POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

DRUG DISCOVERY

Piccinato Mattia - 10717274
Francesco Rita - 10794794

Professore:
G. Palermo

Anno accademico:
2022-2023

Sommario: La ricerca e lo sviluppo di nuovi farmaci per combattere le malattie rappresentano un processo che richiede spesso molti anni di lavoro per raggiungere il successo desiderato. In questo contesto, l'intelligenza artificiale può offrire un prezioso supporto. Le aziende farmaceutiche lavorano alla ricerca di molecole che possano essere adatte a trattare determinate patologie, le quali però risultano spesso essere tanto costose da impedire un margine di guadagno.

In questo progetto si è voluto sfruttare un approccio generativo per l'esplorazione dello spazio chimico, generando molecole simili ad una molecola di input. Dato che questi composti, tuttavia, possono risultare di difficile implementazione pratica o dai costi elevati, si è applicata in uscita anche una ricerca di similarità all'interno di un dataset in cui sono state raccolte molecole facilmente ed economicamente realizzabili.

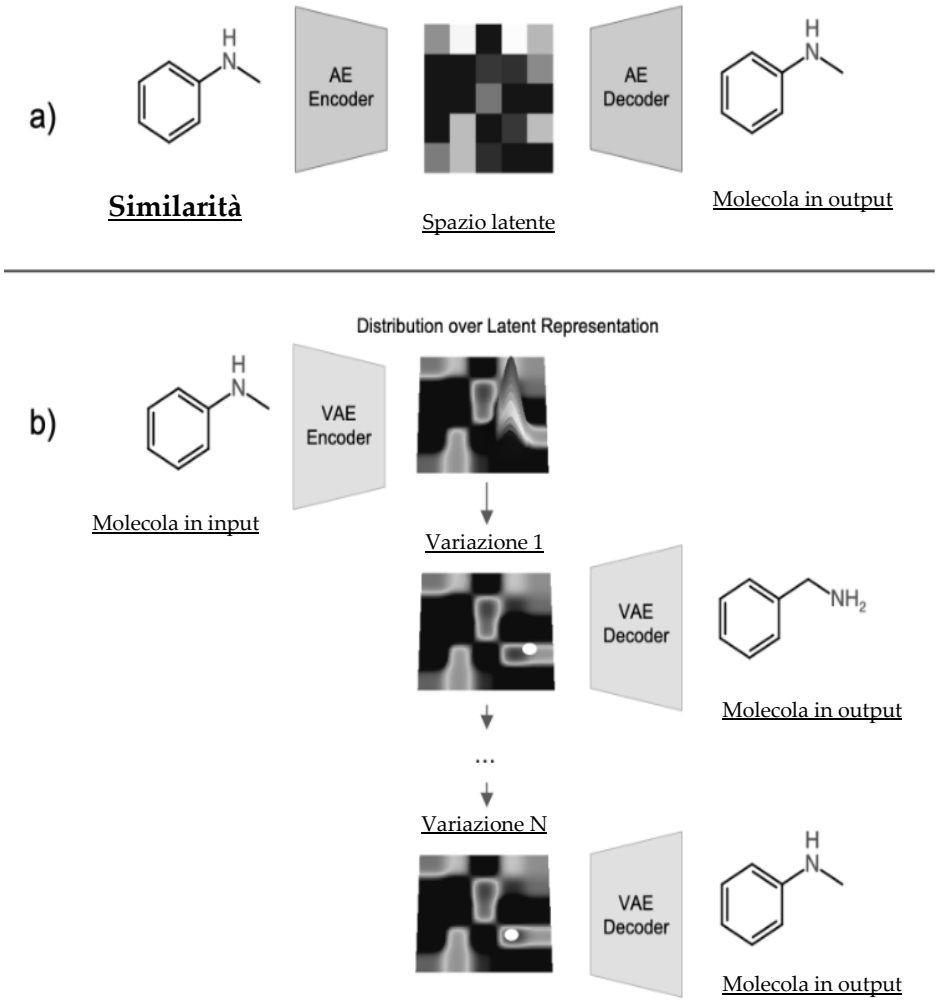
1. Specifiche

1.1 Obiettivi

Scegliere un modello generativo adatto alle specifiche del progetto.

