

UNIVERSITÀ DI PISA



Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea Magistrale in Informatica

Relazione Progetto di Tesi per La
Laurea Magistrale in Informatica

Un Sistema per la Valutazione
di Proprietà Molecolari Basato
SU
Reti Neurali per Grafi

Relatore:

Alessio Micheli

Candidato:

Luca Baronti

Anno Accademico 2013/14

Sommario

Lo studio di modelli per la predizione di proprietà fisico/chimiche e delle interazioni biologiche di sostanze chimiche, è di particolare importanza per la cheminformatica, in particolare nell'ambito della tossicologia computazionale. L'obiettivo di questo lavoro è stato quello di realizzare un sistema interattivo in ambito cheminformatico che fornisca all'esperto la predizione su una determinata proprietà fisico/chimica o biologica di un composto, rendendo facilmente fruibili le informazioni sulla predizione.

Uno dei componenti principali del sistema proposto è costituito da una libreria per la creazione e gestione di istanze di una rete neurale per l'analisi contestuale di grafi (*NN4G*). La libreria è stata progettata seguendo criteri di modularità e di facilità d'espansione, ed è alla base di strumenti sviluppati per l'addestramento e l'analisi di istanze *NN4G*.

Le istanze prodotte sono state utilizzate nella realizzazione di un sistema di valutazione delle predizioni, che sfrutta una riformulazione della funzione di uscita del modello *NN4G* per identificare, in modo innovativo, i contributi strutturali nei risultati di predizione. Prove sperimentali del sistema hanno inoltre permesso di osservare, su esempi in ambito tossicologico, una coerenza tra sotto-strutture rilevate con contributi positivi ed alcune strutture molecolari note in letteratura per essere caratterizzanti per il problema analizzato.

Ulteriori evoluzioni del sistema hanno permesso una riduzione dei contributi risultati come meno rilevanti, permettendo una focalizzazione sulle sotto-strutture principali, migliorando la possibilità di riconoscere strutture critiche da parte dell'esperto del dominio.

Indice

1	Introduzione	1
2	Apprendimento Automatico e Approcci Contestuali	4
2.1	Trattamento dei Dati	4
2.1.1	Separazione del Dataset	7
2.2	Sintesi su Reti Neurali e Algoritmi di Apprendimento	10
2.2.1	Metodi Basati sul Gradiente	13
2.2.2	Metodi Diretti	17
2.3	Metodi Costruttivi	18
2.3.1	Cascade Correlation	19
2.3.2	Cascade Residual	20
2.4	Domini Strutturati: Notazione e Metodologie	22
2.4.1	Metodi Avanzati per il Trattamento di Dati Strutturati	24
2.4.2	Neural Network for Graphs (NN4G)	33
3	Trattamento Dati Tossicologici	37
3.1	Tossicologia Computazionale e Modelli QSAR	38
3.1.1	Descrittori Molecolari	39
3.1.2	Structural Alert	41
3.2	Strumenti Disponibili	42
3.2.1	EPI Suite	42
3.2.2	T.E.S.T.	43
3.2.3	Vega	47

3.3	Librerie Usate	50
3.3.1	OpenBabel	50
3.3.2	Armadillo	51
3.3.3	QT	51
4	Strumenti Sviluppato per la Creazione di Modelli	53
4.1	Libreria <i>felix</i>	54
4.1.1	Utilizzo della Libreria	57
4.1.2	Gestione del Dataset	62
4.1.3	Gestione dei Modelli	64
4.1.4	Gestione degli Algoritmi di Apprendimento	68
4.1.5	Gestione dei Risultati	72
4.1.6	Descrittore	74
4.1.7	Estendibilità	78
4.2	Libreria <i>qfelix</i>	81
4.3	Strumenti di Addestramento e Analisi dei Modelli	82
4.3.1	cougar	82
4.3.2	felix-reader	85
5	Strumenti Sviluppato per la Valutazione di Molecole	90
5.1	File di Uso del Predittore	91
5.2	Utilizzo del Predittore	93
5.2.1	Formula	97
5.2.2	Applicability Domain	98
5.2.3	Similar Mols	100
5.3	Contributi Strutturali	101
5.4	Struttura del Predittore	106
5.5	Rimozione Contributi Minori	109

6	Risultati Sperimentali	112
6.1	Risultati di Collaudo del Sistema di Addestramento	112
6.1.1	Alcani	113
6.1.2	Predictive Toxicology Challenge	114
6.1.3	Bursi	116
6.2	Risultati del Predittore	121
6.2.1	Variante con Rimozione dei Contributi Minori	125
7	Conclusioni	133
	Appendice A Formato SDF	137
	Appendice B Formato Gph Esteso (Gph+)	139
B.1	Definizione del Documento	140
B.2	Definizione dell'Intestazione	140
B.3	Grammatica del Grafo	141
B.4	Grammatica del Vertice	141
B.5	Esempi	141
B.5.1	Esempio di Intestazione	142
B.5.2	Esempio di Grafo	142
	Appendice C Esempi di Files	143
C.1	Esempio di File del Modello	143
C.2	Esempio Completo di Training	145
C.3	Esempio di File del Descrittore	147
C.3.1	Esempio di File Grid	148
	Appendice D Lista Completa dei Risultati	151
D.1	Alcani	151
D.1.1	Cascade Correlation	151
D.1.2	Cascade Residual	153

D.2	PTC.FM	155
D.2.1	Cascade Correlation	156
D.2.2	Cascade Residual	158
D.3	PTC.FR	159
D.3.1	Cascade Correlation	159
D.3.2	Cascade Residual	162
D.4	PTC.MM	164
D.4.1	Cascade Correlation	164
D.4.2	Cascade Residual	167
D.5	PTC.MR	169
D.5.1	Cascade Correlation	169
D.5.2	Cascade Residual	172
Appendice E Report Generato dal Predittore		175
E.1	Predictor's Report of $C_8H_{10}N_4O_2$	175
E.2	Infos	175
E.3	Applicability Domain	176
E.4	Similar Molecules	177
Appendice F Report Generato da <i>felix-reader</i>		178
F.1	Results Info	178
F.2	Training Results	179
F.2.1	Fold-1 results	179
F.2.2	Fold-2 results	180
F.3	Parameters	181
F.3.1	Model Parameters	181
F.3.2	Learning Parameters	181
Bibliografia		183

Capitolo 1

Introduzione

L'importanza delle proprietà fisico/chimiche e biologiche delle sostanze chimiche è strettamente legata alla loro presenza nella vita quotidiana. Si stima che più di 100000 prodotti chimici vengono usati quotidianamente e ogni anno un gran numero di nuove sostanze chimiche vengano introdotte, generalmente a scopi industriali e farmacologici. Nonostante il gran numero di sostanze chimiche prodotte e usate, solo una parte di queste sono soggette a controlli specifici, come quelli tossicologici, a causa dell'alto costo necessario a svolgere i test sperimentali.

Per questo motivo, recentemente l'Unione Europea ha promosso una serie di normative che regolamentino la produzione e il commercio di sostanze chimiche, con particolare attenzione al loro impatto ambientale e le loro proprietà tossicologiche. Normative come il REACH (**R**egistration, **E**valuation, **A**uthorisation and **R**estriction of **C**hemical) pongono l'accento sullo studio e l'applicazione di modelli *in silico*, come ad esempio modelli predittivi digitali, per la valutazione di proprietà chimiche, rispetto ai modelli *in vivo*, come quelli derivati dalla sperimentazione animale. Questa normativa segue i principi definiti dall'*OECD* (**O**rganisation for **E**conomic **C**o-operation and **D**evelopment) per la conformità dei modelli *in silico*, tra cui la generalità del metodo, una definizione chiara dell'*endpoint*, ovvero della caratteristica da predire, una documentazione riguardante il dominio applicativo entro cui la predizione debba essere considerata valida e un'interpretazione meccanicistica del modello.

In risposta a questa direttiva, sono stati sviluppati diversi modelli e tool predittivi basati su approcci noti in letteratura, i quali definiscono una relazione matematica tra una molecola e le sue proprietà biologiche come, ad esempio, la sua tossicità. Gli approcci QSAR (**Q**uantitative **S**tructure **A**ctivity **R**elationship), ad esempio, sono utilizzati per inferire determinate proprietà di una molecola, come una sua attività

biologica, partendo dalla sua struttura e da sue determinate proprietà fisico-chimiche. In questi approcci, tradizionalmente, una molecola viene codificata mediante l'uso di *descrittori molecolari*, ovvero proprietà quantitative caratterizzanti della molecola da analizzare, come ad esempio una serie di proprietà fisico/chimiche della molecola o alcune caratteristiche topologiche, tra cui la presenza di certe sotto-strutture note chiamate *frammenti*.

Un metodo per realizzare modelli QSAR su un endpoint è quello di usare tecniche di *machine learning* [1], applicate a dati *flat* costituiti da un insieme di descrittori selezionati specificatamente per quel endpoint.

Nel tempo sono stati sviluppati tool che utilizzano modelli QSAR per fornire un utile supporto all'esperto del dominio che analizza determinate sostanze. Per venire incontro a quanto richiesto dalla normativa REACH, alcuni di questi tool forniscono, oltre alla previsione per un certo endpoint, una serie di informazioni riguardo l'uniformità delle caratteristiche della sostanza da analizzare, rispetto al dominio applicativo per cui il modello usato dal tool è stato realizzato. Sebbene questi metodi siano molto utilizzati, ne sono stati sviluppati altri per aumentarne la generalità. Ad esempio, per i modelli creati seguendo le metodologie QSAR, non è possibile stabilire un insieme di descrittori che vada bene per ogni endpoint, e la loro selezione spesso richiede il supporto di un esperto del dominio che dovrà collaborare nella creazione del modello. Inoltre il forte affidamento ai descrittori, da parte di questi modelli, lascia aperta la possibilità di tralasciare importanti informazioni riguardanti le relazioni strutturali della molecola.

Un modo per realizzare questi obiettivi è quello di implementare un predittore che, invece di usare i descrittori, basi la sua predizione direttamente sull'informazione strutturale della molecola.

Negli anni sono state sviluppate diverse metodologie nell'ambito del machine learning per gestire domini strutturati in modo adattivo, come sequenze o grafi. Diversi approcci di machine learning hanno portato al trattamento di domini strutturati sempre più complessi, partendo dai dati *flat*, arrivando a gestire sequenze e grafi. Nel caso delle molecole, un modo per preservare le loro informazioni strutturali è quello di rappresentarle esplicitamente in forma di grafo, dove i vertici corrispondono agli atomi e gli archi corrispondono ai legami.

Il lavoro di questa tesi ha come obiettivo lo sviluppo di un predittore tossicologico che basi la sua predizione sull'informazione strutturale della molecola e che vada incontro ai punti sollevati dalla normativa REACH ponendosi nell'ambito dei predittori

attualmente disponibili. Questo tool effettuerà la predizione di un endpoint, rispetto ad una molecola, fornendo all'esperto del dominio un adeguato livello di informazioni sul dominio applicativo della predizione. Questo consentirà, all'esperto, di valutare l'affidabilità della predizione, rispetto alla molecola analizzata. Come elemento innovativo del predittore, si vuole fornire all'esperto un'adeguata rappresentazione grafica delle sotto-strutture molecolari che hanno contribuito maggiormente a formulare una predizione positiva o negativa. Per questi scopi, il predittore farà uso di un'istanza addestrata del modello *NN4G* (Neural Network for Graph) [2], modello adattivo per l'analisi contestuale di grafi. L'istanza dovrà essere selezionata, in base a criteri di performance, su un dataset con un endpoint definito. A questo proposito, assieme al predittore, è richiesta la progettazione e sviluppo di un sistema di training per il modello *NN4G*, che permetta di selezionare diversi algoritmi di apprendimento.

Oltre a questo capitolo introduttivo, questo elaborato è suddiviso in altri 5 capitoli principali.

Il capitolo 2 farà un rapido excursus sul trattamento dei dati e l'apprendimento di modelli di machine learning, con particolare attenzione ai dati strutturati e i modelli usati per trattarli. La presentazione dei modelli per il trattamento di dati strutturati seguirà un ordine cronologico, dell'evoluzione dei modelli e, allo stesso tempo, di complessità di dati strutturati trattati, per arrivare al trattamento dei grafi, di nostro particolare interesse. Nel descrivere alcuni dei modelli usati per il trattamento dei grafi, daremo ampio spazio al modello *NN4G*, centrale per il progetto.

Nel capitolo 3 faremo un accenno alle motivazioni dietro lo studio di modelli per la predizione tossicologica, passando ad una rassegna di alcuni tra i principali tool di predizione disponibili.

Il capitolo 4 raccoglierà le principali librerie e i principali tool sviluppati per l'apprendimento e l'analisi di modelli, mentre il capitolo 5 descriverà il predittore sviluppato, con le varianti apportate al modello *NN4G*.

Nel capitolo 6 verranno illustrati i risultati di collaudo del modello *NN4G* ottenuti su diversi dataset e l'analisi di alcuni casi di studio ottenuti mediante il predittore sviluppato.

Il capitolo 7 illustra gli obiettivi raggiunti e i possibili sviluppi futuri.

Capitolo 2

Apprendimento Automatico e Approcci Contestuali

In questo capitolo descriveremo alcuni concetti di base dell'apprendimento automatico, o *machine learning*, partendo dal trattamento dei dati *flat* e l'uso dei modelli, per arrivare ad illustrare alcune delle principali tecniche per il trattamento dei dati strutturati, come grafi.

Nella Sez. 2.1 e 2.2, parleremo brevemente del trattamento dei dati e dell'uso dei modelli nel machine learning facendo riferimento, per semplicità di notazione, ai dati flat. Nella Sez. 2.4 introdurremo i dati strutturati e alcuni dei modelli studiati per il loro trattamento.

2.1 Trattamento dei Dati

Nell'ambito del machine learning [1], viene chiamato *dataset* \mathcal{D} un insieme di N_p coppie (*pattern, target*)

$$(p, \mathbf{t}) \in \mathcal{D}$$

I pattern possono rappresentare dati *flat*, come vettori, o dati *strutturati*, come alberi o grafi. Per semplicità di notazione, in questa sezione descriveremo brevemente alcune tecniche di trattamento dei dati, facendo riferimento esplicito ai dati flat, mentre vedremo come queste tecniche vengono utilizzate sui dati strutturati, aspetto centrale di questo lavoro, nelle successive parti della tesi.

Nel caso dei dati flat, ogni pattern p è formato da un vettore *feature* di valori, o *label*, reali a cui ci riferiremo con $\mathbf{l}(p)$, i cui elementi verranno riferiti nel seguente modo:

$$l_i(p) \in \mathbb{R} \quad \forall i = \{0, \dots, N_l - 1\}$$

Alcune feature possono derivare, ad esempio, da proprietà numeriche della classe di dati a cui appartengono i pattern, mentre tipi di proprietà non numeriche sono codificate in diversi modi. Ad esempio, una tecnica per codificare un insieme finito e ordinato di etichette e_0, \dots, e_{k-1} è chiamata *One of K* (1oK) e consiste nel codificare un'etichetta e_i in un vettore nullo di feature, di dimensione k , imponendo ad 1 il valore in posizione i del vettore.

Gli elementi del vettore \mathbf{t} sono chiamati *valori target* t_j con $j = 0, \dots, N_t - 1$. I valori target sono ottenuti a partire dai valori delle feature, mediante una *funzione target* non nota:

$$\mathcal{F}(\mathbf{l}(p)) = \mathbf{t} \quad \forall (p, \mathbf{t}) \in \mathcal{D}$$

Nel nostro caso, si assume che il dataset ha la caratteristica di essere omogeneo, quindi ogni pattern avrà lo stesso numero di feature e valori target. Il numero e tipo dei valori target definiscono il *task* del dataset. Tra i task che verranno considerati in questo lavoro, abbiamo:

classificazione binaria se abbiamo

$$N_t = 1, \quad t_0 \in \{-1, 1\} \quad \forall (p, \mathbf{t}) \in \mathcal{D}$$

multi-classificazione se abbiamo

$$N_t > 1, \quad t_j \in \{-1, 1\} \quad \forall (p, \mathbf{t}) \in \mathcal{D}, \quad \forall j = 0, \dots, N_t - 1$$

regressione se abbiamo

$$N_t = 1, \quad t_0 \in \mathbb{R} \quad \forall (p, \mathbf{t}) \in \mathcal{D}$$

Altri tipi di task, come ad esempio la multi-regressione, non vengono descritti in questo contesto in quanto non presenti nei dataset considerati.

Nell'ambito del machine learning, il dataset viene usato per creare un modello empirico \mathcal{M} , seguendo un processo chiamato *addestramento* (o *training*), il cui scopo è quello di adattare i parametri liberi del modello. Un modello addestrato, chiamato anche *istanza*, è usato per fornire predizioni su un pattern p non presente nel dataset

usato per il training, usando le sue feature $\mathbf{l}(p)$. Un'istanza di un modello ha una performance tanto più elevate quanto più è in grado di approssimare la funzione target :

$$\mathcal{M}(\mathbf{l}(p)) \approx \mathcal{F}(\mathbf{l}(p))$$

Il funzionamento dei modelli e dell'algoritmo di apprendimento varia a seconda di diversi parametri interni come, ad esempio, quelli che definiscono la struttura della rete, nel caso di una rete neurale, o il *learning rate* nel caso di un algoritmo di discesa del gradiente (vedere Sez. 2.2). L'insieme dei parametri del modello e dell'algoritmo di apprendimento prendono il nome di *iperparametri* e la selezione degli iperparametri che portano al modello con performance migliori viene chiamata *model selection*.

Esistono diversi criteri che permettono di stabilire la qualità di un predittore, e dipendono dal task specifico. Dato un modello \mathcal{M} , con $\mathcal{M}(p)$ vettore di uscite in risposta al pattern p di ingresso, e un dataset \mathcal{D} , nel caso di regressione, uno dei modi per stabilire la performance di \mathcal{M} sul dataset \mathcal{D} è mediante l'indice *Mean Absolute Error* (*MaE*)

$$MaE(\mathcal{M}, \mathcal{D}) = \frac{1}{N_p \cdot N_t} \sum_{(p, \mathbf{t}) \in \mathcal{D}} \sum_{o=0}^{N_t-1} |\mathcal{M}(\mathbf{l}(p))_o - t_o| \quad (2.1)$$

o il *Max Absolute Error* (*MaxAbsE*)

$$MaxAbsE(\mathcal{M}, \mathcal{D}) = \max_{(p, \mathbf{t}) \in \mathcal{D}} \left(\sum_{o=0}^{N_t-1} |\mathcal{M}(\mathbf{l}(p))_o - t_o| \right) \quad (2.2)$$

Altri indici utili nel caso di regressione ad output singolo sono rappresentati dalle seguenti funzioni:

$$S(\mathcal{M}, \mathcal{D}) = \sqrt{\frac{1}{N_p} \sum_{(p, \mathbf{t}) \in \mathcal{D}} (\mathcal{M}(\mathbf{l}(p)) - t)^2}$$

$$R(\mathcal{M}, \mathcal{D}) = \sqrt{1 - \frac{1}{N_p} \sum_{(p, \mathbf{t}) \in \mathcal{D}} \frac{(\mathcal{M}(\mathbf{l}(p)) - t)^2}{(t - \mu_t)^2}}$$

dove μ_{t_o} è la media di t_o su tutti i pattern p .

Nel caso di multi-classificazione possiamo definire l'*accuracy* $Acc(\mathcal{M}, \mathcal{D})$ come segue:

$$Acc(\mathcal{M}, \mathcal{D}) = \frac{1}{N_p} \sum_{(p, \mathbf{t}) \in \mathcal{D}} I(\mathcal{M}(\mathbf{l}(p)), \mathbf{t}) \quad \in [0, 1] \quad (2.3)$$

dove

$$I(\mathcal{M}(\mathbf{l}(p)), \mathbf{t}) = \begin{cases} 1 & \mathcal{M}(\mathbf{l}(p))_o = t_o \quad \forall o = 0, \dots, N_t - 1 \\ 0 & \text{altrimenti} \end{cases}$$

Nel caso di classificazione binaria è inoltre possibile ottenere una *matrice di confusione* che contiene il numero dei falsi positivi (FP), falsi negativi (FN), veri positivi (TP) e veri negativi (TN) del modello in risposta ai pattern del dataset. Usando questi valori è possibile ridefinire la formula dell'accuracy e definire altri utili indici relativi al dataset testato:

Accuracy semplificata rispetto al caso di multiclassificazione, indica il numero di risposte corrette del modello:

$$Acc(\mathcal{M}, \mathcal{D}) = \frac{TN + TP}{N_p} \in [0, 1]$$

Precision indica il numero relativo di veri positivi ottenuti dal modello:

$$Prec(\mathcal{M}, \mathcal{D}) = \frac{TP}{TP + FP} \in [0, 1]$$

Recall chiamata anche *sensitivity*, indica il numero di veri positivi ottenuti dal modello, in rapporto al numero totale di positivi nel dataset:

$$Rec(\mathcal{M}, \mathcal{D}) = \frac{TP}{TP + FN} \in [0, 1]$$

Specificity indica il numero di veri negativi ottenuti dal modello, in rapporto al numero totale di positivi nel dataset:

$$Spec(\mathcal{M}, \mathcal{D}) = \frac{TN}{TN + FP} \in [0, 1]$$

2.1.1 Separazione del Dataset

Il processo di addestramento di un modello può avvenire mediante tre fasi distinte:

1. training del modello su un *training set*;
2. selezione del modello migliore attraverso le performance ottenute su un *validation set*;
3. valutazione delle performance finali del modello selezionato su un *test set*;

La suddivisione dei pattern di un dataset \mathcal{D} dipende dalla tecnica usata. Nelle sezioni seguenti vedremo alcune tecniche di uso comune.

