Polster, Bernhard Steiert, and Jitao David Zhang. An introduction to machine learning. *Clinical Pharmacology & Therapeutics*, 107(4):871–885, 2020.

[13] Fan Liang, Cheng Qian, Wei Yu, David Griffith, and Nada Golmie. Survey of graph neural networks and applications. *Wireless Communications and Mobile Computing*, 2022(1):9261537, 2022.

[14] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2022.

[15] Yanrong Ji, Zhihan Zhou, Han Liu, and Ramana V Davuluri. DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome. *Bioinformatics*, 37(15):2112–2120, 02 2021.

[16] David Fenstermacher. Introduction to bioinformatics. *Journal of the American Society for Information Science and Technology*, 56(5):440–446, 2005.

[17] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. From dna to rna, 2002.

[18] Hao Wu, Xiaorong Li, and Hui Li. Gene fusions and chimeric rnas, and their implications in cancer. *Genes Diseases*, 6(4):385–390, 2019.

[19] Phillip E C Compeau, Pavel A Pevzner, and Glenn Tesler. How to apply de bruijn graphs to genome assembly. *Nature Biotechnology*, 29:987–991, 11 2011.

[20] Victoria G Crawford, Alan Kuhnle, Christina Boucher, Rayan Chikhi, and Travis Gagie. Practical dynamic de Bruijn graphs. *Bioinformatics*, 34(24):4189–4195, 06 2018.

Ringraziamenti

Bene, è arrivato per me il momento più difficile di tutta la stesura di questa tesi, il momento dei ringraziamenti. Non sono molto bravo ad esternare i miei sentimenti, e questo rende questa parte un lavoro estenuante, che mi farà spercare molte energie, più di quante io ne abbia sprecate in questi utlimi tre anni di studio.

La prima persona che desidero ringraziare è il mio relatore, il Prof. *Rocco Zaccagnino*, per avermi accompagnato durante questo percorso con competenza, gentilezza, simpatia e disponibilità. Il suo modo unico di insegnare mi ha colpito profondamente, distinguendolo dagli altri docenti e spingendomi a scegliere di essere seguito proprio da lui. Mi ha accolto in laboratorio con grande calore, elogiandomi davanti a professori e studenti, un gesto che per me è stato tanto inaspettato quanto significativo. Grazie di cuore per tutto.

Un'altra persona cruciale per questo lavoro di tesi, che merita tutta la mia gratitudine, è il Dott. *Gerardo Benevento*. Senza il suo supporto, questo lavoro non sarebbe stato possibile. Grazie per i preziosi consigli, che si sono rivelati fondamentali, per la tua costante disponibilità e per avermi aiutato in ogni situazione, anche quando il problema non rientrava nelle tue competenze. Grazie, inoltre, per i bei momenti di leggerezza e risate trascorsi in laboratorio, hanno reso questo percorso ancora più piacevole.

Questi tre anni non sono stati solo un periodo di studio, ma per la maggior parte anni di divertimento, risate e serate interminabili passate a parlare di qualsiasi argomento che la mente umana possa concepire, a giocare, a giudicare le persone e persino a prendersi cura di chi aveva una forte dipendenza dall'alcol. Tutto questo non sarebbe stato possibile senza aver incontrato il gruppo più disparato, complessato, rumoroso, caotico, imprevedibile, confusionario, esuberante, scatenato, stravagante, chiassoso, incontenibile, sopra le righe, variegato, energico, eclettico, travolgente e instancabile che, nonostante

tutto, non posso fare a meno di considerare la mia seconda famiglia. Sì, sto proprio parlando di voi, ragazzi di Banco. Non so proprio da dove iniziare, quindi la prima cosa che dirò è KOSAAAA?.

Gennaro, mio fratello, la nostra infanzia non è stata facile: mamma ha divorziato da papà e ha dovuto crescerci da sola. Era evidente che tu fossi il suo preferito, e questo mi ha fatto provare un po' di gelosia nei tuoi confronti. Tuttavia, crescendo, quel sentimento è svanito, e oggi ritengo che il nostro rapporto sia straordinario. Sei una persona geniale con un'intelligenza fuori dal comune, adoro quei momenti in cui inizi a spiegare un argomento stranissimo che solo tu conosci, quelli che qualcuno chiamerebbe "momenti dell'autismo". A parte gli scherzi, ti ritengo un amico prezioso. La tua bontà, la tua spontaneità e il tuo essere genuino sono qualità rare, e per questo ringrazio il giorno in cui ti sei seduto accanto a noi. Un amico come te era esattamente ciò di cui il nostro gruppo aveva bisogno. Infine, grazie per tutte le serate passate a giocare, sono stati momenti di puro divertimento, che porterò sempre con me. Ti voglio lasciare con una frase: "Ma quanti anni avete, 10?"