Scrivere uno script finalizzato ad esplorare lo spazio latente del modello, e produrre in output un insieme di molecole "adeguatamente simili" ad una molecola in input.

Implementare la ricerca per similarità per cercare le molecole più simili a quelle generate in un dataset di molecole disponibili.



1.2 Requisiti implementativi

Non ci sono vincoli sul linguaggio di programmazione da utilizzare. Non ci sono vincoli sul modello da utilizzare. Non ci sono vincoli sul formato del file contenente il dataset.

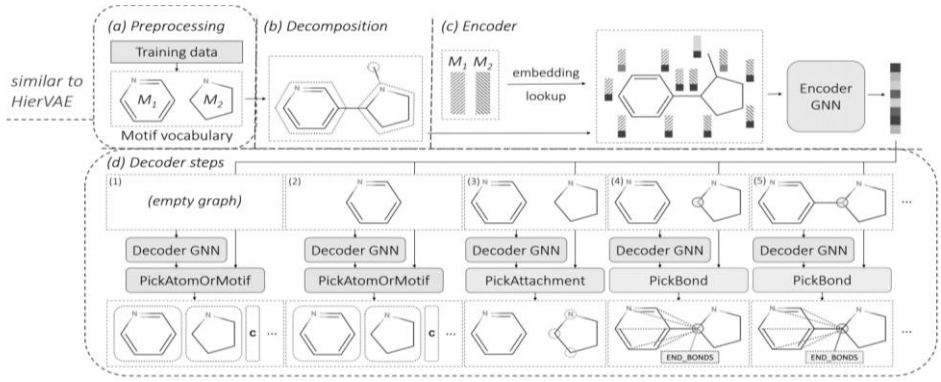
La molecola in input viene fornita nella codifica standard SMILES [1].

2. Implementazione

2.1 Scelta del modello

Allo stato dell'arte, si distinguono due categorie di modelli utilizzati nell'approccio generativo: Variational Autoencoder (VAE) e Generative Adversarial Network (GAN) [2].

Siccome i GAN manifestano un alto grado di specializzazione, rendendo più complessa l'individuazione di una rete adeguata per i propri obiettivi, abbiamo scelto di adottare un VAE [3]. In particolare, abbiamo utilizzato *molecule_generation* [4], un'implementazione in Python del modello MoLeR [5] sviluppata dal dipartimento di ricerca di Microsoft.



Schema del funzionamento di MoLeR

La decisione di utilizzare un modello VAE è giustificata dalla sua intrinseca efficacia nel preservare interi pattern strutturali dell'input anche nell'output generato. Questo effetto riveste un notevole interesse per il nostro scopo, poiché l'interazione chimica desiderata, globalmente, dipende spesso da una porzione specifica della molecola piuttosto che dall'intera struttura.

2.2 Esplorazione dello spazio latente del modello

Tramite il metodo *encode()* del modello, è possibile codificare una lista di stringhe SMILES nello spazio latente del modello. Le molecole sono mappate su un vettore da 512 numeri decimali, denominati *features*, le quali possono essere manipolate per discostarsi nello spazio 512-dimensionale di un certo delta. In questo modo, è possibile cercare molecole simili a quelle date in input.

Abbiamo utilizzato, per la realizzazione, l'istanza allenata del modello fornita contestualmente al pacchetto (vedi paragrafo 5.1).

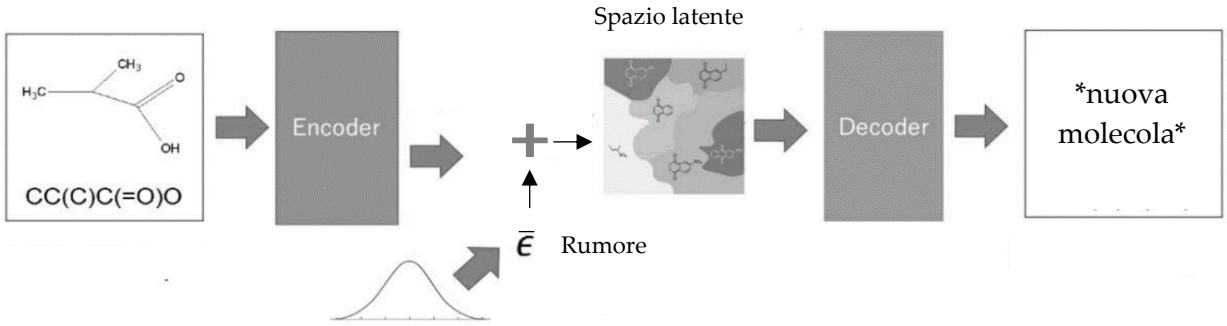
Per esplorare lo spazio latente vi sono varie strategie alternative, e per ciascuna si pone il problema di scegliere un valore da assegnare ai rispettivi parametri.