2.1.1.1 Hold-Out

Usando questa tecnica il dataset originario \mathcal{D} viene diviso in tre parti distinte:

- \mathcal{D}_{tr} che andrà a costituire il training set;
- \mathcal{D}_{val} che andrà a costituire il validation set;
- \mathcal{D}_{test} che andrà a costituire il test set;

La scelta della distribuzione dei dati su questi sottoinsiemi dipende dal problema in questione. In genere, è comunque opportuno distribuire il campione nel modo più omogeneo possibile¹ assegnando una parte importante dei dati al training set, in modo che il modello abbia un campione sufficientemente ampio per approssimare la funzione target².

Con questo sistema, modelli diversi vengono prima addestrati su \mathcal{D}_{tr} e poi selezionati in base alle performance ottenute su \mathcal{D}_{val} . Il modello ritenuto migliore viene addestrato nuovamente sul dataset composto dall'unione del training e del validation set $\mathcal{D}_{tr} \cup \mathcal{D}_{val}$, e viene testato su \mathcal{D}_{test} in modo da avere una stima della performance del modello su dati che non sono stati usati ne' direttamente, mediante l'addestramento, ne' indirettamente, mediante la selezione del modello.

2.1.1.2 k -Folds Cross-Validation

Un modo per fornire la performance di un modello usando tutti i dati del dataset, consiste nell'uso della tecnica del k -Folds Cross-Validation (CV), la quale prevede la separazione del dataset originario \mathcal{D} in k *fold* disgiunti, dove:

1. $\mathcal{D}_0 \cup \mathcal{D}_1 \cup \dots \cup \mathcal{D}_{k-1} = \mathcal{D}$
2. $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset \quad \forall i, j \in \{0, \dots, k-1\}, \quad i \neq j$

Dato un modello \mathcal{M} ed un insieme di fold ordinate, questo riceverà k addestramenti distinti, ognuno dei quali eseguito su un training set $\mathcal{D}_{tr}^{(i)}$, ottenendo così k modelli addestrati. Le performance di ogni modello, verrà successivamente valutata su un validation set $\mathcal{D}_{val}^{(i)}$. I dataset sono formati nel seguente modo:

¹ripartizioni sbilanciate, sia in termini di valori delle features, che in termini di valori target possono pregiudicare il training.

²A titolo di esempio, una ripartizione comunemente usata è di assegnare i dati secondo le seguenti percentuali (70-20-10)% sui tre dataset

- $\mathcal{D}_{tr}^{(i)} = \bigcup_{j \neq i} \mathcal{D}_j$
- $\mathcal{D}_{val}^{(i)} = \mathcal{D}_i$

Questo addestramento darà luogo a k performance distinte³

$$MaE_0(\mathcal{M}^{(0)}, \mathcal{D}_{val}^{(0)}), \dots, MaE_{k-1}(\mathcal{M}^{(k-1)}, \mathcal{D}_{val}^{(k-1)})$$

la cui media sarà la performance complessiva del modello.

Questa tecnica, sebbene utile in situazioni con dataset a bassa cardinalità, non seleziona un modello specifico, ma si limita a fornire una performance generale rispetto agli iperparametri scelti per quel modello.

Oltre a testare la performance di un singolo modello, con la CV è possibile stimare le performance di una famiglia di modelli $\bar{\mathcal{M}}$, chiamata *griglia*, con iperparametri diversi. Un modo per formare questa famiglia di modelli consiste nel selezionare un modello *campione* a cui vengono fatti variare i suoi iperparametri iniziali, in modo da comporre n modelli, a cui corrispondono tutte le combinazioni di iperparametri voluti.

2.1.1.3 Double Cross-Validation

La tecnica del *Double Cross-Validation* (DCV) suddivide il dataset \mathcal{D} in k folds *esterne*, formando k coppie training $\mathcal{D}_{tr}^{(i)}$ e validation set $\mathcal{D}_{val}^{(i)}$.

Una volta formati i k sotto-dataset esterni, per ogni training set $\mathcal{D}_{tr}^{(i)}$ vengono eseguite n CV interne, con lo scopo di selezionare il modello migliore tra quelli nella griglia.

Il modello selezionato $\mathcal{M}_{best}^{(i)}$ viene addestrato nuovamente su tutto il dataset $\mathcal{D}_{tr}^{(i)}$ e successivamente testato su $\mathcal{D}_{val}^{(i)}$.

A questo punto, per ogni modello, è possibile selezionare le performance *interne* di training e validation, mediate su ogni CV interna. Per i modelli selezionati come migliori, è possibile valutare le performance *esterne* di training e test, come media sul risultato della CV esterna.

Considerando come k' il numero di fold della CV *interna*, è possibile stimare il numero di singoli addestramenti N_{tr} con:

$$N_{tr} \approx (|\bar{\mathcal{M}}| \cdot k' + 1) \cdot k \quad (2.4)$$

³In questo contesto assumeremo di usare la *MaE* come misura di performance del modello.

dove $|\mathcal{M}|$ rappresenta la cardinalità della grid. Sebbene questa tecnica permetta di avere una stima migliore sulle performance di una particolare configurazione di iperparametri della griglia, l'elevato numero di addestramenti necessari rendono questa tecnica molto onerosa dal punto di vista computazionale.

Una variante tesa a ridurre il suo costo computazionale, prevede l'utilizzo di una separazione di tipo Hold Out al posto della CV, riguardo il loop interno, imponendo così $k' = 1$ nell'equazione (2.4).

2.2 Sintesi su Reti Neurali e Algoritmi di Apprendimento

Tra i modelli principali usati nel machine learning abbiamo le *reti neurali artificiali* (NN). Modello matematico [3] ideato prendendo ispirazione dalla controparte biologica, le NN fanno uso di una rete di *neuroni* interconnessi tra loro per determinare un valore di uscita, partendo da un insieme di feature in ingresso. In una rete *feedforward*, architettura comunemente più usata nelle NN, i neuroni (o *unità*) sono classificati in:

neuroni nascosti (o neuroni *hidden*) quando la loro uscita è connessa in ingresso ad un altro neurone;

neuroni di uscita (o neuroni *output*) quando la loro uscita costituisce la risposta complessiva della rete ad un certo input;

Il sottoinsieme di neuroni di una certa rete neurale, che condividono gli stessi ingressi, viene chiamato *layer* \mathcal{L} . Ogni neurone i è costituito da:

- un'uscita $X_i(\mathbf{l}(p))$, che rappresenta la risposta di quel neurone alle feature del pattern p ;
- una serie di $N_{in}^{(i)}$ valori di ingresso $x_0, \dots, x_{N_{in}^{(i)}-1}$, che possono essere costituiti dalle N_l feature di un certo pattern p o le uscite di altri $N_h^{(i)}$ neuroni $X_h(\mathbf{l}(p))$ collegati in ingresso all'unità. Per ogni ingresso j dell'unità i è associato un peso w_{ij} per il quale, dove servirà a scopo illustrativo, faremo distinzione tra *peso di input* \bar{w}_{ij} o *hidden* \tilde{w}_{ih} , qualora ci riferissimo ad un peso associato ad una feature

j o all'uscita di un'unità h , rispettivamente. Nell'unità è inoltre presente un peso b_i chiamato *bias*, a cui non è associato alcun ingresso⁴;

- una funzione di attivazione f , che converte la somma pesata degli ingressi $net_i(\mathbf{l}(p))$, nel valore di uscita del neurone $f(net_i(\mathbf{l}(p))) = X_i(\mathbf{l}(p))$;

I valori $X_o(\mathbf{l}(p))$ delle unità di uscita, rappresentano la risposta del modello alle feature del pattern p , identificata anche con $X_o(\mathbf{l}(p)) = y_o(p)$.

La somma pesata degli ingressi $net_i(\mathbf{l}(p))$ del neurone i in risposta alle feature $\mathbf{l}(p)$, sarà così definita:

$$net_i(\mathbf{l}(p)) = \sum_{k=0}^{N_{in}^{(i)}-1} w_{ik}x_k + b_i \quad (2.5)$$

Usando dataset omogenei, una volta fissata l'architettura della rete neurale, la cardinalità degli ingressi (e dei relativi pesi) di un certo neurone è invariante rispetto al pattern considerato. Nel caso di una rete *multi-layer*, ovvero con un numero di layer superiore ad uno, ogni layer viene ordinato partendo dal layer \mathcal{L}_0 , le cui unità possiedono solo gli ingressi delle feature, e seguendo la catena di dipendenze derivate dall'equazione di ricorrenza (2.5). Tra le funzioni di attivazione $f_i(net_i(\mathbf{l}(p)))$ principali troviamo la funzione:

Lineare

$$f_i(net_i(\mathbf{l}(p))) = net_i(\mathbf{l}(p)) \quad \in \mathbb{R}$$

Sigmoidale

$$f_i(net_i(\mathbf{l}(p))) = \frac{1}{1 + e^{-net_i(\mathbf{l}(p))}} \quad \in [0, 1]$$

Fast Sigmoidal

$$f_i(net_i(\mathbf{l}(p))) = \frac{net_i(\mathbf{l}(p))}{1 + |net_i(\mathbf{l}(p))|} \quad \in [-1, 1]$$

Tangente Iperbolica

$$f_i(net_i(\mathbf{l}(p))) = \tanh(net_i(\mathbf{l}(p))) \quad \in [-1, 1]$$

A parte la funzione lineare, che non è limitata, è possibile controllare l'intervallo di uscita della funzione di attivazione moltiplicando un opportuno parametro di *range* e aggiungendone uno di *offset* alla funzione. La *slope* (o pendenza) della funzione,

⁴alternativamente, e in modo del tutto equivalente, è possibile considerare il bias come un peso associato ad un ulteriore ingresso impostato sempre ad 1

può essere controllata similmente, moltiplicando l'ingresso per un parametro $s \in (0, 1]$ (quindi passando da $f(\text{net}_i(\mathbf{1}(p)))$ a $f(s \cdot \text{net}_i(\mathbf{1}(p)))$).

Durante l'addestramento di una rete neurale i pesi di tutti i neuroni vengono modificati in modo da minimizzare una certa funzione di errore. Chiameremo $E_o(p, t_o)$ *errore residuo* per l'uscita del modello y_o relativa al pattern p la seguente differenza:

$$E_o(p, t_o) = y_o(p) - t_o \quad (2.6)$$

L'algoritmo di apprendimento si occupa di modificare i pesi in modo da minimizzare la seguente funzione degli scarti quadratici medi \hat{R}

$$\hat{R} = \frac{1}{2N_p N_o} \sum_{o=0}^{N_o-1} \sum_{(p,t) \in \mathcal{D}} E_o(p, t_o)^2 = \frac{1}{2N_p N_o} \sum_{o=0}^{N_o-1} \sum_{(p,t) \in \mathcal{D}} (y_o(p) - t_o)^2 \quad (2.7)$$

Al fine di minimizzare \hat{R} , è possibile usare diversi algoritmi di apprendimento. In particolare vedremo alcuni *metodi iterativi*, i quali minimizzano la funzione \hat{R} procedendo iterativamente fino al raggiungimento di uno *stop criteria*, soffermandoci in particolare sui *metodi basati sul gradiente* (vedi Sez. 2.2.1).

Nel caso di unità di uscita con funzione di attivazione lineare, è possibile eseguire un loro addestramento mediante il metodo del calcolo della *pseudoinversa* (vedi Sez. 2.2.2.1), il quale fa parte della famiglia dei metodi diretti, in quanto raggiungono la soluzione in un numero di passi definiti e non necessitano di uno stop criteria.

Vedremo inoltre alcuni *metodi costruttivi* (vedi Sez. 2.3.1 e Sez. 2.3.2), i quali eseguono addestramenti locali su sottoparti di una rete neurale, modificandone la struttura fino al raggiungimento di uno stop criteria.

Nei metodi iterativi, tutti i pattern del training set vengono usati per calcolare le uscite, e quindi l'errore residuo E_o della unità di uscita della rete neurale. Il termine di questa fase iniziale, viene chiamata *epoca*.

Al termine di un'epoca, l'algoritmo di apprendimento aggiornerà i pesi e controllerà che sia soddisfatto uno degli stop criteria, ovvero una delle condizioni di terminazione, in caso contrario una nuova epoca verrà eseguita, iterando il processo.

Tra gli stop criteria usati comunemente troviamo:

- Numero massimo di epoche raggiunto;
- Raggiungimento di un valore di performance prefissato;
- Presenza di uno *stallo*, ovvero il miglioramento di performance nelle ultime epoche è inferiore ad una certa soglia;

Diversi stop criteria possono essere contemporaneamente presenti durante un addestramento, il quale si arresterà quando uno tra questi sarà soddisfatto.

I metodi diretti eseguono una prima epoca, ed usano i valori generati dalla rete per aggiornare i pesi senza che sia necessario eseguirne una seconda. I metodi costruttivi che prenderemo in considerazione fanno uso, di altri metodi (diretti o iterativi) per eseguire addestramenti locali.

Nelle prossime sezioni analizzeremo alcuni metodi, appartenenti alle suddette famiglie, che sono stati implementati nella libreria *felix* (vedere Sez. 4.1) e che sono serviti a produrre i risultati sui dataset analizzati.

2.2.1 Metodi Basati sul Gradiente

Tra i metodi iterativi, i metodi basati sul gradiente modificano i pesi delle unità della rete cercando di minimizzare la funzione dell'errore (2.7), facendo uso del suo gradiente. Questi metodi vengono anche chiamati *di discesa*, in quanto cercano, ad ogni passo, di ottenere un valore minore della funzione dell'errore rispetto a quello che aveva al passo precedente.

Per quanto riguarda il gradiente di \hat{R} , tutte le funzioni d'attivazione prese in considerazione sono derivabili, quindi è possibile calcolare le derivate parziali della generica unità di uscita o come:

$$\frac{\partial \hat{R}}{\partial w_{oi}} = \frac{1}{N_p} \sum_{(p,\mathbf{t}) \in \mathcal{D}} \underbrace{(X_o(\mathbf{l}(p)) - t_o) \cdot f'_o(\text{net}_o(\mathbf{l}(p)))}_{\delta_o(p)} \cdot \frac{\partial \text{net}_o(\mathbf{l}(p))}{\partial w_{oi}} \quad (2.8)$$

dove la derivata parziale della funzione $\text{net}_o(\mathbf{l}(p))$ è formata dal seguente insieme di equazioni, a seconda del tipo di peso w_{oi} che consideriamo:

$$\frac{\partial \text{net}_o(\mathbf{l}(p))}{\partial w_{oi}} \begin{cases} \frac{\partial \text{net}_o(\mathbf{l}(p))}{\partial \bar{w}_{oi}} = l_i(p) \\ \frac{\partial \text{net}_o(\mathbf{l}(p))}{\partial \tilde{w}_{oi}} = X_i(\mathbf{l}(p)) \\ \frac{\partial \text{net}_o(\mathbf{l}(p))}{\partial b_o} = 1 \end{cases}$$

Mentre, chiamando DS_h l'insieme degli indici delle unità che hanno collegato in ingresso l'uscita dell'unità nascosta h , abbiamo le derivate parziali dei pesi della generica unità nascosta h come:

$$\frac{\partial \hat{R}}{\partial w_{hi}} = \frac{1}{N_p} \sum_{(p,\mathbf{t}) \in \mathcal{D}} \underbrace{f'_h(\text{net}_h(\mathbf{l}(p))) \cdot \sum_{k \in DS_h} \delta_k(p) \tilde{w}_{kh}}_{\delta_h(p)} \cdot \frac{\partial \text{net}_h(\mathbf{l}(p))}{\partial w_{hi}} \quad (2.9)$$

con la derivata di $net_h(\mathbf{l}(p))$ definita in modo analogo a $net_o(\mathbf{l}(p))$. Alla fine di un'epoca t , ogni peso $w_{ij}^{(t)}$ di ogni unità della rete verrà incrementato di un certo *passo* $\Delta w_{ij}^{(t)}$

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t)} \quad (2.10)$$

Il segno del gradiente $\nabla \hat{R}$ sarà chiamata *direzione* di crescita, così come il segno delle derivate parziali saranno chiamate direzioni di crescita relative al peso. Il modo in cui il peso verrà incrementato, e quindi il valore $\Delta w_{ij}^{(t)}$ cambia a seconda dell'algoritmo che consideriamo.

2.2.1.1 Backpropagation

Nel caso dell'algoritmo di *Backpropagation* (*BackProp*) [1] seguiremo la direzione di *massima decrescita* della funzione, ovvero l'opposto del gradiente, con un certo passo $\eta > 0$. L'aggiornamento del generico peso j di un'unità i della rete neurale, all'iterazione t , sarà dunque:

$$\Delta w_{ij}^{(t)} = -\eta \cdot \frac{\partial \hat{R}}{\partial w_{ij}} = -\eta \cdot \frac{1}{N_p} \sum_{(p,t) \in \mathcal{D}} \delta_i(p) \cdot \frac{\partial net_i(\mathbf{l}(p))}{\partial w_{ij}}$$

Varianti a questo algoritmo includono l'aggiunta di un'*inerzia* alla discesa mediante l'aggiunta della modifica al passo precedente $\Delta w_{ij}^{(t-1)}$ pesata con un opportuno parametro di *Momentum* $\alpha > 0$

$$\Delta w_{ij}^{(t)} = -\eta \cdot \frac{1}{N_p} \sum_{(p,t) \in \mathcal{D}} \delta_i(p) \cdot \frac{\partial net_i(\mathbf{l}(p))}{\partial w_{ij}} + \alpha \cdot \Delta w_{ij}^{(t-1)}$$

Al fine di tenere basso il modulo dei pesi (effettuando quindi una *regolarizzazione*) è inoltre possibile modificare l'equazione dell'errore (2.7) usando la tecnica del *Weight Decay*, ovvero sommando la norma quadrata dei pesi di tutte le unità della rete, pesata con un opportuno parametro di penalizzazione $\lambda > 0$, ottenendo quindi:

$$\hat{R}^{wd} = \hat{R} + \lambda \sum_i \|\mathbf{w}_i\|^2$$

portando l'equazione di modifica dei pesi, considerando anche il Momentum, ad essere:

$$\Delta w_{ij}^{(t)} = -\eta \cdot \frac{1}{N_p} \sum_{(p,t) \in \mathcal{D}} \delta_i(p) \cdot \frac{\partial net_i(\mathbf{l}(p))}{\partial w_{ij}} - \lambda \cdot w_{ij}^{(t-1)} + \alpha \cdot \Delta w_{ij}^{(t-1)} \quad (2.11)$$

dove la componente δ_i si riferisce alla parte evidenziata in equazione (2.8) o (2.9), a seconda del tipo di unità considerato.

2.2.1.2 Resilientpropagation

L'algoritmo *Resilientpropagation* (*Rprop*) [4] esegue una discesa del gradiente prendendo in considerazione il solo segno del gradiente, trascurandone quindi il suo modulo.

Ad ogni peso j della generica unità i verrà associato un valore $\Delta_{ij}^{(t)}$ relativo all'epoca t . Ad ogni epoca, verrà valutato il segno della derivata parziale dell'errore (2.7). Se il segno è concorde con quello della derivata parziale all'epoca precedente, il valore $\Delta_{ij}^{(t)}$ verrà aumentato moltiplicandolo per un fattore $\eta^+ > 1$, altrimenti verrà ridotto moltiplicandolo per un fattore $0 < \eta^- < 1$. In sintesi:

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ \cdot \Delta_{ij}^{(t-1)} & \frac{\partial \hat{R}^{(t-1)}}{\partial w_{ij}} \cdot \frac{\partial \hat{R}^{(t)}}{\partial w_{ij}} > 0 \\ \eta^- \cdot \Delta_{ij}^{(t-1)} & \frac{\partial \hat{R}^{(t-1)}}{\partial w_{ij}} \cdot \frac{\partial \hat{R}^{(t)}}{\partial w_{ij}} < 0 \\ \Delta_{ij}^{(t-1)} & \text{altrimenti} \end{cases}$$

Questo permette di aumentare progressivamente il passo (e quindi la velocità di convergenza dell'algoritmo), diminuendolo solo quando viene superato un minimo locale (e quindi il segno della derivata risulta invertito). La modifica del peso $\Delta w_{ij}^{(t)}$ sarà costituita dal passo $\Delta_{ij}^{(t)}$ moltiplicato l'opposto⁵ del segno della derivata parziale:

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)} & \frac{\partial \hat{R}^{(t)}}{\partial w_{ij}} > 0 \\ +\Delta_{ij}^{(t)} & \frac{\partial \hat{R}^{(t)}}{\partial w_{ij}} < 0 \\ 0 & \text{altrimenti} \end{cases} \quad (2.12)$$

Prima di effettuare la modifica effettiva del peso, l'algoritmo controlla se è stato superato un minimo locale. In questo caso, il valore $\Delta w_{ij}^{(t)}$ verrà invertito di segno e, per evitare perturbazioni, la derivata parziale corrente verrà considerata nulla, nell'epoca successiva:

$$\frac{\partial \hat{R}^{(t-1)}}{\partial w_{ij}} \cdot \frac{\partial \hat{R}^{(t)}}{\partial w_{ij}} < 0 \Rightarrow \begin{cases} \Delta w_{ij}^{(t)} = -\Delta w_{ij}^{(t)} \\ \frac{\partial \hat{R}^{(t)}}{\partial w_{ij}} = 0 \end{cases}$$

A questo punto il peso w_{ij} potrà essere modificato in base alla regola (2.10).