Giacomo, essendo questo un documento ufficiale non posso scrivere quello che avrei detto realmente ma sono costretto a scrivere delle belle parole nei tuoi confronti (che non saranno vere). Nonostante non ci capiti molto spesso di parlare di noi stessi, ti considero comunque un ottimo amico, uno di cui non potrei fare a meno. Credo che una buona amicizia non si basi soltanto sulle parole, ma soprattutto sulla capacità di contare l'uno sull'altro. Anche se passiamo la maggior parte del tempo ad insultarci e offenderci, so che, se avessi un problema, tu ci saresti e faresti del tuo meglio per aiutarmi. Perchè, dietro a quella faccia da vipera che giudica chiunque, questa è la persona che sei, una persona buona, che cerca sempre di aiutare tutti, anche chi in realtà non lo meriterebbe (non facciamo nomi). So che non stai attraversando un buon momento ultimamente, ma io ti auguro di riuscire a superare queste difficoltà e di ritornare ad essere il Giacomo di una volta, quello con la pelle e gli occhi gialli.

Giovanni, tu rappresenti la nostra nota orientale all'interno del gruppo, perché siamo un gruppo inclusivo e accogliamo tutti a braccia aperte, a prescindere da qualsiasi differenza. Penso che non sia stato facile per te ambientarti in un posto così diverso da quello da cui provieni, ma voglio che tu sappia che noi siamo qui per questo, per farti sentire a casa. Sei una persona sempre disponibile, che tiene al bene del gruppo e con cui si può

scherzare liberamente, senza il rischio di offenderti. So che per te non è sempre scontato passare dei bei momenti, ma quando sei con noi, sono sicuro che sarà speciale. Spero davvero che anche per te questa sensazione possa diventare la normalità. Come ultima cosa, voglio dirti che le tue risposte sul gruppo sono sempre imprevedibili e ogni volta mi fanno morire dalle risate. Proprio come sentire nelle cuffie, mentre giochiamo, l'audio a pieno volume dei TikTok che guardi, che mi piacerebbe sentire più spesso.

Antonio, persona dall'ambiguo orientamento sessuale, in tre anni che ci conosciamo non mi hai mai mostrato la tua collezione di rame rubato (lo so che ce l'hai, tutti quelli come te la hanno). Sei stata la prima persona del gruppo che ho conosciuto e non avrei mai pensato che esistesse qualcuno tanto stupido quanto me e proprio per questo motivo, siamo entrati subito in sintonia. Le nostre conversazioni non sono come semplici chiacchierate: parliamo principalmente utilizzando solo insulti, che fortunatamente nessuno legge, altrimenti non saremmo qui. Il nostro appuntamento fisso ogni sera su Discord, tra partite, risate e tu che vieni bannato per qualche insulto di troppo (in fondo, sei solo un ragazzo chill che si arrabbia facilmente quando gli altri giocano male), mi ricorda che anche se non ci vediamo spesso, sei sempre presente per me.

Tresy, aspettavi questo momento da un sacco di tempo e non hai fatto altro che assillarmi chiedendomi se avessi scritto i ringraziamenti. Sono sicuro che, mentre li sto leggendo, stai piangendo tutta l'acqua che hai in corpo. Non aspettarti frasi sdolcinate, perché come ben sai, non sono la persona giusta per queste cose; aspettati piuttosto qualcosa nel mio stile. Nonostante tu non faccia altro che infastidirmi o dire cose alquanto cringe (che non riporterò per questo motivo), grazie per essere sempre al mio fianco e per la fiducia che mi dai ogni giorno, sostenendomi in ogni momento del mio percorso. La tua presenza e il tuo affetto (anche se un po' eccessivo) mi hanno davvero migliorato. Da quando stiamo insieme, sono diventato più empatico e, sorprendentemente, anche più affettuoso... non sono più il ragazzo dal cuore di ghiaccio, ma qualcuno che, grazie a te e a tutto l'amore che mi dai (per cui non ti ringrazierò mai abbastanza), si è un po' sciolto. Tuttavia, nonostante tutto il bene che ti voglio, e che non ti dico mai, non smetterò di infastidirti e non smetterò di insultare la tua amatissima Ariana Grande o il grandissimo patrimonio dell'UNESCO, ovvero lo straordinario carnevale di Palma Campania.