Ricerca di punti vicini in base alla distanza

Un primo approccio consisterebbe nella generazione di uno scostamento mediante una distribuzione normale per ogni feature, con media centrata nel valore della feature corrente e varianza "piccola a sufficienza".

In tal caso, poiché un piccolo spostamento in una regione a bassa densità comporterebbe una bassa probabilità di trovare molecole nelle vicinanze - e siccome è ragionevole supporre che una molecola sia fortemente caratterizzata dalle sue feature "rare" - è altrettanto ragionevole non variare il valore di una feature qualora questa si trovi in un intervallo poco denso della sua distribuzione.

Un difetto di questo approccio, tuttavia, è il fatto che, dato un valore $\sigma^2{}_i$ come varianza da utilizzare per aggiungere rumore sulla *i*-esima feature della molecola data in input, non si avrebbe controllo diretto sul numero di molecole che vengono incluse nel "raggio" della gaussiana, poiché questo dipende dalla densità dello spazio latente - rispetto all'i-esima dimensione - nell'intorno del valore assunto da tale feature.



Ricerca inesatta dei punti più vicini

Un metodo migliore per affrontare il problema consiste nel calcolare l'iper-volume minimo contenente le n molecole più vicine a quella data.

Siccome tale problema ha una complessità intrattabile, si può procedere con una soluzione inesatta: sia m_0 la molecola data in input, sia $f_{i,j}$ il valore assunto dalla i-esima feature della molecola m_j, si vuole calcolare, per ogni $i \in [1, 512]$, l'intervallo I_i tale che

- sia centrato in $f_{i,0}$
- $\exists\, m_1, \dots, m_n$ t.c. $\forall j \in [1, n]\ f_{i,j} \in I_i$

Utilizzando gli intervalli trovati come lati di un iper-volume, si trova un'approssimazione di quello esatto.

Sia $p = n/N$ con N numero di molecole nello spazio latente, e sia $r_i = |I_i|/2$, segue l'algoritmo adottato per calcolarlo.

Partendo dal valore di $f_{i,0}$, si calcolano iterativamente i raggi *R(k)* con *k = 1, …* fino a trovare *k* per cui $M(k) \geq p$, dove *M(k)* è la percentuale di molecole "coinvolte" nell'intervallo centrato in $f_{i,0}$ di ampiezza $2R_i(k)$.

Sia df la larghezza degli intervalli sui quali sono state discretizzate le distribuzioni delle feature (vedi paragrafo 3.2), *R(k)* è definita ricorsivamente ponendo:

$$R_i(1) = \min\ \{\ resto(f_{i,0}, df)\,,\ df\ -\ resto(f_{i,0}, df)\ \}$$

$$R_i(k) = \begin{cases} R_i(k-1) + df - R_i(1)\,, & se\ k\ pari \\ R_i(k-1) + R_i(1)\,, & altrimenti \end{cases}$$

In questo modo, si aggiunge al raggio *R(k-1)* quanto serve per raggiungere il bordo dell'intervallo df adiacente più vicino - ovvero, di fatto, completando l'intervallo raggiunto al passo precedente - e potendo così calcolare agevolmente la percentuale p_k raggiunta al passo iterativo *k*-esimo; infatti, ad ogni passo iterativo, vale $p_k = p_{k-1} + \frac{\Delta R(k)}{I}(f_k + f_{k-1})$ dove f_k rappresenta la frequenza dell'intervallo raggiunto al passo *k*-esimo.