Il passo viene inizialmente settato ad un valore fisso $\Delta_{ij}^{(0)} = \Delta_0$ e, ad ogni epoca, l'algoritmo controlla che non assuma valori troppo alti o troppo bassi, tenendolo entro

⁵Essendo l'Rprop un algoritmo di discesa del gradiente, si usa l'opposto del segno del gradiente

il range definito da due appositi parametri

$$\Delta_{min} \leq \Delta_{ij}^{(t)} \leq \Delta_{max} \quad (\text{con } 0 < \Delta_{min} \leq \Delta_0 < \Delta_{max})$$

2.2.1.3 Quickpropagation

L'algoritmo *Quickpropagation* (*Qprop*) [5] è un metodo del secondo ordine che fa uso, per ogni peso, del valore della derivata parziale all'epoca precedente. Assumendo che la funzione dell'errore (2.7), considerata per ogni peso, sia approssimabile con una parabola a coefficiente positivo, e che il cambiamento della derivata parziale rispetto ad un peso non sia influenzato dal cambiamento delle altre derivate parziali⁶, l'algoritmo usa il valore della derivata all'epoca precedente per arrivare direttamente al minimo della parabola. La modifica del peso è la seguente:

$$\Delta w_{ij}^{(t)} = \frac{\frac{\partial \hat{R}^{(t)}}{\partial w_{ij}}}{\frac{\partial \hat{R}^{(t-1)}}{\partial w_{ij}} - \frac{\partial \hat{R}^{(t)}}{\partial w_{ij}}} \cdot \Delta w_{ij}^{(t-1)}$$

Con questa regola di aggiornamento del peso, se la derivata corrente è più piccola di quella precedente, ma mantiene la stessa direzione, il peso verrà modificato nella stessa direzione. Se, al contrario, la derivata corrente è di segno opposto alla precedente, significa che un minimo locale è stato superato, e il prossimo passo collocherà la funzione in un punto tra quello corrente e quello precedente.

Usando questa formula, ci sarebbero dei problemi nel caso in cui la derivata corrente ha segno concorde con la derivata precedente, e modulo uguale o maggiore. In questo caso si avrebbe un passo di ampiezza infinita o, nel caso il modulo sia maggiore, si andrebbe nella direzione opposta a quella di decrescita. Per ovviare a questo problema, viene usato un parametro $\mu > 0$ che definisce la massima ampiezza che un passo deve avere, relativamente al passo precedente.

$$\Delta w_{ij}^{(t)} = \max \left(\frac{\frac{\partial \hat{R}^{(t)}}{\partial w_{ij}}}{\frac{\partial \hat{R}^{(t-1)}}{\partial w_{ij}} - \frac{\partial \hat{R}^{(t)}}{\partial w_{ij}}} \cdot \Delta w_{ij}^{(t-1)}, \mu \cdot \Delta w_{ij}^{(t-1)} \right) \quad (2.13)$$

All'inizio dell'algoritmo, e nel caso in cui la derivata al passo precedente sia nulla, il passo $\Delta w_{ij}^{(0)}$ sarà quello di massima decrescita (2.11). Alcune versioni di questo

⁶Assunzioni simili a quelle fatte nella Quick Propagation sono state implicitamente fatte anche per altri algoritmi

algoritmo aggiungono sempre una componente di massima decrescita al passo (2.13), tranne nel caso in cui la derivata al passo corrente sia di segno opposto a quella del passo precedente.

Per mantenere il modulo dei pesi entro un intervallo ragionevole viene usato il Weight Decay in modo simile a quanto fatto nella backpropagation.

2.2.2 Metodi Diretti

Come accennato in precedenza, nel caso in cui le unità di uscita del modello abbiano una funzione di attivazione lineare, è possibile calcolare in modo diretto il valore dei pesi di queste unità. Prendiamo in considerazione un'unità di uscita o , con \mathbf{w}_o vettore di pesi dei suoi ingressi e $A^{N_p \times N_{in}}$ matrice degli N_{in} ingressi, con N_p pattern e $\mathbf{t}^{(o)}$ vettore degli N_p target t_o contenuti nel dataset \mathcal{D} .

Sotto l'ipotesi di avere un'uscita lineare, il problema di trovare i pesi che, dato l'ingresso, forniscano in uscita i valori target è dato dal seguente sistema lineare, con \mathbf{w}_o incognita:

$$A\mathbf{w}_o = \mathbf{t}^{(o)} \quad (2.14)$$

Il problema in generale può essere mal condizionato, quindi possono esistere molteplici soluzioni. In questa sezione illustreremo brevemente due metodi diretti, usati per trovare una soluzione dell'equazione (2.14).

2.2.2.1 Calcolo della Pseudoinversa

Un modo diretto per risolvere il sistema lineare espresso nell'equazione (2.14) è usare l'inversa della matrice A :

$$\mathbf{w}_o = A^{-1}\mathbf{t}^{(o)}$$

Il problema di questo approccio è che per calcolare A^{-1} occorre un notevole impegno computazionale e che A sia non singolare, cosa non garantita nel nostro caso. Per ovviare a questi problemi è possibile usare, in sostituzione della matrice inversa A^{-1} , la *pseudoinversa di Moore-Penrose* (PI) [6] A^+ per la quale è garantita esistenza ed unicità per qualsiasi matrice A .

$$\mathbf{w}_o = A^+\mathbf{t}^{(o)} \quad (2.15)$$

Per il calcolo effettivo della matrice A^+ è possibile ricorrere a diversi metodi algebrici, come la fattorizzazione QR o passando per la *decomposizione a valori singolari* [7].

2.2.2.2 Metodo Ridge Regression

Un modo alternativo per trovare la soluzione dell'equazione (2.14) è quello di risolvere il problema dei minimi quadrati che, applicato al nostro caso, consiste nel trovare il valore di \mathbf{w}_o che minimizzi il quadrato della seguente norma:

$$\|A\mathbf{w}_o - \mathbf{t}^{(o)}\|_2^2$$

Il metodo *Ridge Regression* (RR) [8] aggiunge una penalità sulla dimensione dei valori di \mathbf{w}_o , riformulando il problema nel seguente modo:

$$\mathbf{w}_o^* = \operatorname{argmin}_{\mathbf{w}_o} (\|A\mathbf{w}_o - \mathbf{t}^{(o)}\|_2^2 + \lambda\|\mathbf{w}_o\|_2^2) \quad (2.16)$$

con $\lambda \geq 0$ parametro di regolarizzazione. Mediante l'equazione (2.16) è possibile trovare direttamente il valore di \mathbf{w}_o nel seguente modo:

$$\mathbf{w}_o = (A^T A + \lambda I)^{-1} A^T \mathbf{t}^{(o)}$$

2.3 Metodi Costruttivi

I metodi costruttivi operano partendo da una rete neurale minimale, aggiungendo ad ogni passo un'unità in più, fino al raggiungimento di un determinato stop criteria. Oltre agli stop criteria usati nei metodi iterativi, questo approccio permette di usare stop criteria più complessi, come l'*Incremental Strategy* che prevede di fermare gli addestramenti locali ad un numero di epoche sempre maggiori, a seconda del numero di unità già inserite.

L'addestramento di una nuova unità, chiamata *unità candidata*, avviene mediante l'utilizzo di un'altro algoritmo di apprendimento, come ad esempio gli algoritmi di discesa del gradiente già visti. Questo sotto-algoritmo proseguirà l'addestramento fino al raggiungimento di uno stop criteria *locale*, che determinerà l'avvenuto addestramento della nuova unità. Di contro, gli stop criteria che decretano la fine del complessivo processo di addestramento, vengono chiamati *globali*. Il modo in cui la nuova unità viene addestrata, ed aggiunta alla rete, è specifico dell'algoritmo costruttivo.

Questa classe di algoritmi è risultata particolarmente importante per gli scopi del progetto, in quanto algoritmi che operano su dati strutturati, come il NN4G (vedere Sez. 2.4.2), richiedono un addestramento incrementale al fine di ampliare il contesto analizzato.

2.3.1 Cascade Correlation

L'algoritmo di apprendimento *Cascade Correlation (CC)* [9] addestra ogni unità candida i separatamente dal resto della rete, addestrandola seguendo la massimizzazione di una opportuna funzione di *correlazione* S_i :

$$S_i = \sum_{o=0}^{N_o-1} \left| \sum_{(p,\mathbf{t}) \in \mathcal{D}} (E_o(p, t_o) - \bar{E}_o)(X_i(\mathbf{l}(p)) - \bar{X}_i) \right| \quad (2.17)$$

con \bar{X}_i e \bar{E}_o la media di, rispettivamente, l'uscita X_i e l'errore E_o su tutti i pattern del training set. La derivata parziale della funzione S , per un generico peso w_{ij} risulta essere:

$$\frac{\partial S_i}{\partial w_{ij}} = \sum_{o=0}^{N_o-1} \sigma_o \sum_{(p,\mathbf{t}) \in \mathcal{D}} (E_o(p, t_o) - \bar{E}_o) f'_i(\text{net}_i(\mathbf{l}(p))) \cdot \frac{\partial \text{net}_i(\mathbf{l}(p))}{\partial w_{ij}} \quad (2.18)$$

dove

$$\sigma_o = \text{sgn} \left(\sum_{(p,\mathbf{t}) \in \mathcal{D}} (E_o(p, t_o) - \bar{E}_o)(X_i(\mathbf{l}(p)) - \bar{X}_i) \right)$$

Tutti gli algoritmi di discesa del gradiente visti cercavano di minimizzare la funzione d'errore (2.7). Per ottenere una massimizzazione di (2.17) occorre quindi sostituire nelle equazioni di aggiornamento del peso le derivate parziali della funzione di errore (2.9) con il valore⁷ della derivata parziale di S (2.18).

L'algoritmo alterna una serie di addestramenti, prima per l'unità nascosta appena aggiunta, poi per le unità di output. Questi addestramenti terminano al sopraggiungere di uno stop criteria locale. Nel dettaglio, i passi dell'algoritmo sono i seguenti:

1. La rete viene inizializzata con una sola unità nascosta, alla quale vengono connessi i valori di input, ed una output unit, che in questa fase non verrà addestrata. Questa unità nascosta verrà addestrata facendo correlare la sua uscita con il valore target⁸, fino a raggiungimento di uno stop criteria locale;
2. I pesi delle unità di uscita verranno reimpostati ad un valore casuale, e le unità di uscita subiranno un nuovo addestramento teso a minimizzare la funzione di errore (2.7), fino al raggiungimento di uno stop criteria locale. A questo punto si controlla se una delle condizioni degli stop criteria globali sono soddisfatte, altrimenti si continua col procedimento;

⁷mentre i metodi di discesa si basano sull'opposto del gradiente, in questo caso si usa direttamente il valore del gradiente in quanto ci interessa effettuare una massimizzazione.

⁸per il primo passo della CC, l'errore nella equazione (2.17) verrà sostituito con l'opposto del valore target.

3. Una nuova unità nascosta, l'unità candidata, viene aggiunta alla rete, a cui vengono collegati in ingresso tutte le uscite delle altre unità nascoste e l'input. L'uscita dell'unità candidata non verrà collegata in ingresso alle unità di uscita. L'unità candidata verrà addestrata massimizzando la correlazione tra la sua uscita e l'errore (seguendo quindi l'equazione (2.17)). Una volta raggiunto uno stop criteria locale, l'uscita dell'unità candidata verrà collegata in ingresso a tutte le unità di uscita, e il processo riprenderà dal punto 2

L'algorithmo addestra, per ogni passo, solo i pesi di una determinata unità, lasciando gli altri *frozen*. Questo, unito al fatto che la nuova unità nascosta al passo 3 non partecipa a formare l'output della rete, permette di effettuare gli addestramenti locali in modo efficiente. Ad esempio, per addestrare l'unità nascosta, non è necessario ricalcolare ad ogni epoca l'uscita della rete.

Come detto, sia l'addestramento relativo all'unità candidata, che quello relativo all'unità di uscita, possono essere svolti con uno qualsiasi degli algoritmi di apprendimento visti fin'ora. Dato che l'inserimento di un'unità candidata ben correlata è fondamentale per il raggiungimento di buone performance⁹, è possibile, al passo 3, addestrare diverse unità candidate contemporaneamente, scegliendo di inserire nel modello quella che ottiene una migliore correlazione.

2.3.2 Cascade Residual

La tecnica del *Cascade Residual (CR)* [10] è una variante della CC dove l'unità candidata verrà addestrata, invece che massimizzando la correlazione tra l'uscita e l'errore (2.17), minimizzando direttamente l'errore residuo (2.6).

Analogamente all'algorithmo CC, questo algoritmo esegue un addestramento locale su un'unità candidata, la cui uscita non contribuisce al calcolo dell'uscita della rete neurale.

La differenza sostanziale, dal punto di vista algoritmico, sta nel fatto che l'unità candidata i viene addestrata contemporaneamente con una serie di pesi w_{oi} , inizializzati casualmente, che verranno usati come pesi associati all'uscita dell'unità candidata, quando questa verrà inserita nella rete neurale. Dato che questi pesi durante la fase di addestramento dell'unità candidata non contribuiranno al calcolo dell'uscita della rete neurale, ci riferiremo a questi come pesi d'uscita *virtuali*.

⁹l'algorithmo della CC, come molti algoritmi costruttivi, non prevede un meccanismo di rimozione di unità una volta che queste sono inserite.

Durante l'addestramento dell'unità candidata, per ogni pattern p in ingresso, si valuta l'uscita $X_o(\mathbf{l}(p))$ della generica unità di uscita o senza il contributo dell'unità candidata, il quale verrà invece considerato nel calcolo della modifica del peso w_{oh} e dei pesi propri dell'unità candidata h . A questo fine, occorre modificare la derivata parziale dell'errore residuo (2.8) in modo da considerare il contributo dell'unità candidata:

$$\frac{\partial \hat{R}^+}{\partial w_{oh}} = \frac{1}{N_p} \sum_{(p, \mathbf{l}) \in \mathcal{D}} \underbrace{(\mathbf{f}'_o(\text{net}_o^+(\mathbf{l}(p))) - t_o) \cdot \mathbf{f}'_o(\text{net}_o^+(\mathbf{l}(p)))}_{\delta_o^+(p)} \cdot X_h(\mathbf{l}(p)) \quad (2.19)$$

dove:

$$\text{net}_o^+(\mathbf{l}(p)) = \text{net}_o(\mathbf{l}(p)) + X_h(\mathbf{l}(p))w_{oh}$$

mentre, per quanto riguarda i pesi dell'unità candidata h , l'equazione (2.9) resta sostanzialmente invariata:

$$\frac{\partial E_o(p, t_o)^+}{\partial w_{hi}} = \underbrace{\mathbf{f}'_h(\text{net}_h(\mathbf{l}(p))) \cdot \sum_{o=0}^{N_o-1} \delta_o^+(p)w_{oh}}_{\delta_h^+(p)} \cdot \frac{\partial \text{net}_h(\mathbf{l}(p))}{\partial w_{hi}} \quad (2.20)$$

I passi dell'algorithm si possono dunque sintetizzare nel seguente modo:

1. La rete viene inizializzata con le unità di uscita necessarie, un insieme di pesi d'uscita virtuali, ed un'unità candidata. In questa prima fase, sia l'unità candidata, che i pesi d'uscita virtuali, verranno addestrati minimizzando il target¹⁰, fino al raggiungimento di uno stop criteria locale. Terminato l'addestramento locale, l'unità candidata verrà collegata alle unità di uscita associando i pesi virtuali addestrati alla sua uscita;
2. Una nuova unità candidata h viene generata, insieme ai vari pesi virtuali w_{oh} , la quale viene addestrata seguendo le equazioni (2.19) e (2.20), rispettivamente per i pesi virtuali d'uscita e per quelli propri di h , fino al raggiungimento di uno stop criteria locale;
3. Si controlla l'eventuale raggiungimento di uno stop criteria globale. In caso negativo, si riprende il procedimento dal punto 2;

I bias delle unità di uscita vengono resettati e addestrati nuovamente ogni volta che una nuova unità candidata viene generata, seguendo l'equazione (2.19) considerando come ingresso il valore 1 al posto dell'uscita della candidata $X_h(p)$.

¹⁰per il primo passo della CR, si calcola il residuo come se le unità di uscita dessero sempre risposta nulla a tutti i pattern in ingresso alla rete.

2.4 Domini Strutturati: Notazione e Metodologie

Fin'ora abbiamo illustrato alcune nozioni di base sull'apprendimento automatico in ambito di domini *flat*, dove un pattern p di un dataset \mathcal{D} è rappresentato da un vettore di feature $\mathbf{l}(p)$ reali.

Nei domini strutturati i dati vengono rappresentati mediante strutture dati complesse, come ad esempio *sequenze*, costituite da insiemi $s \in \mathcal{S}$ linearmente ordinati di N_s elementi:

$$s = [s_0, \dots, s_{N_s-1}]$$

ognuno dei quali è costituito da un vettore $\mathbf{l}(s_i)$ di feature.

In questa sezione forniremo la notazione usata nelle successive sottosezioni, dove illustreremo alcune delle metodologie di machine learning usate per il trattamento di domini strutturati. Per ragioni storiche introdurremo brevemente alcune tecniche per il trattamento di sequenze, concentrandoci successivamente sulle metodologie per il trattamento dei grafi, di nostro principale interesse.

Parlando di grafi, passeremo dunque da una notazione di pattern flat p a quello di grafo g rappresentato come coppia di un insieme di N_v^g vertici \mathbf{v}^g e uno di N_a^g archi \mathbf{a}^g :

$$(p, \mathbf{t}) \in \mathcal{D} \quad \Rightarrow \quad (g, \mathbf{t}) \in \mathcal{G} \quad g = (\mathbf{v}^g, \mathbf{a}^g)$$

dove il vertice $v_i \in \mathbf{v}^g$ è formato da un vettore di feature, a cui ci riferiremo con $\mathbf{l}(v_i)$, mentre l'arco $a_i \in \mathbf{a}^g$ sarà rappresentato mediante la seguente codifica:

$$a_i = (v_k, v_{k'})$$

e rappresenterà il collegamento, diretto nel caso di grafo orientato, dal vertice v_k al vertice $v_{k'}$. Definiamo l'insieme $edg(v)$ come l'insieme degli archi che legano il vertice v , ovvero

$$edg(v) = \{a \mid a = (v, u) \vee a = (u, v)\}$$

e gli insiemi $in_set(v)$ e $out_set(v)$ che, nel caso di grafo orientato, rappresentano per v :

- i vertici della stella entrante $in_set(v) = \{u \mid (u, v) \in edg(v)\}$;
- i vertici della stella uscente $out_set(v) = \{u \mid (v, u) \in edg(v)\}$;

Useremo inoltre la notazione $\mathcal{N}(v)$ per indicare l'insieme dei vertici vicini a v che, nell'esempio di un grafo non orientato, può essere espresso nel seguente modo:

$$\mathcal{N}(v) = \{u \in \mathbf{v}^g \mid (u, v) \in \text{edg}(v)\}$$

Dei modelli basati su reti neurali che illustreremo, molti calcolano l'uscita, rispetto ad un grafo g , a partire dall'analisi dei singoli vertici \mathbf{v}^g .

In questo caso, la generica unità nascosta h produrrà un vettore di *encoding* $\mathbf{x}_h(\mathbf{v}^g)$ i cui valori, o *stati*, $x_h(v)$ relativi ad un singolo vertice $v \in \mathbf{v}^g$ vengono generati attraverso una funzione di encoding, variabile a seconda del modello specifico.

Al fine di illustrare le funzioni di encoding usate, introdurremo gli operatori di *shift* q_j^{+1} e q_j^{-1} i quali, applicati ad uno stato $x_h(v)$, restituiscono lo stato relativo al j -esimo vertice della stella entrante e uscente di v , rispettivamente:

$$q_j^{+1}x_h(v) = x_h(\text{in_set}_j(v)) \quad q_j^{-1}x_h(v) = x_h(\text{out_set}_j(v))$$

Useremo inoltre la notazione abbreviata $\mathbf{q}^{\pm 1}x_h(v)$ per riferirci al *vettore di shift* riguardante tutti i vertici della stella entrante, o uscente, del vertice v per l'unità h :

$$\begin{aligned} \mathbf{q}^{+1}x_h(v) &= [q_0^{+1}x_h(v), \dots, q_{\max_{in}-1}^{+1}x_h(v)]^T \\ \mathbf{q}^{-1}x_h(v) &= [q_0^{-1}x_h(v), \dots, q_{\max_{out}-1}^{-1}x_h(v)]^T \end{aligned}$$

Analizzando i grafi, parleremo di contesto di un vertice v e in particolare di *finestra contestuale* di ampiezza c riferendoci alle variazioni di uno stato di encoding $x_i(v)$ in risposta a variazioni degli stati $\mathbf{q}^{\pm 1}x_i(v), \dots, \mathbf{q}^{\pm c}x_i(v)$.

La ricerca di modelli adattivi capaci di trattare efficacemente i domini strutturati ha portato, nel tempo, alla realizzazione di diversi approcci.