Silvana, non sei solo una semplice amica, ma anche una mamma, una figlia, un jimbro, un piccolo Eolo e, soprattutto, la migliore partner che potessi desiderare per le nostre performance canore. So che in famiglia non sono il tuo figlio preferito e che il nostro rapporto è fatto di litigi e momenti di fastidio reciproco, ma nonostante tutto, non posso negare che ti voglio un bene dell'anima. Te lo dico ora, perché credo tu sia lucida, così non potrai dirmi che lo faccio solo per farti contenta. Non avrei mai immaginato che da un semplice "*hai seguito bene analisi ehhh*" potesse nascere un'amicizia così bella e sincera, basata su complicità e fiducia. Grazie per farmi compagnia ogni giorno a lezione, nonostante sia stata tu a costringermi a seguire la magistrale con te. Senza di te sarebbe una noia mortale e non saprei proprio cosa fare: passerei le ore a seguire le lezioni, cosa che proprio non fa per me. Grazie anche per continuare a sfamarmi, nonostante io consumi sempre tutta la tua dispensa (ti porterò dei pacchi di pasta). Voglio lasciarti con una frase molto filosofica che riflette a pieno il nostro rapporto: "I cavallucci marini sono belli perché stanno nell'azzurro".

Martina, se mi chiedessero come sarebbe Hello Kitty se fosse reale, risponderei semplicemente con il tuo nome. Sei entrata nel gruppo come la fidanzata di Gennaro, e da allora non hai più cambiato ruolo... anzi, ultimamente sei stata anche declassata a sorella di Gerardo. I tuoi modi di fare e la tua simpatia sono contagiosi e riescono sempre a farmi sorridere. Ma il vero spettacolo avviene quando dentro di te scorre l'essenza magica dell'alcol. In quei momenti, tutto è imprevedibile: potresti passare dal disperarti per uno chignon rovinato a temere di sederti per colpa degli 'spini', fino a ripetere la stessa domanda "mi vuoi bene?", alla quale risponderò sempre con un 'sì'. Ma in fondo, che ci sia alcol o meno, con te ho sempre vissuto dei bei momenti, e per questo ti dico: "Uh sceeee, babbasoo".

Simona, non ho mai capito se ti piacesse il rosa, visto che non te l'ho mai visto addosso. Possiamo considerarti la zia boomer, quella che ha il gruppo con le amiche dove si invia solo il buongiorno. Grazie a te, la media d'età del gruppo è decisamente più alta. Sei una persona che non bisogna far arrabbiare, a meno che uno non voglia sentirsi urlare addosso "keep calm bella". Scherzi a parte, anche se non ci vediamo spesso o interagiamo poco, ti apprezzo davvero come persona. Penso che siamo molto simili nel dire le cose come stanno. La tua schiettezza è il tuo difetto peggiore e il tuo pregio migliore, ma forse no, sappiamo tutti che il tuo vero difetto è l'inglese. Ci andiamo a prendere un Midori SHOUR?

Veronica, Maria e Gigi, volevo ringraziare anche voi, perché, nonostante ci conosciamo relativamente da poco e non ci vediamo spesso, avete comunque contribuito a rendere questi anni più belli. Spero di poter trascorrere più tempo insieme e consolidare il nostro rapporto, e chissà, magari nei prossimi ringraziamenti avrete una sezione tutta per voi (tranne per Veronica, ovviamente). *Sara*, lo so che in fondo per te siamo solo un ripiego, ma nonostante tutto, volevo comunque ringraziarti, perché senza di te alla magistrale, avrei dovuto sorbirmi Silvana da solo! Scherzi a parte, spero che in questo periodo tu ti sia sentita a tuo agio con noi e che, alla fine di questi anni, ti sentirai ancora più integrata nel gruppo.