Sia dunque k_f l'ultimo valore di *k*, tale per cui $p_{k_f} \geq p$ e $p_{k_f - 1} < p$. É possibile impostare una semplice equazione di primo grado in Δr:

$$p = p_{k_f - 1} + \frac{\Delta r}{df}(f_{k_f} + f_{k_f - 1})$$

La cui soluzione è l'incremento da apportare a $R(k_f - 1)$, da cui:

$$r = R(k_f - 1) + \frac{(p - p_{k_f-1}) * df}{f_{k_f} + f_{k_f-1}}$$

L'unica approssimazione utilizzata è dovuta alla discretizzazione delle distribuzioni (vedi paragrafo 3.2): si assume infatti che nel singolo intervallo df la densità sia costante (per intervalli infinitamente piccoli, la distribuzione tende ad essere continua).

Discussione dell'efficacia

L'iper-parallelepipedo risultante potrebbe essere vuoto (a meno della molecola di partenza) nel remoto caso in cui nessuna molecola dovesse occorrere nella migliore percentuale *p* per ogni sua feature (scenario frequente solo per percentuali molto basse). In generale, l'insieme di molecole che si intende raggiungere viene ridotto ad un suo sottoinsieme drasticamente minore.

Per rispondere all'eventualità di non trovare alcuna nuova molecola, si può ridimensionare *p* dinamicamente dopo averne generata una. Sia $p(t)$ la probabilità utilizzata al passo *t*-esimo (per calcolare la t-esima molecola), allora

$$p(t) = \begin{cases} p(t-1) + \Delta p_{doppione} & , \quad se\ viene\ trovato\ un\ doppione \\ \max\{\ p(t-1) + \Delta p_{nuova}, 0\ \} & , \quad altrimenti \end{cases}$$

con $\Delta p_{doppione} > 0$ e $\Delta p_{nuova} < 0$ (vedi paragrafo 3.3). La scelta della probabilità iniziale p(S0) (vedi paragrafo 3.3) non preclude quindi la capacità dell'algoritmo di generare molecole in nessun modo, poiché p varia dinamicamente, e, a partire da essa, l'iper-parallelepipedo tenderà a ridursi fino a che vi saranno molecole utili da esplorare, o ad espandersi per includerne altre. Il processo termina quando *p(t)* supera una soglia p_{max}, oltre la quale si ritiene che espandere ulteriormente comporti l'inclusione di molecole non simili (vedi paragrafo 3.3).

Inoltre, ad ogni iterazione, verranno escluse le code della distribuzione fino a $p(t) * tail$ (vedi paragrafo 3.3), sia a sinistra che a destra (la distribuzione di tutte le feature ha forma "a campana", vedi paragrafo 3.2). In questo modo, come per la strategia trattata nel precedente sottoparagrafo, le feature più rare non verranno modificate, e il numero di feature modificate cresce e decresce volta per volta.

In realtà, nell'implementazione di questo algoritmo, ad ogni iterazione generiamo due molecole, ciascuna con un diverso valore di *tail*, garantendo una più varia esplorazione dello spazio e riducendo la probabilità di escludere molecole ottime.