Per trattare le sequenze, alcune soluzioni [11, 12] comportano l'uso di una *sliding window*, ovvero di una normale rete feedforward al cui ingresso è collegata ad una "finestra" di un certo numero di elementi $s_i, \dots, s_{i+\delta}$ appartenenti alla sequenza s da analizzare. Il processo prevede di calcolare l'uscita della rete feedforward rispetto a quella sotto-sequenza, e di scorrere la finestra includendo nuovi elementi e generando un nuovo output. Il processo viene iterato fino a che la finestra non raggiunge la fine della sequenza.

Questi approcci, usando unicamente una rete feedforward, hanno l'indubbio vantaggio di permettere di affrontare il problema dell'apprendimento di modelli per dati strutturati usando la teoria, maggiormente collaudata, dei modelli per domini flat. Il

problema di queste soluzioni è che, usando una sliding window a dimensione fissa δ , hanno difficoltà nel cogliere interazioni molto distanti tra i dati che compongono la sequenza di input.

I modelli che tratteremo in seguito superano questi inconvenienti sfruttando, al contempo, le caratteristiche strutturali peculiari dei dati appartenenti a sequenze e grafi.

2.4.1 Metodi Avanzati per il Trattamento di Dati Strutturati

Come anticipato in precedenza, in questa sezione faremo una rapida rassegna ad alcuni tra gli approcci più usati per trattare dati strutturati, con particolare attenzione ai grafi, di nostro specifico interesse.

Partendo con una descrizione degli approcci a Kernel per il trattamento dei grafi, passeremo ad illustrare gli approcci a reti neurali. Per questa classe di metodi, partiremo illustrando alcuni modelli ideati per trattare sequenze, passando rapidamente a metodi per il trattamento dei grafi.

2.4.1.1 Metodi Basati sui Kernel

Un metodo di machine learning alternativo alle reti neurali per il trattamento di problemi di classificazione e regressione è l'approccio a *kernel*. I kernel sono funzioni simmetriche che implementano un prodotto interno tra due elementi x, y appartenenti ad un dominio I , chiamato spazio di input:

$$k(x, y) = k(y, x) = \langle \phi(x), \phi(y) \rangle \quad k : I \times I \rightarrow \mathbb{R}$$

dove ϕ è una funzione che implementa un prodotto interno proietta un elemento dallo spazio di input a quello delle features \mathcal{H}

$$\phi : I \rightarrow \mathcal{H}$$

Di fatto i kernel implementano una metrica tra due pattern diversi, appartenenti ad uno stesso dominio e affinché il kernel implementi effettivamente un prodotto interno in uno spazio delle features \mathcal{H} , occorre che soddisfi la condizione del teorema Mercer [13], ovvero che sia definito positivo. In realtà, per definire una funzione kernel non è necessario definire esplicitamente né la funzione ϕ né lo spazio delle features \mathcal{H} , potenzialmente infinito. Nella progettazione di una funzione kernel è sufficiente tenere conto delle proprietà suesposte e possedere una metrica valida nello spazio di input I .

Utilizzando la funzione kernel, è possibile usare un qualsiasi metodo di separazione lineare per trattare efficacemente problemi non lineari definiti in un qualsiasi dominio I , sia esso rappresentato da dati flat o strutturati, a patto di riuscire ad implementare una funzione kernel efficace per quel dominio.

Il metodo principale che utilizza il kernel è rappresentato dalla Support Vector Machines (SVM) [14, 15], che permette di ottenere buone performance sia nel campo dei dati flat [16] che nei domini strutturati, come la tossicologia computazionale [17].

Esempi di utilizzo dei kernel per domini strutturati sono i kernel *convoluzionali* [18], dove il kernel viene definito mediante i kernel definiti sulle componenti del dato strutturato. Parlando di grafi, kernel come Marginalized [19] e Optimal Assignment ed Expected Match [20] sono esempi di applicazioni nella chiminformatica.

L'approccio a kernel risulta particolarmente vantaggioso quando è già definita, nel dominio applicativo, una metrica per i dati che dobbiamo trattare. Non in tutti i contesti è presente, tuttavia, una metrica esplicita tra i dati [21], ed in questi casi occorre usare un certo grado di arbitrarietà nella definizione del kernel.

2.4.1.2 Recurrent Neural Network (RNN)

I modelli *Recurrent Neural Network* (RNN) [3, 22, 23] appartengono ad una classe di reti neurali che estende i modelli feedforward permettendo un apprendimento contestuale per domini strutturati.

Creati originariamente per trattare in modo efficace sequenze di dati, si distinguono dalle normali reti feedforward per l'aggiunta di N_R unità nascoste, che presentano connessioni *ricorrenti*.

Le connessioni ricorrenti collegano l'uscita di un'unità nascosta i , chiamata *unità ricorrente*, ai suoi stessi ingressi, associandovi un opportuno peso \hat{w}_i . Nel calcolare l'uscita dell'unità, questa connessione porterà in ingresso il risultato della stessa unità al passo precedente o, nel caso di una sequenza, dell'elemento precedente a quello da valutare.

Questa caratteristica conferisce ai modelli RNN la possibilità di avere uno stato interno, codificato nella matrice di encoding \mathbf{X} , comprensiva di tutti i vettori di encoding di tutte le unità ricorrenti, la quale mantiene una codifica delle informazioni riguardanti i dati passati.

Se è possibile vedere una rete feedforward come un insieme di funzioni, una per

ogni uscita, che definiscono per un certo pattern p la risposta del modello:

$$\mathbf{y}(p) = y_0(p), \dots, y_o(p), \dots, y_{N_o-1}(p)$$

nel caso delle reti ricorrenti è necessario definire anche una funzione di *transizione dello stato interno* τ che, dato un input s_i appartenente ad una certa sequenza $s = [s_0, \dots, s_{N_s-1}]$, trasforma il vettore di encoding corrente in quello relativo al nuovo input:

$$\tau(s_{i+1}, \mathbf{x}(s_i)) = \mathbf{x}(s_{i+1})$$

dove, relativamente all'unità ricorrente r , abbiamo:

$$\begin{aligned} \tau_r(s_{i+1}, \mathbf{x}(s_i)) = x_r(s_{i+1}) = \\ = \mathfrak{f}_r \left(\sum_{j=0}^{N_l-1} \bar{w}_{rj} l_j(s_{i+1}) + \sum_{h=0}^{N_h^{(r)}-1} \tilde{w}_{rh} x_h(s_i) + \hat{w}_r x_r(s_i) + b_r \right) \end{aligned}$$

Data la particolare definizione di questi modelli, occorre usare varianti specifiche degli algoritmi di apprendimento visti. Tra gli algoritmi di apprendimento principali per questo tipo di reti ricordiamo il *RealTime Recurrent Learning* (RTRL) [24], che esegue l'apprendimento basandosi sull'*unfolding* della rete, ovvero al suo sviluppo nel tempo. Il RTRL presenta, tuttavia, un alto costo computazionale. Per ovviare a questo problema, nel tempo, sono state proposte alternative del tutto equivalenti, in termini di modifiche finali dei pesi, come il *Back Propagation Through Time* (BPTT) [25], che permette di eseguire l'apprendimento con un costo computazionale inferiore¹¹, pagando tuttavia un costo maggiore in memoria.

2.4.1.3 Echo State Networks (ESN)

Il modello *Echo State Networks* (ESN) [26] è un tipo particolare di rete ricorrente, usata con successo [27, 28] per effettuare predizioni su sequenze. Seguendo il paradigma del *Reservoir Computing* (RC) [23, 29, 30], l'ESN organizza la rete in due strati principali:

- Lo strato *reservoir* contiene le N_h unità nascoste della rete, le quali sono ricorrenti, con funzione d'attivazione non lineare, e sono connesse all'input;
- Lo strato *readout* è costituito dalle unità di uscita, lineari e non ricorrenti, le quali sono connesse a tutte le unità nascoste del reservoir, ma non all'input;

¹¹L'algoritmo RTRL presenta un costo computazionale di $O(N_R^4)$, mentre il BPTT richiede solamente $O(N_R^2)$

L'ESN effettua un addestramento solo sulle unità del readout. Questo conferisce semplicità costruttiva e rapidità nell'addestramento della rete, il quale può essere eseguito, ad esempio, mediante il calcolo della pseudoinversa (vedere Sez. 2.2.2.1) riguardo unicamente i pesi delle unità di uscita.

Il reservoir conta diverse¹² unità nascoste non lineari. Queste sono connesse tra loro in modo casuale, i cui pesi sono inizializzati in modo da rispettare la seguente condizione:

$$\|\hat{\mathbf{w}}\|_2 < 1 \quad (2.21)$$

dove $\hat{\mathbf{w}}$ è la matrice dei pesi ricorrenti, per ogni unità nascosta. Questo, assieme all'utilizzo della tangente iperbolica come funzione di attivazione per le unità nascoste, permette alla seguente funzione di transizione di stato:

$$\begin{aligned} \tau_r(s_{t+1}, \mathbf{x}(s_t)) &= x_r(s_{t+1}) = \mathbf{f}_r(s_{t+1}, x_r(s_t)) = \\ &= \tanh \left(\sum_{j=0}^{N_l-1} \bar{w}_{rj} l_j(s_{t+1}) + \sum_{h=0}^{N_h^{(r)}-1} \tilde{w}_{rh} x_h(s_t) + \hat{w}_r x_r(s_t) + b_r \right) \end{aligned}$$

di essere *contrattiva*, ovvero di rispettare la seguente proprietà:

$$\begin{aligned} \exists \mu \in \mathbb{R} \quad 0 \leq \mu < 1 \quad \text{t.c.} \quad \forall s \in \mathcal{S} \quad \forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^{N_h} : \\ \|\tau(s, \mathbf{x}) - \tau(s, \mathbf{x}')\| \leq \mu \|\mathbf{x} - \mathbf{x}'\| \end{aligned}$$

Questa proprietà garantisce alla rete l'*Echo State Property* (ESP) ovvero l'indipendenza a lungo termine tra gli stati. In altre parole, col passare del tempo ($k \rightarrow \infty$) il valore assunto da uno stato futuro $x(t+k)$ dipende sempre meno dal valore dello stesso stato ad un tempo antecedente $x(t)$. Questo permette di avere un duplice vantaggio:

- Durante il training, superati un certo numero di dati di input¹³, i valori degli stati del reservoir diventano indipendenti dal valore, casuale, che avevano a tempo di inizializzazione, e restano dipendenti solo dai valori di ingresso;
- La sequenza di stati del reservoir risulta possedere un'organizzazione Markoviana, che permette di associare output simili, a sequenze di ingresso con suffissi simili.

¹²in genere un reservoir viene costruito con un numero variabile da 10^1 a 10^3 unità nascoste, a seconda del problema.

¹³La fase iniziale di training di una ESN, chiamata *washout*, serve solamente a far perdere la dipendenza degli stati del reservoir con la configurazione iniziale; durante questa fase iniziale le uscite della rete verranno pertanto ignorate.

Quest'ultima caratteristica conferisce buone performance alla ESN in quei problemi in cui la funzione target segue un comportamento Markoviano, mentre presenta un comportamento peggiore in quei casi in cui le dipendenze a lungo termine sono importanti per generare l'output [31, 32].

2.4.1.4 Recursive Cascade Correlation (RCC)

Una tecnica ideata per elaborare direttamente strutture dati più complesse delle sequenze, è il modello *Recursive Cascade Correlation* [33] (RCC), il quale fa uso della ricorsione per sfruttare il contesto su domini quali alberi e grafi orientati e aciclici.

L'architettura, nata come generalizzazione del *Recurrent Cascade Correlation* [34], prevede l'utilizzo dell'algoritmo costruttivo Cascade Correlation (vedere Sez. 2.3.1) per l'apprendimento, e fa uso di un'espressione ricorsiva per definire la funzione di transizione dello stato interno. In base a questa architettura, gli stati del vettore di encoding $\mathbf{x}(v)$, relativo ad un vertice $v \in \mathbf{v}^g$ di un certo albero di input, sono definiti mediante la seguente funzione di transizione:

$$\begin{aligned} x_i(v) &= \tau_i(\mathbf{l}(v), \mathbf{q}^{-1}[x_0(v), \dots, x_i(v)]) = \\ &= \mathbf{f}_i \left(\sum_{j=0}^{N_l-1} \bar{w}_{ij} l_j(v) + \sum_{h=0}^i \tilde{w}_{ih} \mathbf{q}^{-1} x_h(v) + b_i \right) \end{aligned} \quad (2.22)$$

Questa formulazione è basata su una visione strutturale della definizione di causalità: ovvero l'uscita $\mathbf{y}(v)$ per ogni vertice v dipende, oltre che dal vettore di etichette correnti, unicamente dalla stella uscente di v .

Se prendiamo, ad esempio, un grafo orientato, aciclico e posizionale (DPAG) come quello mostrato in Fig. 2.1b, nel modello RCC, la valutazione dello stato di un suo vertice v dipende unicamente dalle features associate a v e dallo stato calcolato sui suoi diretti discendenti. Dove, nel caso di un DPAG, per discendenti di un vertice si intendono i vertici $u \in out_set(v)$.

Analogamente, lo stato dei diretti discendenti di v è a sua volta dipendente da quello dei loro diretti discendenti, quindi il contesto relativo allo stato $x_i(v)$ è da considerarsi dipendente dallo stato di tutti vertici discendenti da v .

Questo modello permette di calcolare l'uscita analizzando direttamente la struttura del grafo, tuttavia comporta alcuni limiti nell'analisi strutturale di un grafo non riuscendo, ad esempio, a distinguere l'albero in figura 2.1a dal DPAG in figura 2.1b.

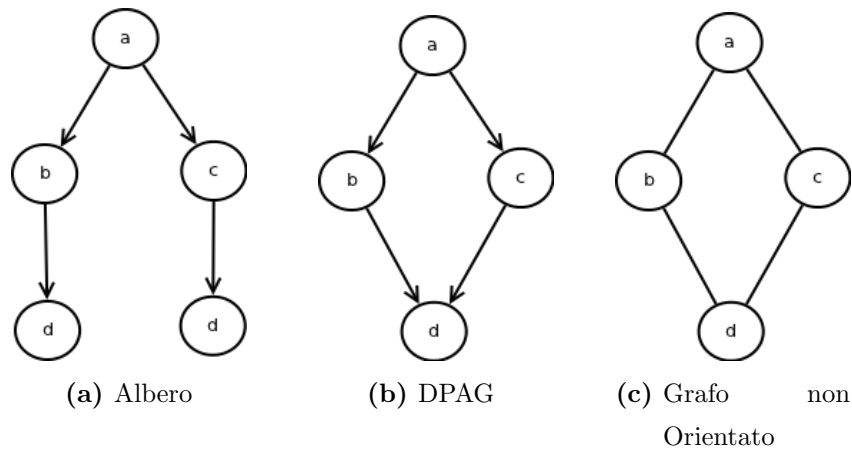


Figura 2.1: Tipologie di strutture dati che possono essere analizzate dai vari modelli. In particolare, il modello CRCC è in grado di distinguere il caso (a) dal caso (b), mentre il modello NN4G è in grado di trattare anche grafi non orientati.

2.4.1.5 Contextual Recursive Cascade Correlation (CRCC)

Un ulteriore approccio che è stato ideato, riguarda un'architettura che estende l'idea del RCC riuscendo a sfruttare meglio il contesto sui grafi. Il modello, chiamato *Contextual Recursive Cascade Correlation* [35] (CRCC), riesce a superare i limiti del modello RCC, sfruttando meglio il contesto in particolari situazioni, come il DPAG mostrato in esempio.

Osservando l'equazione (2.22) che definisce la transizione dello stato, rispetto ad un vertice v , di una generica unità nascosta del modello RCC, si nota come il modello utilizzi i valori di encoding delle unità frozen, relativamente ai vertici della stella uscente di v , per calcolare lo stato dell'unità nascosta.

Tuttavia, l'utilizzo di un approccio costruttivo, permette alle unità frozen di avere nel loro vettore di encoding, le informazioni su tutti i vertici del grafo. Usando questa caratteristica è possibile modificare l'equazione di transizione dello stato interno del modello RCC, riguardo un generico vertice v , inserendo l'informazione riguardo la sua stella entrante.

Il modello CRCC usa la seguente funzione di transizione di stato:

$$\tau_i(\mathbf{l}(v), \mathbf{q}^{-1}[x_0(v), \dots, x_i(v)], \mathbf{q}^{+1}[x_0(v), \dots, x_{i-1}(v)])$$

che prevede la stessa equazione (2.22) per definire la transizione dello stato interno

della prima unità nascosta:

$$x_0(v) = \mathbf{f}_i \left(\sum_{j=0}^{N_i-1} \bar{w}_{0j} l_j(v) + \tilde{w}_{00} q^{-1} x_0(v) + b_0 \right)$$

mentre per le successive, utilizza anche l'informazione della stella uscente di v :

$$x_i(v) = \mathbf{f}_i \left(\sum_{j=0}^{N_i-1} \bar{w}_{ij} l_j(v) + \sum_{h=0}^i \tilde{w}_{ih}^- \mathbf{q}^{-1} x_h(v) + \sum_{h=0}^{i-1} \tilde{w}_{ih}^+ \mathbf{q}^{+1} x_h(v) + b_i \right) \quad (2.23)$$

facendo distinzione tra pesi hidden relativi alla stella uscente \tilde{w}_{ih}^- e quelli della stella entrante \tilde{w}_{ih}^+ . La funzione di uscita è invece la stessa della RCC.

È importante notare che nell'equazione (2.23) nella parte relativa all'operatore di shift \mathbf{q}^{+1} non si prende in considerazione la i -esima unità per evitare dipendenze cicliche.

Per quanto riguarda il contesto, nell'architettura CRCC, oltre a quanto viene già incluso nel modello RCC, aggiunge l'informazione relativa alla codifica dei diretti predecessori di v . Questo permette di includere nel contesto, assieme al predecessore, anche tutti i suoi discendenti, ampliando notevolmente la finestra contestuale. In questo modo è possibile catturare dipendenze complesse, ad esempio nel modello RCC lo stato del vertice a assumerà lo stesso valore in entrambi i casi rappresentati nelle Fig. 2.1a e 2.1b, mentre il modello CRCC è in grado di cogliere le differenze strutturali di strutture come grafi posizionali orientati e aciclici (DPAG).

2.4.1.6 Graph Neural Network (GNN)

I modelli visti fin'ora permettono di trattare efficacemente strutture complesse come alberi, grafi aciclici e DPAG. Il problema di trattare altre classi di dati strutturati, come grafi ciclici o non orientati, viene trattato seguendo diversi approcci. Un esempio è dato dal *Graph Neural Network* [36] (GNN), dove viene implementata una funzione di transizione

$$\tau(\mathcal{g}, v) \in \mathbb{R}^m$$

che proietta un grafo \mathcal{g} ed uno dei suoi vertici v in uno spazio euclideo m -dimensionale. Per farlo, utilizza la seguente funzione di encoding:

$$x(v) = f_{\mathbf{w}}^x (\mathbf{1}(v), \mathbf{1}(\text{edg}(v)), \mathbf{1}(u), \mathbf{x}(u)) \quad \forall u \in \mathcal{N}(v) \quad (2.24)$$

dove:

- $\mathbf{l}(\text{edg}(v))$ rappresenta l'insieme delle features $\mathbf{l}(a)$ per ogni arco che collega v , ovvero $a \in \text{edg}(v)$;
- $\mathbf{l}(u)$ rappresenta l'insieme di tutte le features dei nodi adiacenti a v ;
- $\mathbf{x}(u)$ rappresenta tutti gli stati di encoding relativi ai vicini di v ;
- $f_{\mathbf{w}}^x$ funzione parametrica, con \mathbf{w} vettore dei pesi;

Il modello utilizza inoltre la seguente funzione delle uscite:

$$y_o(v) = f_{\mathbf{w}}^o(\mathbf{l}(v), x(v))$$

Considerando \mathbf{x} e \mathbf{l} come il vettore di encoding e l'insieme di tutte le features, relative a tutti i vertici e archi di un grafo, è possibile sintetizzare l'equazione (2.24) nel seguente modo:

$$\mathbf{x} = F_{\mathbf{w}}(\mathbf{x}, \mathbf{l})$$

Il teorema del punto fisso di Banach [37] garantisce l'esistenza e unicità del vettore di encoding \mathbf{x} , purchè $F_{\mathbf{w}}$ sia una mappa di contrazione, ovvero che esista un $0 \leq \mu < 1$ tale per cui:

$$\|F_{\mathbf{w}}(\mathbf{x}, \mathbf{l}) - F_{\mathbf{w}}(\mathbf{y}, \mathbf{l})\| \leq \mu \|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^m$$

Per ottenere la convergenza, il modello utilizza il metodo iterativo di Jacobi [38]

$$\mathbf{x}^{(t+1)} = F_{\mathbf{w}}(\mathbf{x}^{(t)}, \mathbf{l})$$

il quale permette di convergere allo stato finale \mathbf{x}^* , ottenuto come punto fisso ad una certa iterazione t'

$$\mathbf{x}^{(t')} \approx \mathbf{x}^{(t'+1)} = \mathbf{x}^*$$

In questo modello, la funzione f^x è implementata mediante una rete ricorrente, mentre la funzione f^o è implementata mediante una rete neurale feedforward. I collegamenti della rete f^x sono divisi in *interni*, che sono propri della rete neurale, ed i collegamenti ricorrenti detti *esterni*, che dipendono dalla topologia del grafo \mathcal{g} analizzato. La funzione f^o non ha restrizioni particolari, tuttavia il modello necessita che la rete f^x sia costruita in modo che $F_{\mathbf{w}}$ sia una mappa di contrazione, ponendo dei limiti al tipo di rete da usare.