Grazie a tutti per questi anni fantastici. Anche se non sempre lo mostro apertamente, sappiate che vi voglio un sacco di bene. Spero di continuare a vivere bei momenti insieme, anche se sappiamo tutti che, una volta finite le lauree, il gruppo si sfascerà, non è vero?

Siamo finalmente arrivati ai ringraziamenti per la mia famiglia, e come giusto che sia, le prime persone che meritano di essere ringraziate sono quelle che mi hanno permesso di arrivare fin qui e senza le quali non sarei la persona che sono oggi: i miei genitori. Grazie per tutti i sacrifici che fate ogni giorno, non potrò mai ringraziarvi abbastanza. Lo so, forse non sono il figlio perfetto: parlo poco, sono sempre distratto e raramente vi do una mano, ma nonostante tutto, continuate ad amarmi e a volermi bene. E per fortuna, non mi avete ancora cacciato di casa. Mi avete insegnato l'importanza del rispetto, del sacrificio e della perseveranza, valori che porterò sempre con me. Voi siete il mio punto di riferimento, so che ci sarete sempre per me e non mi abbandonerete mai. E io, nel mio piccolo, prometto di esserci sempre per voi. Vi voglio un mondo di bene.

Altre due persone a cui è doveroso esprimere un ringraziamento sono mio fratello *Alberto* e la sua fidanzata *Ines*, che, nonostante la distanza che ci separa, sono sempre stati vicini e si sono costantemente interessati al mio percorso e a come stesse andando. *Alberto*, da piccoli non andavamo molto d'accordo. Non riuscivamo a stare insieme senza che finissi sempre per piangere, perché litigavamo e, come sempre, avevi la meglio. Poi siamo cresciuti e tutto è cambiato. Oggi il nostro legame fraterno è solido e forte. Sono grato di avere un fratello come te e orgoglioso dei traguardi che hai raggiunto. Ti auguro di conquistarne ancora di più, sempre più importanti. Grazie per tutto quello che hai fatto per me, anche senza chiedere nulla in cambio. Prometto che un giorno mi rifarò. Sei e sarai sempre il mio fratellone.

Ines, sono davvero felice che tu faccia parte della vita di mio fratello. Non avrebbe potuto trovare una persona migliore di te. Sei una presenza importante nella nostra famiglia, e ti considero come una sorella. Grazie per tutto ciò che porti nella nostra vita.

Ci tenevo a ringraziare anche i miei zii più cari: *Aniello* e *Giuseppina*, *Felice* e *Maria*, *Antonietta* e *Aldo*, *Gerardina* e *Sebastiano*. Ognuno di voi mi ha insegnato il vero significato di una famiglia unita e di come sostenersi sempre, soprattutto nei momenti più difficili. Vi ringrazio di cuore per tutte le lezioni di vita che mi avete dato e l'affetto che mi avete sempre mostrato. Ogni insegnamento è un dono che custodirò con cura e che porterò con me per sempre, cercando di farne tesoro in ogni momento della mia vita.

Dopo gli zii, è il momento dei cugini: *Giusy, Pasquale, Arianna, Giuseppe, Simona, Alessio, Roberta, Mena, Sara, Fabrizio e Sonia*. Essendo il più piccolo della famiglia (e un tempo anche in altezza), mi avete sempre trattato come un fratellino minore. Con alcuni di voi sono cresciuto fianco a fianco, mentre con altri c'è una differenza di età che ci separa. Nonostante questo, non mi sono mai sentito fuori posto quando ero con voi e mi avete sempre accolto con affetto. Un ringraziamento speciale va a mia cugina *Sonia*, che ha sempre riposto in me una fiducia enorme. Non hai mai esitato a schierarti dalla mia parte, anche quando significava andare contro tutti. Sei quella che mi capisce meglio di chiunque altro. Grazie per aver sempre creduto in me.

Infine, desidero ringraziare una persona che, pur non conoscendomi, ha fatto moltissimo per me: il maestro *Eiichiro Oda*, autore del capolavoro chiamato *One Piece*. Proprio per questo motivo, concludo dicendo:

Ore wa Monkey D. Luffy, Kaizoku-ō ni naru otoko da.