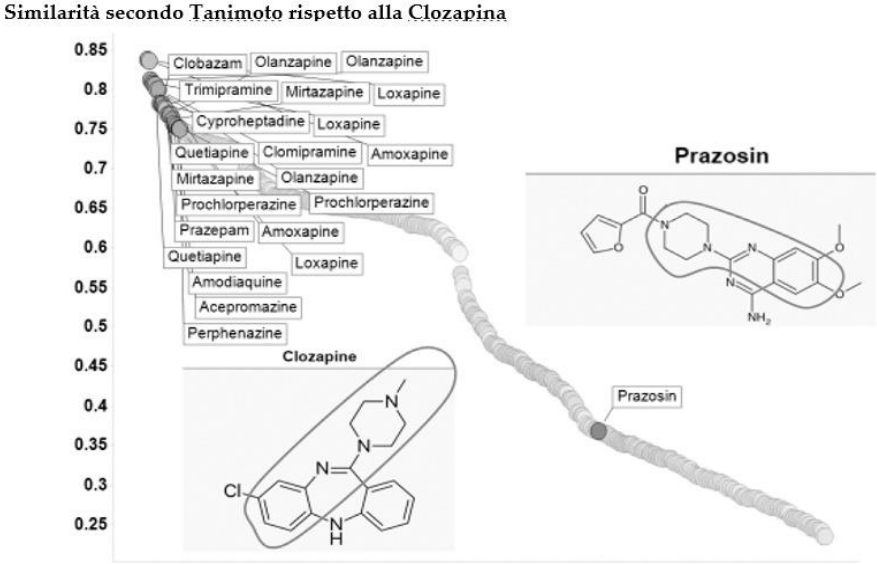
2.3 Ricerca di similarità

Per la ricerca di similarità abbiamo adottato l'utilizzo delle fingerprint topologiche di *RDKit* [6][7][8] per la rappresentazione delle molecole, poiché queste sarebbero meno sensibili a piccoli cambiamenti locali della struttura rispetto ad altri tipi di fingerprint, focalizzandosi, ad esempio, sulla connettività globale della molecola piuttosto che sull'aggiunta o rimozione locale di un legame.

In merito alle strutture coinvolte, per creare un efficiente sistema di ricerca per similarità abbiamo utilizzato il pacchetto *FPSim2* [9], il quale ci ha permesso di generare un database di fingerprint in formato *.h5* appositamente progettato per l'implementazione di una ricerca rapida ed efficiente.

La metrica che abbiamo adottato per misurare la similarità tra due molecole è il coefficiente di Tanimoto [10], la misura più comune allo stato dell'arte. Il coefficiente di Tanimoto è un numero reale compreso tra 0 e 1, pari ad 1 se le due molecole sono identiche e pari a 0 se le due molecole sono completamente diverse. È importante

notare come Tanimoto sia un indicatore strutturale abbastanza rigido: due molecole con similarità inferiore a *0.4* (ad esempio) possono condividere una parte integrante della struttura (vedi immagine).



Nota: il valore di similarità può variare in base al tipo di fingerprint utilizzato per codificare le molecole.

Dopo la generazione, per ogni molecola m_i generata, vengono selezionate all'interno del dataset tutte le molecole la cui similarità rispetto a m_i superi la soglia s, escludendo quelle già selezionate in precedenza (per evitare soluzioni duplicate).

Dalle molecole che soddisfano il suddetto criterio vengono estratte le prime x (in ordine di similarità decrescente) e tra queste viene scelta solo quella più simile alla molecola data in input (vedi paragrafo 3.6).

In questo modo, ad ogni molecola generata dal modello - siccome potrebbe essere irrealizzabile o economicamente insostenibile - associamo una sua "rappresentante", scelta tra le più simili ad essa all'interno del dataset di molecole disponibili.

Al termine dell'esecuzione, vengono mostrate in output le molecole "rappresentanti" selezionate, in ordine decrescente di similarità rispetto alla molecola data in input (con la rispettiva similarità).

2.4 Risoluzione di problematiche

Nel corso della realizzazione del progetto, ci siamo imbattuti in un errore molto frequente segnalato da *RDKit*: *"Explicit valence for atom #N, X, is greater than permitted"*. Siccome non vi era motivo di credere che gli SMILES nel dataset fossero malformati, abbiamo ipotizzato che seguissero una convenzione diversa da quella adottata da RDKit, che pertanto non era in grado di interpretarli correttamente. Di conseguenza, abbiamo a lungo analizzato gli SMILES patologici nel dataset, fino a notare dei pattern ricorrenti: sembrava che, ovunque vi fosse una carica formale totale su un atomo, questa sarebbe stata sottintesa tra parentesi quadre, invece che essere indicata esplicitamente. Questa nostra ipotesi ha trovato conferma nella pagina inglese di Wikipedia

sulle SMILES [1], ma abbiamo comunque deciso di raccogliere dei dati a supporto prima di sostituire ogni SMILES patologica (vedi paragrafo 3.1).