Sebbene questo modello permetta di trattare efficacemente una più ampia classe di dati strutturati, rispetto agli approcci visti fin'ora, la sua struttura è decisamente più complessa, e questo si ripercuote nel costo computazionale dell'addestramento.

2.4.1.7 Graph Echo State Networks (GraphESN)

Il modello *Graph Echo State Networks* (GraphESN) [39] estende ai grafi l'idea usata nelle ESN per l'analisi di sequenze. Strutturalmente molto simili alle classiche ESN, le GraphESN fanno uso di unità ricorsive, nel reservoir, a posto delle unità ricorrenti.

Fissato un grado massimo k sui grafi che la rete dovrà trattare, la GraphESN sostituisce lo stato ricorrente x_r di un'unità nascosta r con k stati ricorsivi $x_r(v_0), \dots, x_r(v_{k-1})$, uno per ogni possibile vicino di un vertice del grafo. La funzione di transizione τ , rispetto ad un vertice v , diventa quindi:

$$\begin{aligned} x_r(v) &= \tau_r(\mathbf{l}(v), \mathbf{x}(u_0), \dots, \mathbf{x}(u_{k-1})) = \\ &= \tanh \left(\sum_{i=0}^{N_l-1} \bar{w}_{ri} l_i(v) + \sum_{j=0}^{k-1} \sum_{h=0}^{N_h^{(r)}-1} \tilde{w}_{rh} x_h(u_j) + \tilde{w}_{rj} x_r(u_j) + b_r \right) \end{aligned} \quad (2.25)$$

dove u_0, \dots, u_{k-1} rappresenta l'insieme dei vicini del vertice v .

Questa funzione di transizione di stato risulta direttamente calcolabile solo nel caso di grafi diretti aciclici, a causa delle dipendenze cicliche nella definizione. Tuttavia, analogamente a quanto visto nel modello GNN (vedere Sez. 2.4.1.6), la contrattività della funzione τ permette di sfruttare il *principio di contrazione di Banach* per trovare la soluzione di (2.25) mediante una ricerca del punto fisso a mezzo di iterazioni convergenti. Considerando la notazione $\mathbf{x}(v)^{\{i\}}$, in questo contesto, come l'insieme degli stati $\mathbf{x}(v)$ all'iterazione i -esima del processo, il punto fisso viene ricavato mediante il seguente metodo iterativo:

$$\begin{cases} x_r(v)^{\{0\}} = 0 & i = 0 \\ x_r(v)^{\{i\}} = \tanh \left(\sum_{i=0}^{N_l-1} \bar{w}_{ri} l_i(v) + \sum_{j=0}^{k-1} \sum_{h=0}^{N_h^{(r)}-1} \tilde{w}_{rh} x_h^{\{i-1\}}(u_j) + \tilde{w}_{rj} x_r^{\{i-1\}}(u_j) + b_r \right) \end{cases}$$

applicato ad ogni unità nascosta h del reservoir.

Sebbene questo processo, sotto l'ipotesi di contrattività della funzione, generi una sequenza di stati convergente ad un unico punto fisso, in pratica viene interrotto quando la distanza tra gli stati di un'iterazione e quelli dell'iterazione successiva sono inferiori ad una certa soglia ϵ

$$\|x_r^{\{i\}}(v) - x_r^{\{i+1\}}(v)\| \leq \epsilon \quad \forall v \in \mathbf{v}^g$$

Per calcolare l'uscita finale, il modello utilizza una *state mapping function* (SMF) ovvero una funzione usata per effettuare il *mapping* tra gli stati del reservoir in un unico

valore di uscita. La variante più semplice di SMF calcola il valore di uscita utilizzando la media dei valori degli stati, valutati per ogni vertice del grafo in ingresso.

Sebbene questo approccio erediti la semplicità costruttiva delle ESN la natura ricorsiva della funzione di encoding richiede, nel caso di grafi non orientati o ciclici, la ricerca di un punto fisso per stabilizzare lo stato del reservoir.

2.4.2 Neural Network for Graphs (NN4G)

Un ulteriore approccio usato per trattare lo stesso tipo di dati strutturati del GNN e il GraphESN, come i grafi ciclici o non orientati, è il modello *Neural Network for Graph* (NN4G) [2] il quale, diversamente da alcuni approcci visti, non utilizza connessioni ricorrenti nella funzione di transizione di stato.

L'idea di base, similmente ad alcuni modelli già visti (come RCC e CRCC), è quella di fornire un approccio costruttivo della rete al fine di sfruttare le informazioni sui vertici, presenti nelle unità frozen, per generare il vettore di encoding per l'unità candidata. In un approccio di questo tipo, l'algoritmo di apprendimento aggiungerà progressivamente una nuova unità nascosta, la quale utilizzerà nella funzione di encoding le informazioni sulle altre unità nascoste, in accordo con la topologia del grafo. Diversamente da altri modelli visti (come GNN e GraphESN), la funzione di encoding non fa uso di dipendenze cicliche nella sua definizione, e non necessita quindi di un processo di stabilizzazione.

Per generare l'elemento $x_i(v)$ di un'unità candidata i , vengono usate solo le informazioni sui vertici e gli elementi dei vettori di encoding delle unità frozen, relativamente ai vicini $\mathcal{N}(v)$ del vertice v :

$$\begin{cases} x_0(v) = \mathfrak{f}_0 \left(\sum_{j=0}^{N_i-1} \bar{w}_{0j} l_j(v) + b_0 \right) \\ x_i(v) = \mathfrak{f}_i \left(\sum_{j=0}^{N_i-1} \bar{w}_{ij} l_j(v) + \sum_{j=0}^{i-1} \tilde{w}_{ij} \sum_{u \in \mathcal{N}(v)} x_j(u) + b_i \right) \end{cases} \quad (2.26)$$

per ogni $i = 1, \dots, N_h - 1$, con N_h unità nascoste.

Questa definizione di transizione dello stato interno, oltre a non avere dipendenze cicliche, risulta essere più facilmente calcolabile rispetto a modelli come GNN e GraphESN, in quanto non richiede di effettuare la ricerca del punto fisso per stabilizzarsi.

Per quanto riguarda il *layer* di uscita, è possibile definire lo stato della generica unità di uscita y_o , rispetto al grafo \mathcal{g} come:

$$y_o(\mathcal{g}) = f_o \left(\sum_{h=0}^{N_h-1} \tilde{w}_{oh} X_h(\mathcal{g}) + b_o \right) \quad (2.27)$$

dove $X_i(\mathcal{g})$ rappresenta la somma degli stati $x_i(v)$ su tutti i vertici del grafo:

$$X_i(\mathcal{g}) = \frac{1}{k} \sum_{v \in \mathbf{v}^{\mathcal{g}}} x_i(v) \quad (2.28)$$

con k valore di normalizzazione, il quale, a seconda del problema trattato, può essere pari:

- al numero massimo di vertici tra i grafi nel training set, effettuando una normalizzazione sui grafi del dataset \mathcal{G} (variante k_{norm});
- al numero di vertici del grafo \mathcal{g} , effettuando una media (variante k_{avg});
- al valore 1, effettuando una somma diretta (variante k_{sum});

Come è possibile osservare nell'equazione (2.26), il modello prevede che lo stato $x_i(v)$ sia dipendente dal valore delle features $\mathbf{I}(v)$ e dalla codifica di tutte le unità precedenti $x_j(u)$ con $0 \leq j < i$ e $u \in \mathcal{N}(v)$. Dato che l'unità j conterrà nel suo vettore di encoding le informazioni riguardo i vicini di u , la finestra contestuale sarà tanto più ampia quanto è alto il numero di layer del modello, come mostrato in Fig. 2.2. In particolare, lo sviluppo del contesto nel modello NN_4G segue l'approccio incrementale della crescita dell'architettura, dove la prima unità usa solo l'informazione locale di ogni vertice, la seconda unità usa l'informazione locale, più il contesto relativo alla prima unità, ovvero lo stato dell'unità 1 relativo ai vicini del vertice analizzato, e questo processo continua per tutte le unità nascoste presenti nel modello. È dimostrato [2] che il contesto di ogni vertice si estende a tutti i vertici del grafo, con un numero adeguato di unità.

L'algoritmo di apprendimento presentato nell'elaborato segue da vicino quello usato nel cascade correlation, il quale utilizza un'ascesa del gradiente per l'aggiornamento dei pesi dell'unità candidata. In questo caso la funzione di correlazione (2.17) resta invariata nei suoi termini:

$$S_i = \sum_{o=0}^{N_o-1} \left| \sum_{(\mathcal{g}, \mathbf{t}) \in \mathcal{G}} (E_o(\mathcal{g}, t_o) - \bar{E}_o)(X_i(\mathcal{g}) - \bar{X}_i) \right| \quad (2.29)$$

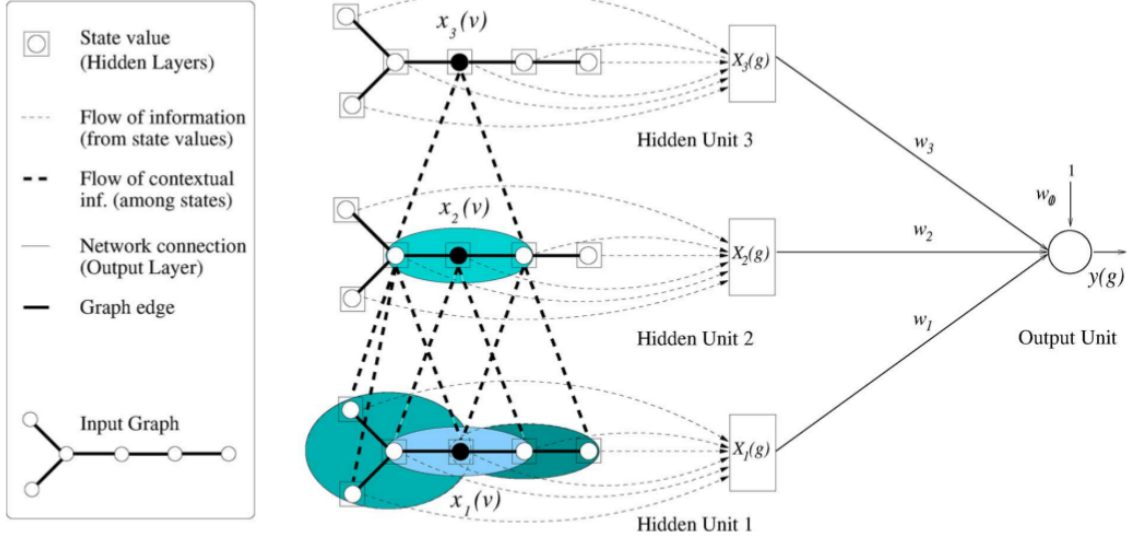


Figura 2.2: Evoluzione della finestra contestuale all'aumentare delle unità nascoste, l'immagine è presa dalla pubblicazione [2]

cambia, tuttavia, la definizione di $X_i(\mathcal{g})$, che segue la definizione vista in (2.28), e la funzione dell'errore (2.6) che in questo contesto diventa:

$$E_o(\mathcal{g}, t_o) = y_o(\mathcal{g}) - t_o$$

Considerando $f'_{(i,v)}$ come la derivata prima della funzione di encoding f relativamente all'unità nascosta i e il vertice v , possiamo scrivere la variazione dei pesi dell'unità candidata i , nel seguente modo:

$$\Delta w_{ij} = \eta \sum_{o=0}^{N_o-1} \sigma_o \left(\sum_{(\mathcal{g}, t) \in \mathcal{G}} (E_o(\mathcal{g}) - \bar{E}_o) \sum_{v \in \mathcal{V}^{\mathcal{g}}} f'_{(i,v)} \cdot I(v, w_{ij}) \right) \quad (2.30)$$

dove la parte $I(v, w_{ij})$ varia a seconda del tipo di peso considerato:

$$I(v, w_{ij}) \begin{cases} I(v, \bar{w}_{ij}) = l_j(v) \\ I(v, \tilde{w}_{ij}) = \frac{1}{k} \sum_{u \in \mathcal{N}(v)} x_j(u) \\ I(v, b_i) = 1 \end{cases}$$

Rispetto ai metodi visti in precedenza, il modello NN4G presenta una formulazione più efficiente della funzione di encoding la quale, non possedendo dipendenze cicliche, non richiede una ricerca di un punto fisso per stabilizzarsi. Inoltre, come avremo modo di vedere più avanti (vedere Sez. 5.3), la capacità di trattare direttamente grafi

permette facilmente, mediante una trasformazione algebrica, di identificare i *contributi* dei vertici di un grafo, che concorrono ad ottenere il valore di uscita del modello.

Il lavoro svolto è fondato sull'utilizzo del modello NN_4G (vedere Sez. 4 e Sez. 5).

Capitolo 3

Trattamento Dati Tossicologici

Negli anni, è stata posta un'attenzione crescente sull'impatto dei composti chimici sulla salute pubblica. Di tutti i composti chimici di uso comune, solo parte di questi sono stati sottoposti a test tossicologici [40].

A partire dal 2007¹ è stata varata una normativa europea tesa a regolamentare il trattamento di sostanze chimiche. Chiamata *REACH* (**R**egistration, **E**valuation, **A**uthorisation and **R**estriction of **C**hemical), questo regolamento pone serie restrizioni sulla produzione, ed importazione, di sostanze chimiche, richiedendo informazioni dettagliate sui composti, a tutte le parti coinvolte. Al fine di tenere sotto controllo i rischi ambientali di determinate sostanze, i produttori e importatori di prodotti chimici devono dimostrare di aver eseguito una valutazione dettagliata dei rischi connessi con le sostanze che trattano. Effettuare test sperimentali su determinate sostanze al fine di valutarne la tossicità richiede un elevato costo, sia in termini economici che di tempo. Per questo motivo la norma REACH promuove lo studio e lo sviluppo di metodologie computazionali che permettano di predire le proprietà di sostanze chimiche, che complementino, o sostituiscano, metodi lenti e costosi come quelli basati sulla sperimentazione animale.

La normativa REACH contempla diversi approcci per prevedere una proprietà di interesse, ovvero *endpoint*, di una determinata sostanza, senza doverla testare sperimentalmente, come ad esempio i metodi *read-across* e i modelli *QSAR*. Dato un insieme di diverse sostanze simili, per esempio a causa di gruppi funzionali comuni, o con proprietà fisico/chimiche o biologiche simili, è possibile classificare queste sostanze all'interno di uno stesso gruppo. Il metodo *read-across* si basa su questo concetto per

¹La normativa, da allora, ha ricevuto costanti aggiornamenti periodici.

inferire il valore di endpoint di una sostanza, basandosi sul valore dello stesso endpoint di sostanze appartenenti allo stesso gruppo della sostanza da predire.

I modelli QSAR verranno discussi nella sezione successiva, mentre in quelle seguenti illustreremo alcuni strumenti free, sviluppati implementando alcune di queste metodologie.

3.1 Tossicologia Computazionale e Modelli QSAR

La *tossicologia computazionale* [41] si occupa di costruire modelli che permettano di prevedere la tossicità dei composti chimici in base alla loro struttura e a proprietà intrinseche. Disciplina facente parte della cheminformatica, la tossicologia computazionale utilizza diversi approcci per lo studio di composti chimici, i cui modelli permettono, data una molecola, di prevedere le sue proprietà biologiche, o fisico-chimiche.

I modelli **QSAR** (**Q**uantitative **S**tructure **A**ctivity **R**elationship) [42, 43] costituiscono un insieme di approcci per inferire proprietà fisico-chimiche, o biologiche, di un composto. Secondo le direttive espresse nella normativa REACH, che si riferiscono ai principi definiti dall'*OECD* (**O**rganisation for **E**conomic **C**o-operation and **D**evelopment) [44] un modello predittivo QSAR deve possedere:

1. un **endpoint** definito. Ovvero, deve essere chiaro il sistema che il modello cerca di simulare e la proprietà che cerca di prevedere;
2. un algoritmo **non ambiguo**, inteso come *trasparenza* delle metodologie implementate nell'algoritmo che effettua la predizione sulla base delle informazioni molecolari;
3. un **dominio applicativo** definito. Per loro natura i modelli QSAR forniscono predizioni accurate solo entro alcuni parametri molecolari, come la classe di molecole trattate. L'insieme di questi parametri definiscono il dominio applicativo del modello, e deve essere espresso chiaramente nelle sue specifiche;
4. un'**interpretazione meccanicista** del modello. Sebbene non sia sempre possibile fornire un'interpretazione meccanicista del modello, è necessario fornire perlomeno delle considerazioni sulle cause meccaniciste che portano ad effettuare la predizione;

Alcune di queste direttive, come la seconda e l'ultima, sono state formulate in modo volutamente generico per non condizionare troppo la progettazione dei modelli possibili.

Altri, come la definizione di un endpoint preciso e del dominio applicativo, servono a stabilire chiaramente cosa il modello deve prevedere, ed entro quali parametri la sua predizione può essere considerata accettabile.

Tra gli endpoint principali riguardanti caratteristiche biologiche dei composti, e predetti da tool che valuteremo in seguito (come Vega in Sez. 3.2.3), troviamo:

Mutagenicità la capacità di una molecola nell'indurre mutazioni genetiche;

Carcinogenicità la capacità di una molecola di favorire l'insorgenza o la propagazione di tumori;

Sensibilizzazione Cutanea la capacità di una molecola di innescare una risposta allergica a seguito di un contatto diretto con la pelle;

Mentre, tra gli endpoint di proprietà fisico/chimiche che è possibile predire, troviamo il coefficiente $\log P$, il quale è definito come rapporto tra la concentrazione del composto disciolto nell'*ottanolo* rispetto allo stesso disciolto nell'acqua, espressa in \log_{10} :

$$\log P(M) = \log_{10} \left(\frac{[\text{soluto}(M)]_{[\text{ottanolo}]}}{[\text{soluto}(M)]_{[\text{acqua}]}} \right)$$

permette di definire il grado di solubilità di un composto in acqua², risultando essere un buon indice della sua interazione biologica.

Una rapida introduzione sui descrittori, usati in molti tool presentati nelle sezioni successive, verrà presentata in Sez. 3.1.1, mentre nella Sez. 3.1.2 accenneremo a come vengono utilizzate particolari sottostrutture molecolari per identificare possibili molecole tossiche.

3.1.1 Descrittori Molecolari

Tra i più comuni approcci alla modellazione QSAR troviamo quelli basati sui *descrittori molecolari*. Uno dei modi per trattare un'entità chimica, come una molecola, in un modello matematico, è quello di codificarla mediante alcune sue proprietà fisico chimiche. Questo ha portato all'identificazione di quelle caratteristiche fisico/chimiche che sono in qualche modo correlate ad un certo endpoint, chiamate descrittori. La caratteristica principale dei descrittori è quella di codificare una molecola in una serie di proprietà numeriche, utilizzabili all'interno di un modello matematico.

²un composto è idrofilo in modo inversamente proporzionale al suo valore di $\log P$

Esempi di semplici descrittori sono rappresentati dal peso molecolare o il momento dipolare, ma in letteratura [45] è presente un ampio numero di tipi di descrittori diversi, ed una loro trattazione completa esula dagli scopi di questo progetto.

Ad esempio, descrittori molecolari come la polarizzabilità di un composto, o il suo peso molecolare, possono essere predette direttamente, oppure usate a loro volta in un modello QSAR per stabilire un'altra proprietà di interesse. Se siamo interessati a stabilire la tossicità $T(M)$ di una certa molecola M , un semplice modello QSAR potrebbe fornire una predizione con una combinazione lineare di alcuni descrittori:

$$T(M) = \alpha Pol(M) + \beta W(M) + \gamma$$

dove $W(M)$ e $Pol(M)$ rappresentano, rispettivamente il peso molecolare e la polarizzabilità del composto M e i parametri α , β e γ sono parametri propri del modello.

L'utilizzo dei descrittori presenta, tuttavia, alcuni problemi. In primo luogo la selezione dell'insieme dei descrittori da usare è specifica dell'endpoint da predire, e necessita del supporto di un esperto del dominio. Inoltre, la codifica di una molecola nell'insieme di descrittori può portare ad una perdita di informazioni potenzialmente rilevanti per la predizione dell'endpoint.

Una particolare classe di descrittori è rappresentata dai *frammenti molecolari*.

3.1.1.1 Frammenti Molecolari

Dato un composto, è possibile determinare la presenza al suo interno di uno o più frammenti. Un frammento è una parte di molecola che ricorre frequentemente all'interno di alcune famiglie di composti chimici, e fa parte dei descrittori molecolari usati in tossicologia. È noto [46] che alcune proprietà fisico/chimiche di un composto, come il valore $\log P$, possono essere determinate dalle proprietà note dei suoi frammenti. I modelli QSAR che si basano sui frammenti vengono chiamati GQSAR (**G**roup **Q**uantitative **S**tructure**A**ctivity **R**elationship) e permettono un'ampia flessibilità nello studio delle relazioni tra i frammenti e la variazione di risposte biologiche.

È possibile codificare la presenza di determinati frammenti in una molecola mediante un vettore di *bit di presenza*, i cui elementi segnalano la presenza, o meno, di un certo frammento all'interno della molecola. Questo vettore prende il nome di *fingerprint* di una molecola, il quale viene spesso usato per definire una metrica tra diverse molecole. Uno degli indici più usati a questo scopo è l'indice di Tanimoto [47], il quale, dati due vettori fingerprint F_1 ed F_2 associati a due molecole M_1 ed M_2 , definisce la

loro somiglianza mediante la seguente equazione:

$$T(F_1, F_2) = \frac{F_1 \cdot F_2}{\|F_1\|^2 + \|F_2\|^2 - F_1 \cdot F_2} \in [0, 1]$$

dove, in questo caso, $F_1 \cdot F_2$ rappresenta il prodotto scalare tra i due fingerprint.