Un altro problema a cui abbiamo dovuto far fronte durante la realizzazione del progetto consiste invece in un bug dovuto ad un conflitto di versioni tra *RDKit* e *molecule_generation*. In particolare, la funzione *GetSSSR* di *RDKit* ha cambiato il tipo di ritorno nelle ultime versioni, comportando alcuni errori di compatibilità ed un conseguente crollo della qualità dell'output del modello. Abbiamo risolto il problema debuggando il codice di *molecule_generation* ed abbiamo segnalato il problema nelle Issues della rispettiva repository su GitHub, ottenendo appunto un consistente miglioramento della qualità dell'output (vedi paragrafo 3.4).

3. Raccolta dati

3.1 Correzione degli SMILES patologici

Si definiscano *σ-Incriminati* quegli SMILES che contengono la sottostringa σ.

Numero di SMILES nel dataset: 4.964.421

Numero di SMILES [N]-Incriminati: *14834*; Percentuale di [N]-Incriminati sul totale: *0.30%*; Tasso d'errore [N]-Incriminati : *100%*

Tasso d'errore di [N]-Incriminati dopo aver sostituito [N] con [N+]: *0%*

Numero di SMILES [B]-Incriminati : *1176*; Percentuale di [B]-Incriminati sul totale: *0.02%*; Tasso d'errore [B]-Incriminati : *99.91%*

Tasso d'errore di ex-[B]-Incriminati dopo aver sostituito [B] con [B-]: *0%*

Numero di SMILES [O]-Incriminati : *581*; Percentuale di [O]-Incriminati sul totale: *0.01%*; Tasso d'errore [O]-Incriminati : *71.60%*

Tasso d'errore di [O]-Incriminati dopo aver sostituito [O] con [O+]: *0%*

Siccome sostituendo ogni [N] con [N+], ogni [B] con [B-] ed ogni [O] con [O+] la totalità di errori legati ad essi è stata azzerata, possiamo inferire che l'ipotesi fosse corretta.

Dopo questa operazione automatica, il *99.9999%* delle molecole era pronto per essere processato, e gli SMILES patologici rimanenti sono stati considerati delle singolarità. In quanto tali, abbiamo deciso di risolvere ciascuna manualmente.

Gli script utilizzati per individuare i pattern, testare gli effetti di un'eventuale sostituzione e per effettuare una sostituzione sono presenti nella cartella (vedi paragrafo 5.1).

3.2 Analisi della distribuzione delle 512 feature

Tramite il modulo *matplotlib* [11] abbiamo calcolato la distribuzione di ciascuna feature su un campione di 10k molecole. Attraverso questo esperimento, siamo stati in grado di osservare che la totalità delle feature presenta una distribuzione dalla forma a "campana", più o meno larga, con un picco intorno allo 0. Con lo stesso script abbiamo inoltre memorizzato la frequenza registrata per ogni intervallo su un file di testo, per poter accedere dinamicamente alla frequenza di ogni valore per ogni feature.

Dopo alcuni test preliminari abbiamo potuto osservare che la distribuzione più larga si estende fino a qualche unità prima e dopo lo zero; per questa ragione abbiamo deciso di utilizzare intervalli lunghi *0.2*, in modo da avere una media di qualche centinaio di molecole per ogni intervallo (rapporto tra la numerosità del campione e il numero di intervalli nel caso pessimo).

Lo script utilizzato per discretizzare le distribuzioni delle feature e visualizzarne i grafici è presente nella cartella (vedi paragrafo 5.1).

3.3 Scelta dei parametri per l'esplorazione dello spazio latente

Per ogni intervallo I_i, se questo si trova nella coda sinistra della distribuzione, la metà sinistra di esso corrisponde sempre ad una probabilità pari al massimo a *p*0.5*, poiché la coda sinistra è crescente. Per questo motivo, escludendo la coda sinistra fino a *p*0.5* si ha la certezza che I_i non possa raggiungere il valore minimo della distribuzione, sprecando di fatto parte della sua ampiezza. Simmetricamente, questa proprietà vale anche per la coda destra. Per quanto detto finora, conviene quindi fissare la costante *tail* ad un valore di almeno *0.5*; empiricamente, abbiamo riscontrato che adottare $tail = 0.5$ e $tail = \frac{2}{3}$ come valori per l'iterazione garantisce un maggior numero di risultati di buona qualità rispetto ad utilizzare soltanto l'uno o l'altro, senza rallentare eccessivamente le prestazioni.

La percentuale di partenza *p(0)* viene fissata allo 0%, in modo da forzare come prima molecola la molecola di input, permettendo poi una lenta crescita di *p* a passi di $\Delta p_{doppione}$ finché necessario. Il $\Delta p_{doppione}$ è fissato a *0.5%* perché abbiamo riscontrato che per step di dimensione maggiore il numero di molecole notevolmente simili diminuisce, mentre per step di dimensione inferiore si ha come unico effetto un peggioramento delle prestazioni. Il Δp_{nuova} è stato fissato a -5% per ragioni analoghe: abbassarlo ulteriormente avrebbe solo peggiorato le prestazioni, ed alzarlo avrebbe inficiato la qualità del risultato.

La probabilità p_{max} è stata fissata al 50% poiché abbiamo verificato sperimentalmente che qualunque molecola calcolata con una probabilità $p > p_{max}$ risulta essere eccessivamente "diversa" (non simile) dalla molecola data in input, e per $p_{max} < 50\%$ l'algoritmo converge troppo velocemente.

Lo script usato per misurare questi risultati è il file principale nella sua versione primaria (vedi paragrafo 5.1).

3.4 Similarità tra input e output del modello

Abbiamo calcolato la similarità tra le molecole generate in output dal modello e le rispettive molecole date in input - ovviamente senza l'aggiunta di rumore nello spazio latente -, su un campione indipendente di 5k molecole, prima e dopo la correzione dell'errore discusso nel capitolo precedente (vedi paragrafo 2.4): nella prima occasione, la similarità media raggiunta dal modello si è assestata intorno a *0.2*; invece, dopo aver corretto l'errore, la stessa ha raggiunto un valore di *0.42*.

Lo script utilizzato per calcolare la similarità media è presente nella cartella (vedi paragrafo 5.1).

3.5 Percentuale di molecole riprodotte esattamente dal modello

Abbiamo calcolato, sullo stesso campione di 5k molecole menzionato nel precedente paragrafo, la percentuale di molecole che sono state riprodotte correttamente dal modello.

- Con similarità *1* tra input e output: *1.4%*
- Con similarità di almeno *0.7* tra input e output: *6.22%*
- Con similarità di almeno *0.5* tra input e output: *22.94%*

Il foglio Excel utilizzato è stato ricavato dall'output dello script menzionato nel paragrafo precedente, ed è presente nella cartella (vedi paragrafo 5.1).

3.6 Scelta dei parametri per la ricerca per similarità

Per selezionare le migliori x molecole, abbiamo scelto una soglia minima s = 0.5, per due ragioni:

- 0.5 indicativamente suggerisce che più della metà della molecola è identica a quella di riferimento; di conseguenza, nei casi più fortuiti, più del 50% potrebbe risultare simile anche alla molecola data in input.
- Abbiamo riscontrato sperimentalmente che con questa soglia sembrerebbe quasi sempre esistere una molecola sufficientemente simile nel dataset (per percentuali più alte questa garanzia viene meno), e l'intenzione è quella di mostrare almeno un risultato per ogni molecola generata.

In sintesi, nel caso peggiore questa scelta comporta l'aggiunta di risultati di scarsa qualità all'output finale, ma in generale accresce la probabilità di trovare una molecola particolarmente simile a quella data in input, che altrimenti non sarebbe stata individuata.

Il numero x di molecole prese in considerazione serve ad aumentare notevolmente le performance: infatti, valutare tutte le molecole sopra la soglia s sarebbe computazionalmente troppo pesante, e questo limite permette un calcolo "avido" quasi istantaneo.

4. Conclusioni

4.1 Compatibilità

Le SMILES utilizzate seguono la convenzione di *RDKit*: si consiglia di ottenerli tramite le funzioni di *RDKit*. La ricerca di similarità è effettuata su un file in formato *.h5* calcolato con FPSim2 [9].

4.2 Risultati

Abbiamo misurato la similarità tra molecola di riferimento e molecola generata su un campione numeroso di molecole "riprodotte esattamente" dal modello (affinché sia valida l'ipotesi di disporre di un modello perfetto).