Approcci basati su questo descrittore, definiscono misure di similarità basate su indici come quello di Tanimoto.

3.1.2 Structural Alert

Diversi studi [48, 49] hanno mostrato una certa ricorrenza di sottostrutture molecolari, chiamate *structural alert* associate a certe caratteristiche tossicologiche. Sebbene la semplice presenza di una di queste sottostrutture non basti a conferire una proprietà tossicologica ad un composto, resta un utile indice per valutarne la sua pericolosità complessiva.

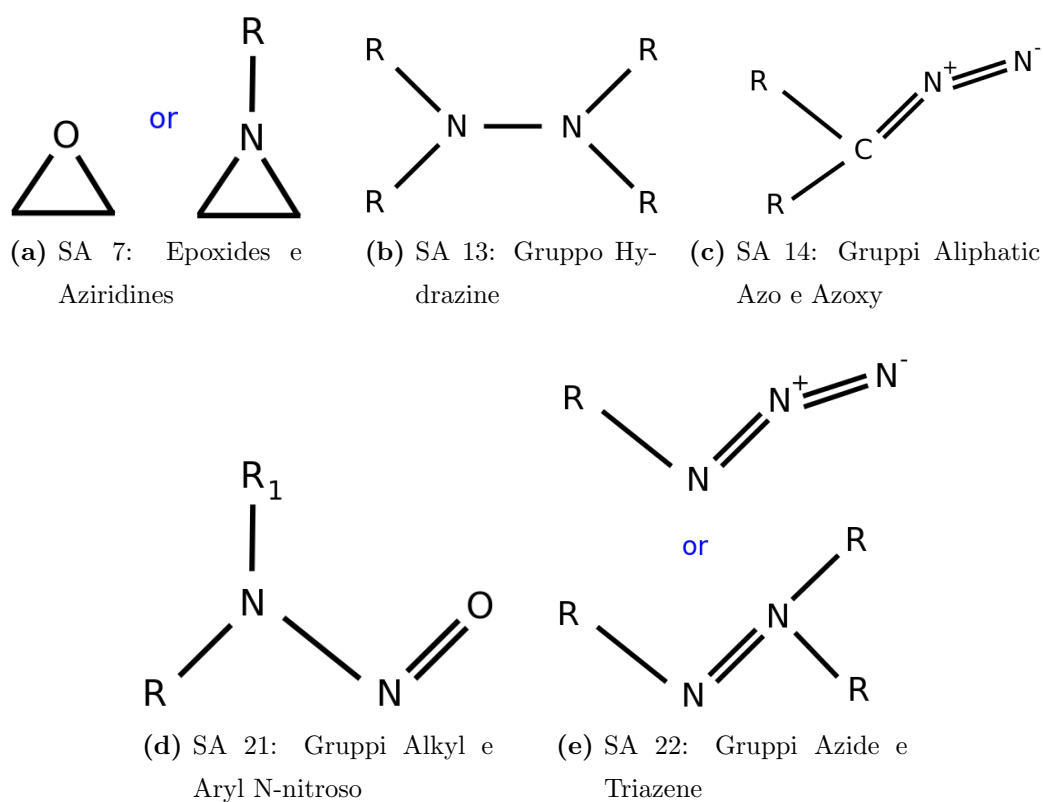


Figura 3.1: Esempi di structural alert mutageniche, dove R rappresenta un qualsiasi gruppo atomico e R_1 un carbonio alifatico o aromatico

Nel tempo sono stati realizzati elenchi [50, 51] di structural alert tossicologiche, dei quali ne vengono qui riportati alcune riguardanti la mutagenicità (visibili in Fig. 3.1) di nostro particolare interesse nell'ambito di questa tesi. Per semplicità di notazione, nel corso di questa tesi faremo riferimento numero alle structural alerts basandoci sulla numerazione riportata nell'elaborato [48].

3.2 Strumenti Disponibili

In risposta alla direttiva REACH, nel tempo, sono stati sviluppati alcuni strumenti capaci di predire alcune tra le più importanti caratteristiche dei composti. Questi strumenti prendono in ingresso la molecola da predire, in uno dei formati comunemente usati nella cheminformatica come SDF [52] o SMILES [53], e restituiscono la predizione per la proprietà di interesse, usando un modello specifico.

Nella cheminformatica il formato SDF è comunemente usato per conservare su disco un database contenente diverse molecole, mentre il formato SMILES ha il vantaggio di poter codificare una singola molecola in un'unica stringa. Formati come SMARTS sono invece usati per ricercare particolari sotto-strutture molecolari, con un formato simile a quello delle espressioni regolari.

In questa sezione faremo una breve rassegna ad alcuni tra i principali strumenti *free* disponibili, ponendo l'accento sul tipo di modelli usati per effettuare le predizioni. Accenneremo inoltre alla possibilità offerta dai seguenti tool di contestualizzare la loro predizione all'interno del loro *dominio applicativo*, ovvero di mostrare quanto la molecola predetta si discosta da quelle usate per formare i modelli.

3.2.1 EPI Suite

EPI [54] (**E**stimation **P**rograms **I**nterface) Suite è un insieme di strumenti usati per stabilire le proprietà fisico/chimiche di una sostanza, con particolare attenzione al suo impatto ambientale. Il tool, visibile in Fig. 3.2, permette di valutare un numero piuttosto ampio di endpoint, come ad esempio:

- stima del coefficiente $\log P$ del composto (si veda Sez. 3.1.1.1);
- punto di fusione, ebollizione e pressione del vapore di sostanze organiche;
- stima del tempo di dimezzamento di diverse classi di composti chimici.

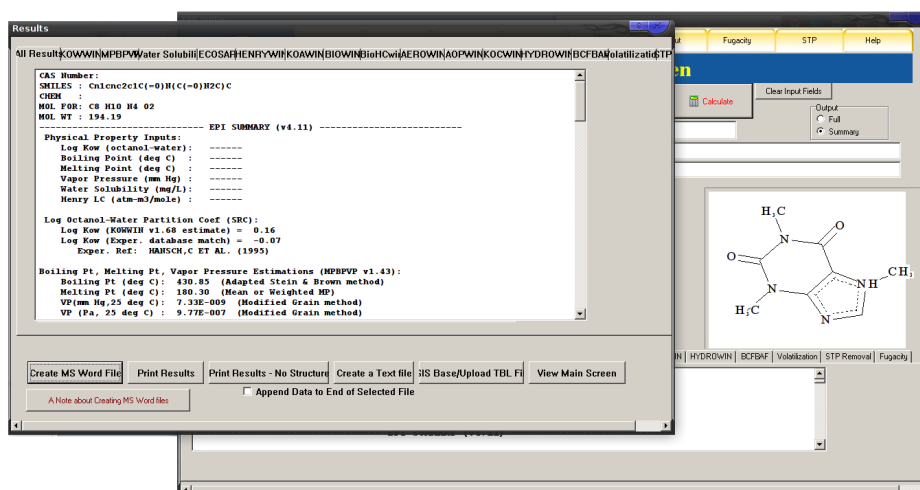


Figura 3.2: Esempio di predizione con il tool EPI Suite.

ed ogni proprietà viene calcolata con un modello specifico. Per ottenere il valore di diversi endpoint, EPI Suite fa affidamento a modelli di regressione lineare basati su dataset, utilizzando i descrittori molecolari come parametri. Questo approccio, sebbene da un lato semplifichi la costruzione dei modelli, dall'altro limita l'attendibilità della previsione.

Inoltre il tool si limita a fornire il valore della previsione senza fornire ulteriori indicazioni sul modello specifico che l'ha generata, o sull'appartenenza della molecola al suo dominio applicativo.

3.2.2 T.E.S.T.

Il sistema **T.E.S.T.** [55] (**T**oxicity **E**stimation **S**oftware **T**ool) permette all'utente, di effettuare predizioni sia di tipo tossicologico, sia rispetto ad altri parametri fisico/chimico. Tra gli endpoint tossicologici supportati abbiamo:

- tossicità dello sviluppo;
- mutagenicità;
- fattore di bioconcentrazione;

Mentre, tra le proprietà fisico/chimiche che è possibile predire, abbiamo:

- temperatura di ebollizione e di fusione;
- viscosità a 25°C;

- densità;
- solubilità in acqua a 25°C;
- conduttività termica;

Per la predizione degli endpoint, T.E.S.T. mette a disposizione diverse metodologie QSAR, tra cui:

FDA Method (**F**ood and **D**rug **A**dministration **M**ethod) La predizione è basata sulla proprietà che possiede il composto noto più simile a quello per cui è richiesta la predizione;

Single-model Method La predizione viene generata da un modello a regressione non lineare. In T.E.S.T. il modello viene addestrato usando i descrittori come features, mediante un algoritmo genetico;

Group Contribution Method La predizione viene generata da un modello a regressione non lineare. In questo caso, il modello viene addestrato usando la conta dei frammenti presenti come features;

Hierarchical Method La tossicità del composto è stimata usando una media pesata delle predizioni di diversi modelli. In T.E.S.T. i differenti modelli sono ottenuti suddividendo il dataset di addestramento in una serie di *cluster* strutturalmente simili, usando il metodo di Ward[56], e generando altrettanti modelli usando una serie di algoritmi genetici;

Nearest Neighbor Method La tossicità è predetta sulla base del valore di tossicità delle molecole più simili all'interno del training set;

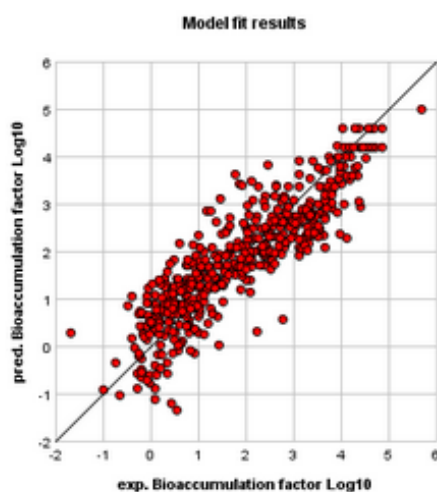
Consensus Method Il valore predetto è il risultato di una media tra i responsi di diversi altri metodi QSAR.

Random Forest Method Il valore di tossicità predetto è stimato usando un albero di decisione che separa i composti chimici entro dei *range* di valori, usando i descrittori chimici come variabili decisionali.

Il risultato viene fornito mediante una serie di file HTML che racchiudono sia il risultato della predizione, sia utili informazioni sul modello che l'ha generata.

Group contribution model

Parameter	Value
Endpoint	Bioaccumulation factor
r^2	0,771
q^2	N/A
#chemicals	492
Model	Group contribution model



Model coefficients			
Coefficient	Definition	Value	Uncertainty*
Intercept	Model intercept	1,2163	0,2593
-O- [aromatic attach]	-O- [aromatic attach] fragment	-0,5124	0,2686

Figura 3.3: La risposta del modello sulle molecole con proprietà note, rispetto al valore target.

Tra gli aspetti che differenzia T.E.S.T. dagli altri strumenti simili troviamo la possibilità di effettuare una predizione usando diversi modelli alla volta e l'attenzione che viene posta sulla trasparenza del risultato.

In questa sezione vedremo un esempio di risultato ottenuto su un composto, mediante un modello appartenente alla classe dei Group Contribution Method.

Una volta effettuata la predizione, il sistema genera un report dettagliato sul risultato. In Fig. 3.3 è possibile osservare il grafo riportato nel report, il quale mostra la distribuzione della risposta del modello rispetto ai valori target di tutte le molecole del training set. A seconda della classe di modelli usata, nel *report* generato vengono forniti i dettagli sul numero e tipo di frammenti individuati nella molecola (vedere Fig 3.5) e dei descrittori usati (vedere Fig 3.6).

[h!] In questo tool, il risultato non viene solamente espresso mediante un responso netto, piuttosto la risposta numerica del modello viene affiancata al range per il quale il composto dovrebbe essere considerato tossico secondo il modello (vedere Fig 3.7).

All'interno del report è inoltre presente l'equazione usata dal modello (visibile in Fig. 3.4) con i pesi associati alla presenza dei vari frammenti nella molecola.

Model equation:

Bioaccumulation factor = $-0,5124 \times (-O- [\text{aromatic attach}]) + 0,1984 \times (>C(=N) [\text{Nitrogen attach}]) - 0,3894 \times (-C(=O)O- [\text{nitrogen attach}]) - 1,3068 \times (-C(=O)O- [\text{aromatic attach}]) - 0,1660 \times (-CH_3 [\text{aromatic attach}]) - 0,2112 \times (-C(=S)- [2 \text{ nitrogen attach}]) - 0,9166 \times (-CF_3 [\text{aliphatic attach}]) + 1,2024 \times (P=S) - 1,0354 \times (-N< [\text{nitrogen attach}]) + 1,4535 \times (-OH [\text{sulfur attach}]) - 1,9791 \times (-COOH [\text{aromatic attach}]) - 0,2351 \times (-S(=O)(=O)- [\text{nitrogen, aromatic attach}]) + 0,0668 \times (>C=N) + 0,1571 \times (\text{Fused aromatic carbon}) - 0,2093 \times (-N< [\text{aromatic attach}]) - 0,5646 \times (-NO_2 [\text{aromatic attach}]) + 0,3252 \times (>C< [\text{aliphatic attach}]) + 0,0492 \times (ACH) - 1,3959 \times (-COOH [\text{aliphatic attach}]) - 0,1188 \times (-CH= [\text{aromatic attach}]) + 0,1727 \times (>C= [\text{aromatic attach}]) - 0,5427 \times (-O- [\text{phosphorus attach}]) - 0,9238 \times (-C(=O)- [\text{aromatic attach}]) - 0,4615 \times (C=O[\text{ketone, aliphatic attach}]) - 0,6373 \times (-NH_2 [\text{aliphatic attach}]) + 0,0874 \times (-F [\text{aliphatic attach}]) - 0,8863 \times (-NH- [\text{aliphatic attach}]) + 0,0672 \times (-Cl [\text{aromatic attach}]) - 0,8256 \times (-C\#N [\text{aromatic attach}]) - 0,5370 \times (AS) - 0,9916 \times (-OH [\text{aromatic attach}]) + 0,0327 \times (AO) - 0,1456 \times (AN) + 0,8570 \times (ACAC) - 1,6560 \times (-C(=O)- [\text{olefinic attach}]) + 0,0305 \times (P(=O)) + 0,3894 \times (AC) - 0,1338 \times (-CH_2- [\text{aromatic attach}]) - 0,0525 \times (-Cl [\text{aliphatic attach}]) - 0,5155 \times (-O- [2 \text{ aromatic attach}]) + 0,6528 \times (-C(=O)- [2 \text{ nitrogen attach}]) - 0,2459 \times (-C(=O)- [\text{nitrogen, aromatic attach}]) - 0,5843 \times (-C(=O)- [2 \text{ aromatic attach}]) - 0,3716 \times (-C\#N [\text{aliphatic attach}]) + 0,2727 \times (-CH< [\text{aliphatic attach}]) - 0,8773 \times (-N< [\text{aliphatic attach}]) + 0,1704 \times (=CH [\text{aliphatic attach}]) + 0,2065 \times (-NH- [\text{nitrogen attach}]) + 0,2326 \times (-CF_3 [\text{aromatic attach}]) - 0,4606 \times (-O- [\text{aliphatic attach}]) - 0,9258 \times (-N=N-) + 0,5259 \times (-CCl_3 [\text{aliphatic attach}]) - 1,2623 \times (-S(=O)(=O)- [\text{aliphatic attach}]) + 0,2195 \times (=C [\text{aliphatic attach}]) + 0,1358 \times (-Cl [\text{olefinic attach}]) - 0,0844 \times (-Br [\text{aliphatic attach}]) - 0,7156 \times (-NH- [\text{aromatic attach}]) - 0,3903 \times (-O- [\text{phosphorus, aromatic attach}]) - 0,4401 \times (-C(=O)O- [\text{aliphatic attach}]) - 0,0552 \times (-S- [\text{phosphorus attach}]) - 0,6726 \times (-OH [\text{aliphatic attach}]) + 0,0763 \times (-CH_3 [\text{aliphatic attach}]) + 0,0895 \times (-Br [\text{aromatic attach}]) - 0,8901 \times (-C(=O)- [\text{nitrogen, aliphatic attach}]) - 0,9462 \times (-NH_2 [\text{aromatic attach}]) - 0,1181 \times (-CH< [\text{aromatic attach}]) + 0,3199 \times (>C< [\text{aromatic attach}]) - 2,1930 \times (-S(=O)(=O)- [\text{aromatic attach}]) + 0,0060 \times (-CH_2 [\text{aliphatic attach}]) + 0,0960 \times (-CH_2- [\text{aliphatic attach}]) - 0,3305 \times (AN [\text{attached to AN in same ring}]) + 0,2449 \times (-C([H])=N [\text{Nitrogen attach}]) + 1,2163$

Figura 3.4: Esempio di equazione estesa del modello usato da T.E.S.T. disponibile nel report generato.

5. Fragments

Fragment	Count
-C(=O)- [nitrogen, aliphatic attach]	1.0
-CH ₂ - [aliphatic attach]	1.0
-CH ₃ [aliphatic attach]	2.0
-CH< [aliphatic attach]	1.0
-Cl [aliphatic attach]	1.0
-N< [aromatic attach]	1.0
AC	1.0
ACH	5.0

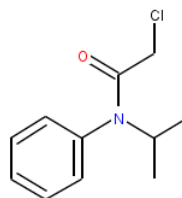


Figura 3.5: Frammenti individuati nel composto.

Constitutional Descriptors

Parameter	Value	Definition
MW	211,7100	Molecular weight
AMW	7,5611	Average molecular weight
Sv	17,3911	Sum of atomic van der Waals volumes (scaled on Carbon atom)
Se	27,9382	Sum of atomic Sanderson electronegativities (scaled on Carbon atom)
Sp	18,6477	Sum of atomic polarizabilities (scaled on Carbon atom)
Ss	33,2778	Sum of Kier-Hall electrotopological states
Mv	0,6211	Mean atomic van der Waals volume (scaled on Carbon atom)
Me	0,9978	Mean atomic Sanderson electronegativity (scaled on Carbon atom)
Mp	0,6660	Mean atomic polarizability (scaled on Carbon atom)
Ms	2,2770	Mean electrotopological state

Figura 3.6: Esempio di descrittori usati.

Predicted Bioaccumulation factor for - from Hierarchical clustering method

Prediction results

Endpoint	Experimental value	Predicted value	Prediction interval
Bioaccumulation factor Log10	N/A	1,37	$1,01 \leq \text{Tox} \leq 1,73$
Bioaccumulation factor	N/A	23,58	$10,24 \leq \text{Tox} \leq 54,30$

Cluster model predictions and statistics

Cluster model	Test chemical descriptor values	Prediction interval Log10	r ²	q ²	#chemicals
917	Descriptors	1,57 ± 0,72	0,873	0,821	10
1053	Descriptors	1,02 ± 0,97	0,814	0,758	73
1066	Descriptors	0,88 ± 0,94	0,879	0,850	118
1069	Descriptors	1,48 ± 0,84	0,895	0,865	130
1073	Descriptors	1,66 ± 0,89	0,867	0,832	151
1075	Descriptors	1,13 ± 0,93	0,864	0,829	180
1079	Descriptors	1,62 ± 1,20	0,762	0,726	274
1080	Descriptors	1,41 ± 1,20	0,732	0,694	541

Cluster models with violated constraints

Cluster Model	Test chemical descriptor values	Prediction interval Log10	r ²	q ²	# chemicals	Message
947	Descriptors	0,22 ± 0,21	0,850	0,666	5	Fragment constraint not met
966	Descriptors	-0,09 ± 0,20	0,903	0,848	8	Fragment constraint not met
985	Descriptors	-0,15 ± 0,31	0,907	0,756	9	Fragment constraint not met
1016	Descriptors	0,92 ± 1,01	0,809	0,706	23	Fragment constraint not met

Figura 3.7: Risultato della predizione espresso numericamente da T.E.S.T.

3.2.3 Vega

Completiamo questa breve rassegna con il tool Vega³ [57], il quale è uno strumento che fornisce supporto decisionale a persone esperte nella valutazione di proprietà chimiche di composti, fornendo un'ampia gamma di informazioni sul dominio applicativo, a corredo della predizione.

Per generare la predizione, Vega fa uso di modelli QSAR e strumenti read-across (vedere Sez. 3.1) allenati su diversi dataset, usando diversi descrittori per codificare le molecole. Questi modelli permettono di effettuare una predizione su diversi endpoint, sia fisico/chimici come la predizione LogP, sia biologici come:

- Mutagenicità;
- Carcinogenicità;
- Sensibilizzazione Cutanea;

³La sigla sta per Virtual Models for Evaluating the Properties of Chemical Within a Global Architecture

Il responso, assieme alle informazioni valutative, viene fornito mediante un report generato in PDF diviso in diverse capitoli, uno per ogni endpoint analizzato. Per ogni endpoint, il pdf conterrà due sezioni principali:

- Riepilogo della Predizione;
- Dominio Applicativo;

Nelle seguenti sezioni analizzeremo nel dettaglio queste parti, rilevanti ai fini della trattazione.

3.2.3.1 Riepilogo della Predizione

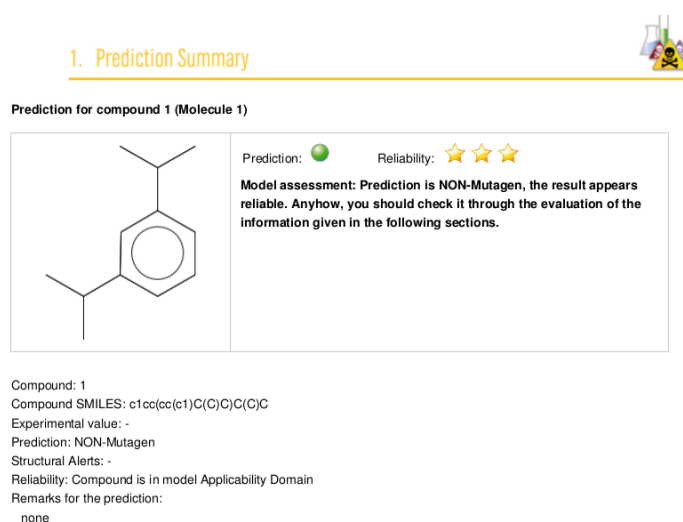


Figura 3.8: *Riepilogo*

In questa sezione del PDF generato è presente un'immagine del composto analizzato, con allegato una breve descrizione della predizione e della sua affidabilità (vedere Fig. 3.8). Fornisce inoltre informazioni sulla presenza:

- di structural alerts individuati all'interno della molecola (vedere Sez. 3.1.2);
- di valori sperimentali legati al composto analizzato;

3.2.3.2 Dominio Applicativo

Il dominio applicativo indica l'idoneità del predittore nel valutare il composto specifico, ed è definito mediante alcuni aspetti cruciali, quali:

Composti Simili Una lista di molecole simili al composto (vedi Fig. 3.9). Ognuno presenta un indice di somiglianza, il suo valore sperimentale per la proprietà che si vuole predire, e il relativo valore predetto da Vega per quella molecola;

Punteggi Valutati sul Dominio Applicativo Che contiene un riassunto delle informazioni utili a valutare la qualità della predizione (vedere Fig. 3.10), come ad esempio il numero di composti nel training set molto simili a quello fornito, quanto siano stati valutati correttamente, o se i parametri del composto siano entro i valori per cui è stato addestrato il predittore;

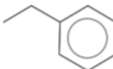

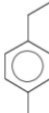
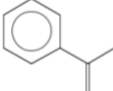
	CAS: 100-41-4 Dataset id: 3602 (training set) SMILES: <chem>c1ccc(cc1)CC</chem> Similarity: 0.91 Experimental value: NON-Mutagen Predicted value: NON-Mutagen
	CAS: 98-51-1 Dataset id: 1664 (training set) SMILES: <chem>c1cc(ccc1C(C)(C)C)C</chem> Similarity: 0.908 Experimental value: NON-Mutagen Predicted value: NON-Mutagen
	CAS: 622-96-8 Dataset id: 1285 (test set) SMILES: <chem>c1cc(ccc1C)CC</chem> Similarity: 0.898 Experimental value: NON-Mutagen Predicted value: NON-Mutagen
	CAS: 98-83-9 Dataset id: 2202 (test set) SMILES: <chem>C=C(c1ccc(C)cc1)C</chem> Similarity: 0.855 Experimental value: NON-Mutagen Predicted value: NON-Mutagen


Figura 3.9: *Composti Simili*

Vega riunisce tutti i dati utili a stabilire l'affidabilità della predizione in un insieme di indici, quali:

- Molecole simili, con un risultato sperimentale noto. Questo indice evidenzia l'eventuale presenza di molecole simili a quella predetta, tra quelle presenti nel training set su cui è stato addestrato il modello.
- L'accuratezza della predizione tra le molecole simili.
- Concordezza tra il valore predetto e quello sperimentalmente riscontrato nelle molecole simili.
- Presenza di uno o più frammenti assenti, o scarsamente presenti, nelle molecole note.

VEGA Mutagenicity model (CAESAR) (version 2.1.9) page 3

3.2 Applicability Domain: Measured Applicability Domain Scores



✓	Global AD Index AD Index = 0.951 Explanation: predicted substance is into the Applicability Domain of the model.
✓	Similar molecules with known experimental value Similarity index = 0.905 Explanation: strongly similar compounds with known experimental value in the training set have been found.
✓	Concordance for similar molecules Concordance index = 1 Explanation: similar molecules found in the training set have experimental values that agree with the predicted value.
✓	Accuracy of prediction for similar molecules Accuracy index = 1 Explanation: accuracy of prediction for similar molecules found in the training set is good.
✓	Atom Centered Fragments similarity check ACF matching index = 1 Explanation: all atom centered fragment of the compound have been found in the compounds of the training set.
✓	Model descriptors range check Descriptors range check = true

Figura 3.10: *Dominio Applicativo*

- Presenza di descrittori, nel composto predetto, i cui valori si discostano ampiamente rispetto a quelli presenti nelle molecole note.

3.3 Librerie Usate

Per gli scopi del progetto, sono state usate diverse librerie usate per la conversione dei dati chimici, come OpenBabel, per alcuni calcoli algebrici, come Armadillo, e per realizzare l'interfaccia utente del predittore, come le QT. Le sezioni successive vengono illustrate molto brevemente le principali librerie usate. Oltre a quelle presenti nelle seguenti sezioni, citiamo anche librerie come il parser RapidXML, Boost e MemUsage che sono state usate solo in limitate circostanze.

3.3.1 OpenBabel

La libreria **OpenBabel** [58] è una raccolta di strumenti per convertire dati chimici da un formato in un altro. La libreria permette di leggere un dataset chimico, memorizzando tutte le informazioni in strutture dati interne. Queste strutture dati possono essere direttamente manipolate, o convertite nuovamente in un dataset. Il punto di

forza di OpenBabel è la capacità di caricare o salvare dataset codificati in più di 100 formati differenti.

Oltre alle funzioni di conversione, OpenBabel consente di avere una buona visibilità sia dei descrittori chimici, sia dei frammenti di una molecola. Inoltre è presente la possibilità di effettuare ricerche specifiche di sotto-strutture molecolari, usando il formato SMARTS.

In questo lavoro, gli strumenti forniti da OpenBabel sono stati usati per creare gli script di conversione da formato SDF, nativo dei dataset usati per gli esperimenti, al formato GPH, usato dalla libreria *felix* (vedere Sez. 4.1). Inoltre viene usato nel predittore (vedere Sez. 5) per poter convertire le molecole di input dal formato SMILES o SDF nelle istanze della classe *Graph* di *felix*, in modo da ottenere l'uscita del modello su quei dati.

3.3.2 Armadillo

Armadillo [59] è una libreria per l'algebra lineare, che fornisce un buon compromesso tra usabilità e performance. Fornisce strutture dati efficienti per memorizzare matrici e vettori, assieme ad un buon numero di funzioni per la manipolazione delle stesse.

In questo lavoro è stato usato unicamente per l'implementazione del calcolo della pseudoinversa (vedere Sez. 2.2.2.1) nella libreria *felix* (vedere Sez. 4.1).

3.3.3 QT

Il sistema QT è un framework multipiattaforma [60] per lo sviluppo di programmi, principalmente dotati di interfaccia utente grafica, sempre più usato nello sviluppo di interfacce in ambito aziendale. Esempi di sistemi molto diffusi che usano QT possono essere: *KDE*, *Opera*, *Google Earth* e *Skype*. QT usa un'estensione del C++ standard, il codice scritto viene preprocessato in codice C++ puro. Le librerie di cui QT dispone non si limitano alla gestione dell'interfaccia utente: contengono anche classi per la gestione di stringhe, I/O, threading, connessioni di rete, containers generici, SQL, e molto altro.

Per semplificare la progettazione e implementazione di applicazioni che lo usano, il framework fa uso di widget con le quali definisce ogni singolo elemento presente sull'interfaccia destinata all'utente finale; sono esempi di *widget* i bottoni, le barre a scorrimento, gli spazi di inserimento e/o lettura di testi, e le finestre in genere.

In questo lavoro è stata utilizzata nella realizzazione della parte grafica del predittore (vedere Sez. 5) e l'analizzatore dei risultati *felix-reader* (vedere 4.3.2).

Capitolo 4

Strumenti Sviluppati per la Creazione di Modelli

Il progetto, durante il suo svolgimento, ha richiesto la realizzazione, l'addestramento e l'analisi di molte istanze del modello *NN4G*, e in questo capitolo illustreremo i principali strumenti realizzati a questo scopo. La creazione delle istanze richiedeva la realizzazione di strutture dati e algoritmi per la creazione dei modelli, per la gestione dei dataset e l'uso di diversi algoritmi di apprendimento. A questo scopo è stata realizzata una libreria, chiamata *felix*, che fornisce una serie di funzionalità di machine learning, tra cui il caricamento di dataset, la creazione di modelli e il loro addestramento. La libreria è stata sviluppata seguendo delle tecniche di progettazione che garantissero criteri di estendibilità, versatilità e semplicità d'uso.

Le funzionalità offerte dalla libreria *felix* sono usate dal tool *cougar*, il quale è stato realizzato con lo scopo di addestrare le istanze del modello *NN4G* nel modo più semplice possibile.

Tutte le librerie e tool sono stati sviluppati nel linguaggio C++ usando, dove specificato, delle librerie esterne per l'esecuzione di compiti specifici.

La libreria *felix* (descritta in Sez. 4.1) conta complessivamente circa 13000 righe di codice e fornisce una collezione di strutture e funzionalità che permettono la creazione di modelli, sia per dati flat che per grafi, e il loro addestramento, gestendo al contempo il caricamento dei dataset ed il salvataggio dei risultati.

La libreria *gfelix* (descritta in Sez. 4.2) conta più di 3000 righe di codice ed è un wrapper della libreria *felix* estendendone alcune funzionalità con lo scopo di facilitare l'integrazione con la libreria *Qt*. La separazione tra le due librerie sviluppate è stata

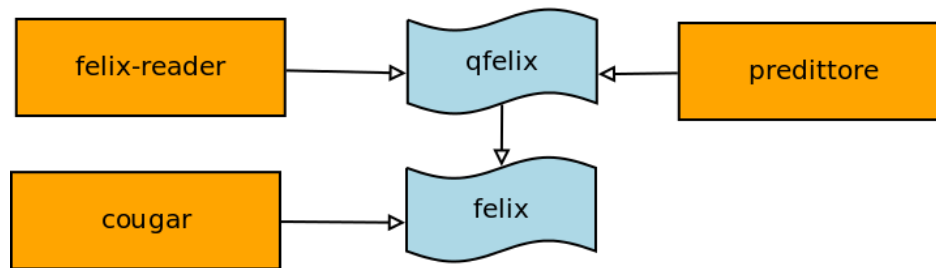


Figura 4.1: Rappresentazione dei componenti sviluppati. In blu sono rappresentate le librerie, in arancione i tool che ne fanno uso. Le frecce indicano la relazione d'uso.

introdotta per ridurre il numero di dipendenze di *felix* rispetto a librerie esterne.

Tra i tool che descriveremo, *cougar* (descritto in Sez. 4.3.1) conta circa 800 righe di codice e fa uso della libreria *felix* per eseguire gli addestramenti, mentre *felix-reader* (descritto in Sez. 4.3.2), con più di mille righe di codice, fa uso della libreria *qfelix* per fornire degli strumenti visuali di analisi post addestramento. Tra gli strumenti di analisi offerti da *felix-reader* abbiamo la visualizzazione grafica dell'andamento dell'addestramento, degli iperparametri usati e del *log* generato dall'addestramento, funzionalità implementate usando le strutture offerte da *qfelix*. Il tool *felix-reader* permette inoltre di racchiudere le principali informazioni sull'addestramento dell'istanza analizzata in un report in formato PDF, generato su richiesta dell'utente che analizza l'istanza, di cui è possibile osservare un estratto in App. F.

Una rappresentazione grafica dei componenti sviluppati, con le loro relazioni è visibile in Fig. 4.1, dove il componente *Predittore* è stato inserito per ragioni di completezza, ma verrà trattato in dettaglio nel Cap. 5.

4.1 Libreria *felix*

La libreria *felix* fornisce gli strumenti per creare ed addestrare modelli su dataset flat o strutturati, usando una buona selezione di algoritmi di apprendimento e fornendo una serie di informazioni utili per l'analisi post addestramento.

Sebbene la realizzazione del supporto per dati flat esulasse dagli scopi di questo progetto, partire dalla sua più semplice realizzazione ha garantito un controllo migliore sugli errori e ha reso più robusto lo sviluppo delle funzionalità avanzate della libreria. Una volta terminata la realizzazione della gestione dei grafi, non vi era motivo per rimuovere la parte riguardante i dati flat, così abbiamo deciso di mantenerla nella libreria a riprova delle capacità di estendibilità della stessa.

Come detto, la libreria è stata progettata con particolare attenzione all'estendibilità e alla praticità di utilizzo, mediante l'utilizzo di alcuni accorgimenti e tecniche in fase progettuale che ne hanno favorito la modularità.

L'addestramento di un modello si poggia su tre aspetti cruciali:

- la gestione dei dataset;
- la creazione dei modelli;
- l'uso degli algoritmi di apprendimento;

La libreria è stata progettata per rendere la gestione di questi aspetti semplice, ma allo stesso tempo abbastanza flessibile da poter introdurre nuove funzionalità minimizzando i moduli coinvolti.

Riguardo questi tre aspetti, al momento *felix* supporta le seguenti funzionalità:

Dataset gestione di dati flat e grafi, con separazione *Hold-out (HO)*, *k-Folds Cross-Validation* e *Double CV* (vedere Sez. 2.1.1);

Modello creazione e addestramento di modelli *Multilayer Perceptron* (vedere Sez. 2.1) e *NN4G* (vedere Sez. 2.4.2);

Apprendimento possibilità di addestramento con algoritmi di discesa del gradiente, come *Backpropagation*, *Resilientpropagation* o *Quickpropagation* (vedere Sez. 2.2.1), o algoritmi composti, come la *Cascade Correlation* (vedere Sez. 2.3.1) o la *Cascade Residual* (vedere Sez. 2.3.2) che utilizzano uno dei precedenti algoritmi per l'addestramento dell'unità candidata e di quelle di output. Per quest'ultimo tipo, nel caso della *Cascade Correlation*, la libreria fornisce anche la possibilità di addestramento con il calcolo diretto della *Pseudoinversa* (vedere Sez. 2.2.2.1);

Per favorire l'estendibilità, le istanze delle classi che implementano queste parti vengono fornite seguendo il paradigma *Abstract Factory*. Ad esempio, per ottenere l'istanza di un algoritmo di apprendimento, occorre inoltrare la richiesta ad un oggetto di tipo *AlgorithmFactory* che restituirà l'istanza opportuna, nascondendo la creazione dell'oggetto effettivo all'utente (vedere Sez. 4.1.1). Sono state predisposte abstract factory per la gestione dei dataset, dei modelli, degli algoritmi di apprendimento, e degli stop criteria che questi usano.

L'uso di questo paradigma, e l'ampio uso del polimorfismo nelle meccaniche interne, sono le caratteristiche che forniscono un alto grado di flessibilità alla libreria.

Ad esempio, gli algoritmi di apprendimento sono stati implementati per operare sulla classe astratta dei modelli *iNeuralNet*, senza nessuna informazione riguardanti il tipo effettivo del modello. Questo è valido per tutti i componenti sopra citati, e permette di minimizzare la complessità del codice dei programmi che useranno la libreria, garantendone la semplicità di estensione.

Controlli di validità vengono eseguiti a tempo di creazione, e richieste non valide, come l'utilizzo di un *NN4G* su un dataset flat, o con un algoritmo di apprendimento non costruttivo, vengono segnalate lanciando un'apposita eccezione. Difatti, qualora si volesse implementare una nuova funzionalità, come ad esempio un nuovo modello, questo approccio permette di minimizzare le modifiche necessarie alla libreria, permettendo inoltre di lasciare sostanzialmente inalterati i codici sorgenti degli strumenti che ne fanno uso (vedere Sez. 4.1.7).

Non si sarebbe potuto ottenere questo grado di astrazione senza creare un meccanismo di astrazione simile per la gestione dei parametri relativi alle varie parti. La soluzione scelta comporta l'uso, da parte dell'utente che effettua l'addestramento, di un *descrittore*, ovvero un file XML che contiene tutte le informazioni necessarie alla creazione e l'addestramento del modello e delle altre componenti (vedere Sez. 4.1.6). Questo approccio ha permesso di centralizzare in un unico file XML¹ tutte le informazioni riguardanti l'addestramento, spesso caratterizzate da un alto numero ed una certa complessità. Ad esempio, a tempo di creazione, i modelli necessitano di conoscere parametri come il range dei valori di inizializzazione dei pesi, o il tipo di funzione di attivazione, che a sua volta possiede i suoi parametri, come il coefficiente di *slope* per la sigmoideale. Per i dataset è necessario sapere il path del file system dove questo è presente, il tipo di separazione, il numero di folds nel caso di una CV o il *cutoff* nel caso di separazione HO, e altri parametri simili. Perfino il tipo stesso di modello o di algoritmo di apprendimento da usare, come *Backpropagation* o *Cascade Correlation*, è un'informazione necessaria affinché l'abstract factory fornisca l'istanza corretta.

Quasi tutte le abstract factory necessitano di un'istanza della classe *DescriptorParameter* che racchiude tutte le informazioni relative alla creazione dell'istanza opportuna. Ogni istanza di *DescriptorParameter* racchiude in se le informazioni relative ad una delle parti suddette, ed essendo una struttura ricorsiva, offre una buona capacità espressiva nella definizione dei parametri.

¹Il formato XML per il descrittore è stato scelto per le sue caratteristiche strutturali, e per la sua semplicità di creazione.

Il modo più semplice di creare i *DescriptorParameter* è quello di passare attraverso il caricamento di un descrittore (vedere Sez. 4.1.6) mediante la classe *TaskDescriptor* e usarlo per istanziare gli oggetti necessari all'addestramento, operazione che sta alla base del funzionamento di *cougar*.

Per effettuare l'addestramento di un'istanza è dunque sufficiente specificare tutti i dettagli dell'addestramento, come il modello e il tipo di algoritmo di apprendimento da usare, all'interno di un descrittore, e passare il path di quest'ultimo al tool *cougar* che fornirà delle informazioni a video sullo stato dell'addestramento e salverà automaticamente i risultati nella cartella specificata nel descrittore.

L'addestramento di un modello termina nella costruzione di un oggetto di tipo *Result*, il quale contiene le performance finali del modello, assieme a tutta una serie di informazioni sull'andamento del modello durante l'addestramento, come le performance al variare delle epoche (o delle unità candidate, nel caso di algoritmi costruttivi), o i log di addestramento registrati dall'algoritmo di apprendimento, che contiene informazioni specifiche dell'algoritmo usato.

Il tool *cougar* richiama la funzione *save* dell'oggetto *Result* per salvare tutte queste informazioni, assieme all'istanza del modello addestrato, in un pacchetto *.res*. Questo pacchetto (spiegato nel dettaglio in Sez. 4.1.5), può essere successivamente aperto mediante il tool *felix-reader* (vedere Sez. 4.3.2) per un'analisi post addestramento completa.

La prossima sezione darà un esempio di utilizzo concreto della libreria *felix* all'interno di un programma, utile nel caso si vogliano usare direttamente le funzionalità della libreria, mentre le sezioni successive descriveranno nel dettaglio la sua struttura interna. La sezione 4.1.6 sarà invece dedicata ad illustrare la struttura e l'uso del *descrittore*, di particolare interesse sia nel caso si voglia usare il tool *cougar*, sia nel caso di voler usare direttamente le funzionalità della libreria. La sezione 4.1.7 è dedicata unicamente a fornire i dettagli sulla capacità di estendibilità della libreria *felix*, ad esempio fornendo esempi sullo sviluppo di nuovi algoritmi di apprendimento.

4.1.1 Utilizzo della Libreria

Questa sezione è volta ad illustrare l'utilizzo della funzionalità offerte dalla libreria *felix*, e propone una serie di esempi di semplice utilizzo di alcune delle sue funzionalità. Come in ogni libreria C++, il primo passo per utilizzare *felix* consiste nell'includere le componenti necessarie allo scopo. Per evitare problemi di aliasing, tutte le componenti

di *felix* sono contenute all'interno del namespace *felix*, il quale sarà omesso in tutti gli esempi forniti in seguito.

Sebbene l'utilizzo di un descrittore sia suggerito in caso di addestramento, per effettuare un semplice test di un modello preaddestrato su un dataset è sufficiente istanziare direttamente gli oggetti relativi al dataset e al modello.

Ad esempio, il seguente codice rappresenta un programma completo per testare un modello *NN4G*, il cui path è passato come primo argomento, su un dataset *GraphDataset*, il cui path è il secondo argomento:

Listing 4.1: Esempio di Testing di un Modello

```
try{
    /// load the dataset
    GraphDataset* dataset=new GraphDataset(argv[2]);
    /// load the model
    NN4G* model2test=new NN4G(argv[1]);
    /// test model and print the results
    result_package* result=Result::testModelOnDataset(model2test,dataset
    );
    cout<<"Test results: "<<result->str()<<endl;

    delete result;
    delete dataset;
    delete model2test;

}catch(runtime_error e){
    cerr<<"ERROR: "<<e.what()<<endl;
}
```

La funzione statica *testModelOnDataset* è una funzione ausiliaria che permette di effettuare dei semplici test. Il risultato viene stampato a video mediante la funzione polimorfica *str* di *result_package*, e il suo formato sarà unicamente dipendente dal tipo di task del dataset.