Abbiamo verificato che, preso in considerazione il caso pessimo, sono sempre state individuate almeno 20 molecole, e la similarità massima registrata per le molecole in output è sempre stata di almeno *0.75*.

In media, per ogni esecuzione del codice, vengono registrate in output circa diciassette molecole con similarità superiore a *0.5*, otto con similarità superiore a *0.7* ed una con similarità superiore a *0.9*; complessivamente, il valore atteso per il numero di molecole in output è superiore a 50.

Nei casi più fortuiti, il nostro codice riesce a calcolare diverse molecole con similarità superiore a *0.9*.

4.3 Future ottimizzazioni e migliorie

Il modello utilizzato (MoLeR [5]) ha manifestato una qualità dell'output relativamente bassa in media (vedi paragrafo 3.4) per le molecole del dataset utilizzato, da cui dipende la qualità presentata nello scorso paragrafo.

Tuttavia, se si intende utilizzare il codice oggetto della relazione, consigliamo di verificare la qualità dell'output del modello tramite l'apposito script nella cartella (vedi paragrafo 5.1) per un campione di molecole estratto dal proprio dataset. Infatti, abbiamo riscontrato la presenza di un sottoinsieme delle

molecole nel dataset, le quali vengono esattamente replicate in output (vedi paragrafo 3.5) - ovvero con similarità pari ad 1 tra input e output.

Nel caso in cui la partizione di interesse dello spazio molecolare dovesse riguardare molecole contenute - o "simili" a quelle contenute - in tale sottoinsieme, si ha ragione di credere che il modello possa fornire buone prestazioni.

Altrimenti, si consiglia di provare ad allenare il modello su un insieme di molecole più adatto, o, alternativamente, di sostituire il modello utilizzato.

5. Documentazione

5.1 Contenuto della cartella del progetto

La cartella principale *drug-discover* contiene il file *fingerprints_generator.py*, che genera il dataset in formato *.h5* data una lista di file contenenti le stringhe SMILES che costituiscono il dataset, e il file *generator.py*, che implementa il progetto; inoltre, contiene le cartelle *model* (contenente l'istanza allenata del modello), *docs* (contenente gli articoli [2], [3] e [5]), *dataset* e *data analysis*.

La cartella *dataset* contiene i file che costituiscono il dataset ed il file *cleaner.c*. La cartella *data analysis* contiene a sua volta quattro cartelle: *exploration* contenente la versione di generator.py utilizzata per raccogliere dati sulla qualità del progetto; *features_distribution_plots* contenente il grafico di ciascuna feature, il file di testo in cui sono archiviate le distribuzioni, e il codice utilizzato per realizzare il tutto; *pathologic_atoms* contenente i tre script utilizzati, rispettivamente, per individuare le SMILES patologiche, testarle un'eventuale sostituzione ed effettuare la sostituzione; e, infine, *similarity*, contenente lo script utilizzato per valutare la qualità dell'output del modello.

5.2 Bibliografia e sitografia

[1] https://it.wikipedia.org/wiki/SMILES

[2] A brief introduction to generative models, Alex Lamb, 2021

[3] Tutorial on Variational Autoencoders, Carl Doersch, 2016

[4] https://github.com/microsoft/molecule-generation

[5] Learning to extend molecular scaffolds with structural motifs, Krzysztof Maziarz, Henry Jackson-Flux, Pashmina Cameron, Finton Sirockin, Nadine Schneider, Nikolaus Stief, Marwin Segler, Marc Brockschmidt, 2022

[6] https://github.com/rdkit/rdkit

[7] https://www.rdkit.org/docs/GettingStartedInPython.html#fingerprinting-and-molecular-similarity

[8] https://rdkit.org/docs/source/rdkit.Chem.rdmolops.html#rdkit.Chem.rdmolops.RDKFingerprint

[9] https://chembl.github.io/FPSim2/

[10] https://it.wikipedia.org/wiki/Indice_di_Jaccard#Coefficiente_di_Tanimoto_(coefficiente_esteso_di_Jaccard)

[11] https://github.com/matplotlib/matplotlib