Ad esempio, se il dataset ha un task di regressione, verrà stampato il valore di *MaE*, *MaxAbsErr*, *R* ed *S* (vedere Sez. 2.1), mentre nel caso di classificazione saranno presenti i valori di *MaE*, *MaxAbsErr* e *Acc* e, nel caso di classificazione binaria, i valori della matrice di confusione.

Descriveremo ora un esempio di addestramento ex-novo di un modello, mediante il caricamento di un descrittore. Per prima cosa si dichiarano le strutture dati che

useremo:

Listing 4.2: Dichiarazione delle Strutture Dati

```
TaskDescriptor* descriptor;
iDataset* training_set;
iDataset* validation_set;
vector<vector<iDataset*> > mainDatasets;
iNeuralNet* model;
iLearningAlgorithm* learning;
Result* final_result;
```

include le abstract factory:

Listing 4.3: Creazione delle Abstract Factory

```
DatasetFactory dsf;
ModelsFactory mfactory;
LearningAlgorithmsFactory lfactory;
```

adesso è possibile caricare il descrittore, il cui path nel file system è assunto essere il primo argomento passato al tool:

Listing 4.4: Caricamento del Descrittore dal File System

```
descriptor = new TaskDescriptor(argv[1]);
```

il dataset verrà caricato in una matrice di istanze di dataset:

Listing 4.5: Caricamento e Separazione del Dataset

```
mainDatasets = dsf.makeConcreteInstances(descriptor->dataParameters);
```

la necessità di usare una matrice è dovuto alla necessità di trattare separazione di tipo Hold Out e CV.

Nel primo caso, il training e il validation set saranno memorizzati in *mainDatasets*[0][0] e *mainDatasets*[1][0].

Nel secondo caso, seguendo la notazione usata in Sez. 2.1.1.2, abbiamo il training set $\mathcal{D}_{tr}^{(i)}$ conservato in *mainDatasets*[0][*i*], mentre il validation set $\mathcal{D}_{val}^{(i)}$ è conservato in *mainDatasets*[1][*i*].

La separazione avviene automaticamente all'interno dell'invocazione di *makeConcreteInstance* dell'abstract factory, in base alle informazioni presenti nel descrittore.

Il numero di folds è reperibile semplicemente nel seguente modo:

Listing 4.6: Recupero del Numero di Folds

```
unsigned int nFolds = mainDatasets[0].size();
```

ed è possibile testarlo per effettuare l'addestramento di tipo Hold Out o CV. Nel primo caso abbiamo:

Listing 4.7: Esecuzione del Training di Tipo Hold Out

```
if (nFolds==1) {
    training_set = mainDatasets[0][0];
    validation_set = mainDatasets[1][0];
    /// make the model
    model = mfactory.makeConcreteInstance(descriptor->modelParameters,
        training_set);
    /// make the learning algorithm
    learning = lfactory.makeConcreteInstance(descriptor->
        learningAlgorithmParameters, training_set, validation_set, model)
        ;
    /// perform the training
    learning->start_training();
    /// keep the results
    final_result = learning->results;
}
```

Come è possibile osservare, i passi prevedono:

- la creazione del modello e dell'algoritmo di apprendimento, usando le rispettive abstract factory;
- l'avvio dell'addestramento mediante l'invocazione di *start_training* dell'algoritmo di apprendimento;
- la memorizzazione del risultato, presente nell'istanza dell'algoritmo di apprendimento;

Nel caso di separazione CV, abbiamo:

Listing 4.8: Esecuzione del Training di Tipo CV

```
else{
    vector<felix::Result*> partial_results;
    for (unsigned int i=0; i<nFolds; ++i) {
```

```

    /// get the i-th training and validation set
    training_set = mainDatasets[0][i];
    validation_set = mainDatasets[1][i];
    /// make the model
    model = mfactory.makeConcreteInstance(descriptor->modelParameters,
        training_set);
    /// make the learning algorithm
    learning = lfactory.makeConcreteInstance(descriptor->
        learningAlgorithmParameters, training_set, validation_set, model);
    /// perform the training
    learning->start_training();
    /// save a result's copy
    partial_results.push_back(learning->results);
}
final_result = new ResultsCVCollector(partial_results);
}

```

In questo caso, i risultati parziali dell'addestramento sulle diverse fold, vengono conservati all'interno del vettore *partial_results*, usato a training terminato per generare un'istanza di *ResultsCVCollector*.

Indipendentemente dal tipo di training usato, il salvataggio e la stampa del risultato finale è eseguito mediante il seguente codice:

Listing 4.9: Salvataggio e Stampa del Risultato

```

string training_str, validation_str;
training_str = validation_str = "";
final_results->lastResults2str(training_str, validation_str);
cout << "Final Results:" << endl
    << "Training: " << training_str << endl
    << "Validation: " << validation_str << endl;

string resultsPath=final_result->save(descriptor);
cout << "Results saved in: " << resultsPath << endl;

```

L'esempio mostrato traslascia, per ragioni di semplicità espositiva, la gestione delle eccezioni e l'eliminazione delle strutture dati create. L'esempio completo, con il trattamento di questi aspetti e l'uso di altri piccoli accorgimenti, è disponibile in App. C.2.

Da questo esempio, è possibile notare come il codice sia indipendente dal tipo di modello e algoritmo di apprendimento usato. Queste informazioni sono contenute all'interno del descrittore, e le istanze corrette del modello e dell'algoritmo di apprendimento, con i relativi iperparametri, vengono create mediante le abstract factory.

Sempre nel descrittore, saranno presenti le informazioni relative alla posizione del dataset nel file system, al tipo di separazione da usare, all'eventuale presenza di un seed preimpostato e alla directory dove verrà salvato il risultato.

4.1.2 Gestione del Dataset

La gestione dei dataset viene implementata mediante le seguenti classi:

iDataset Classe astratta che implementa l'interfaccia per i metodi comuni a tutti i tipi di dataset supportati;

FlatDataset Sottoclasse concreta di *iDataset*, la quale implementa le funzioni di caricamento e gestione di dataset flat;

GraphDataset Sottoclasse concreta di *iDataset*, la quale implementa le funzioni di caricamento e gestione di grafi;

DatasetFactory Abstract factory usata per istanziare le sottoclassi di *iDataset* appropriate;

I formati dei file supportati sono il *.gph* per i grafi, ed il *.fds* per i dati flat. Il formato *.gph* supportato è un'estensione (descritta in App. B) del formato originario, al quale è stato fornito alcune funzionalità, tra cui la possibilità di trattare feature con valori differenti per ogni vertice². Il formato *.fds* (sigla che sta per **flat dataset**) è una variante del formato *.csv* (**c**omma **s**eparated **v**alue) dove ogni elemento del dataset (p, \mathbf{t}) viene rappresentato in una sola riga, con gli elementi del vettore di feature $\mathbf{I}(p)$ e quelli del vettore target separati da una virgola e i due vettori separati tra loro da un carattere “:”, seguendo questa codifica:

$$l_0(p), \dots, l_{N_t-1}(p) : t_0, \dots, t_{N_t-1}$$

Per caricare un dataset è possibile creare un'istanza dell'abstract factory *DatasetFactory* ed usarla per ottenere l'istanza del dataset nei seguenti modi alternativi:

²originariamente, nel formato *.gph* era possibile soltanto inserire delle tipologie fisse di vertici, con valori di feature definiti.

- usando la funzione *makeConcreteInstance*, a cui occorre passare il *Descriptor-Parameter* contenente le informazioni del dataset. Ad esempio, se abbiamo un descrittore *descriptor.cfg* (vedere Sez. 4.1.6) nella directory corrente, è possibile caricarlo ed usare l'istanza *dataParameters* per caricare il dataset nel seguente modo:

Listing 4.10: Caricamento del Dataset Mediante il Descrittore

```
TaskDescriptor* descriptor = new TaskDescriptor("descriptor.cfg")
;
iDataset* dataset = dsf.makeConcreteInstances(descriptor->
dataParameters);
```

- mediante la funzione *loadDataset*, a cui va passata la stringa relativa al path del dataset. Se ad esempio vogliamo direttamente caricare il dataset *bursi.gph* contenuto nella directory corrente, è sufficiente invocare:

Listing 4.11: Caricamento Diretto del Dataset

```
iDataset* dataset = dsf.loadDataset("bursi.gph");
```

Entrambi i modi permettono di caricare il dataset senza doverne specificare il tipo, il quale verrà estrapolato automaticamente. Nel primo caso il tipo verrà letto dal descrittore:

Listing 4.12: Estratto del Tag Data di un Descrittore

```
<Data name="GraphDataset" type="gph" path="/home/barontil/tesi/data/
bursi/bursi.gph" shuffle="False">
```

mentre nel secondo caso il sistema farà riferimento all'estensione del file. In entrambi i casi, le funzioni restituiscono un'istanza di tipo apparente *iDataset* che l'utente potrà utilizzare per addestrare (o testare) il modello.

Tra le funzioni di interfaccia comuni, abbiamo quelle usati per controllare i dati, tra cui *at*, *add* e *clear*, quelle per dividere un dataset in due parti, come *split*, e quelle per unire due dataset in uno, come *merge*, o la semplice funzione *shuffle* per mescolare i dati.

Sono inoltre presenti funzioni ausiliarie, usate per facilitare le operazioni più comuni nell'ambito del machine learning. Ne è un esempio la funzione *getCVdatasets* che suddivide il dataset in cui è invocata in due vettori di dataset, implementando la separazione di tipo Cross-Validation³ (vedere Sez. 2.1.1.2).

Ogni sottoclasse concreta di *iDataset* è obbligata ad implementare le funzioni *save* e *load*, necessariamente specifiche del tipo di dataset usato.

Per usare questo insieme di funzionalità comuni è sufficiente agire sul tipo apparente dell'istanza, mentre occorre effettuare cast espliciti per ottenere accesso a funzionalità specifiche, come ad esempio la funzione *generateRandomGraph* della sottoclasse concreta *GraphDataset*, usata per generare dataset casuali per debugging e fare test rapidi.

La gestione dei pattern contenuti nei dataset avviene seguendo la stessa politica. I vari pattern all'interno della classe *iDataset* sono codificati mediante istanza della classe astratta *iDataPattern* che, allo stesso modo, racchiude le strutture e funzionalità comuni ai pattern flat e ai grafi.

Sottoclassi concrete di *iDataPattern* sono *FlatDataPattern* e *Graph*, rispettivamente per i pattern flat e grafi.

4.1.3 Gestione dei Modelli

La libreria implementa due tipi di reti neurali usate per la gestione di dati flat e grafi, fornendo le relative istanze in modo analogo a quanto visto nei dataset.

Una classe astratta *iNeuralNet* implementa le interfacce comuni alle due sottoclassi concrete *MultiLayerPerceptron* e *NN4G*. Analogamente a quanto visto nella gestione dei dataset, è presente una classe *ModelsFactory* che implementa l'abstract factory usata per istanziare il modello. Anche in questo caso il modello può essere istanziato mediante l'uso del descrittore:

Listing 4.13: Creazione del Modello Mediante il Descrittore

```
DatasetFactory mf;  
iNeuralNet* model = mf.makeConcreteInstance(descriptor->dataParameters  
    , dataset);
```

o caricandolo direttamente da file, ad esempio il file *saved_model.xml* presente nella directory corrente:

³La separazione Hold Out è implementata mediante la funzione *split*

Listing 4.14: Caricamento del Modello da File

```
DatasetFactory mf;
iNeuralNet* model=mf.makeConcreteInstance("saved_model.xml");
```

Nel primo caso, è necessario passare all'abstract factory anche un'istanza del dataset su cui il modello dovrà operare. Questa istanza serve unicamente a fornire alcune informazioni dei dati al modello, come la cardinalità delle features e dei valori target, nel caso del *NN4G*.

La classe *iNeuralNet* racchiude le informazioni strutturali della rete neurale all'interno di due vettori pubblici, uno contenente i layers:

Listing 4.15: Matrice dei Layer

```
vector<vector<iNeuralUnit*>> layers;
```

ed uno contenente le unità di uscita:

Listing 4.16: Vettore delle Unità di Uscita

```
vector<iNeuralUnit*> outputs;
```

La visibilità pubblica di queste strutture garantisce la massima flessibilità da parte dell'algorithmo di apprendimento, specialmente nel caso di algoritmi costruttivi che agiscono direttamente sull'architettura della rete.

Tra le funzioni comuni più importanti di questa classe troviamo: *createNewUnit* che restituisce una nuova unità pronta ad essere aggiunta alla rete da un algoritmo costruttivo, o la funzione *save* che permette di salvare il modello su file. La funzione di utilità *makeIterator* permette di creare un iteratore sulle unità del modello. Utile nel caso si voglia effettuare lo stesso tipo di operazione su tutte le unità della rete, come l'aggiornamento del valore dei pesi all'interno dei metodi del gradiente:

Listing 4.17: Esempio di Uso dell'Iteratore delle Unità di un Modello

```
iNeuralUnit* currentUnit;
UnitsIterator iter = model->makeIterator();
while(iter.hasNext()){
    currentUnit = iter.next();
    if(!currentUnit->frozen){
        trainUnit(currentUnit,true);
    }
}
```

```

}
```

Ogni sottoclasse concreta di *iNeuralNet* deve implementare la funzione virtuale pura *input* che viene invocata quando si vuole conoscere la risposta del modello ad un pattern.

Dopo che la funzione viene invocata, la risposta del modello viene conservata nel vettore⁴ pubblico

Listing 4.18: Vettore dei Valori di Uscita del Modello

```
vector<double> results;
```

Ad esempio, il codice seguente stampa a video la risposta del modello *model* (assunto ad output singolo) a tutti i pattern presenti in *dataset*:

Listing 4.19: Esempio di Stampa della Risposta del Modello

```
for(unsigned int i=0; i<dataset->size(); ++i){
    model->input(dataset->at(i));
    cout << model->results[0] << endl;
}
```

4.1.3.1 Gestione delle Unità

La classe astratta *iNeuralUnit* rappresenta la singola unità della rete neurale ed implementa le funzionalità comuni alle sottoclassi concrete *Perceptron* e *NN4GUnit* usate, rispettivamente nelle classi *MultiLayerPerceptron* e *NN4G*. A seconda del tipo di rete neurale usata, il tipo effettivo delle unità contenute nei vettori *layers* e *outputs* sarà relativo ad una delle due sottoclassi concrete di *iNeuralUnit*.

La funzione *createNewUnit* di *iNeuralUnit* permette agli algoritmi di apprendimento di generare unità del tipo effettivo corretto, senza dover conoscere il tipo di rete su cui stanno operando. Quando un algoritmo costruttivo necessita di una nuova unità, invoca *createNewUnit* sul modello

Listing 4.20: Esempio di Creazione di una Nuova Unità

```
iNeuralUnit* currentUnit = model->createNewUnit(iNeuralUnit::Hidden);
```

⁴Solo nel caso di modelli multiooutput la dimensione del vettore *results* è superiore ad uno.

il quale restituisce l'unità del tipo effettivo appropriato, copiandola da un'unità *template*, esterna alla rete, generata a tempo di creazione del modello.

Il calcolo dell'uscita del modello avviene chiamando, dall'interno della funzione *input* del modello, la funzione astratta *compute* di ogni unità della rete, partendo dai layer più bassi, arrivando alle unità di uscita. Invocando la funzione *compute* l'unità aggiorna la sua uscita, contenuta nella sua variabile pubblica *currentValue*.

Gestione dei Pesi I pesi di un'unità della rete neurale sono contenuti all'interno della classe *iNeuralUnit* distinguendo da peso di output, hidden e il bias:

Listing 4.21: Strutture dei Pesi all'Interno di un'Unità

```
vector<weight*> inputWeights;
vector<weight*> hiddenWeights;
weight* bias;
```

Il tipo *weight* è una *struct* definita in *iNeuralUnit* contenente il valore del peso, contenuto nella variabile pubblica *value*, e due variabili usate in modo esclusivo, a seconda del tipo di peso appartenente alla generica unità *i*:

- nel caso il peso \bar{w}_{ij} sia di input, viene usata la variabile *input* per conservare il valore della features l_{ij} ;
- nel caso il peso \tilde{w}_{ij} sia di hidden, viene usata la variabile *inHiddenReference* che contiene un riferimento all'unità *j*;

All'interno della *struct weight* sono conservate le informazioni riguardanti la derivata parziale rispetto a quel peso, in modo da agevolare le operazioni degli algoritmi di apprendimento che non dovranno usare strutture ausiliari (e parallele) per conservare i valori del gradiente.

4.1.3.2 Salvataggio e Caricamento

I modelli possono essere salvati in files xml e caricati in modo da essere usati successivamente per effettuare predizioni.

Il salvataggio viene effettuato invocando la funzione *save* del modello, la quale si occupa di salvare alcune informazioni generiche del modello, come il path del dataset su cui è stato addestrato, o i contributi massimi e minimi (vedere Sez. 5.3). Si occupa,

inoltre, di salvare lo stato di ogni unità invocando la funzione *save* per ogni *iNeuralUnit* presente.

Il caricamento avviene invocando la funzione *loadModel* di un'istanza di *ModelsFactory*.

Un esempio completo di XML dei un modello NN4G è presente in App. C.1.

4.1.4 Gestione degli Algoritmi di Apprendimento

Una volta istanziato un modello, è possibile passarlo ad un algoritmo di apprendimento per eseguire l'addestramento.

Anche in questo caso si è ricorsi all'uso di un'abstract factory *LearningAlgorithmsFactory*, la cui invocazione della funzione *makeConcreteInstance* restituisce l'istanza dell'algoritmo appropriata, in base a quanto descritto nel *DescriptorParameter* passato:

Listing 4.22: Creazione dell'Algoritmo di Apprendimento

```
LearningAlgorithmsFactory laf;  
iLearningAlgorithm* learning;  
learning= laf.makeConcreteInstance(descriptor->  
    learningAlgorithmParameters, trainingset, validationset, model);
```

Oltre all'istanza del modello, l'algoritmo di apprendimento necessita, a tempo di creazione, delle istanze del training e validation set, necessari per effettuare l'addestramento.

La classe astratta *iLearningAlgorithm* contiene le principali funzionalità comuni a tutti gli algoritmi di apprendimento. Ad esempio, una volta istanziato l'algoritmo di apprendimento, per far partire l'addestramento è sufficiente invocare la funzione *start_training*:

Listing 4.23: Esecuzione del Training

```
learning->start_training();
```

Il risultato dell'addestramento può essere prelevato dall'istanza *results* della classe *Result* (per i dettagli vedere Sez. 4.1.5) presente all'interno dell'istanza dell'algoritmo.

Le seguenti righe di codice, ad esempio, stampano il risultato finale dell'addestramento:

Listing 4.24: Esempio di Stampa dei Risultati di Training

```

learning->start_training();
/// now the training is over
Result* results = learning->results;
std::cout<< "Training terminated, stop criteria met: "
  << results->finalStopCriteria << endl
  << "\tFinal Training Result: " << endl
  << "\t\t" << results->lastTrainingResults->str() << endl
  << "\tFinal Validaiton Result: " << endl
  << "\t\t" << results->lastValidationResults->str() << endl;

```

il risultato a video di queste righe di codice comprende un numero maggiore di informazioni rispetto a quanto visto in Sez. 4.1.1, ed è il seguente:

Listing 4.25: Stampa a Video Relativa al Codice Precedente

```

Training terminated, stop criteria met: Current MaxAbsError is equals
  or lesser than the mimimum one (0.0480365<=0.05)
Final Training Result:
  MaE=0.013672 MaxAE=0.0480365 R=0.999386 S=0.0176099
Final Validaiton Result:
  MaE=0.0186312 MaxAE=0.0533141 R=0.996919 S=0.0240095

```

Come è possibile osservare, il codice espresso sopra è indipendente dal tipo di modello, di algoritmo di apprendimento e tipo di dataset usato. Usando la struttura polimorfica della libreria, la creazione del modello, lo svolgimento dell'addestramento e la raccolta e stampa dei risultati si adatterà a seconda del task, e del tipo di modello e algoritmo di apprendimento scelto, mediante il descrittore.

Tutti gli algoritmi di apprendimento implementati in *felix* sono sottoclassi concrete di *iLearningAlgorithm* e sono stati implementati, a partire delle loro formulazioni teoriche⁵, nelle seguenti classi:

Backpropagation descritto in Sez. 2.2.1.1;

Resilientpropagation descritto in Sez. 2.2.1.2;

Quickpropagation descritto in Sez. 2.2.1.3;

Pseudoinverse descritto in Sez. 2.2.2.1, utilizzabile solo per l'addestramento delle unità di uscita;

⁵Fa eccezione il caso della pseudoinversa, dove ci siamo appoggiati alla libreria *armadillo* per il calcolo effettivo della matrice.

CascadeCorrelation descritto in Sez. 2.3.1;

CascadeResidual descritto in Sez. 2.3.2;

La funzione astratta *start_training* possiede un'implementazione in *iLearningAlgorithm* che viene usata da tutti gli algoritmi di discesa del gradiente, la quale prevede di:

1. calcolare l'errore residuo per ogni pattern del training set;
2. aggiornare, usando l'errore residuo, il valore delle derivate parziali contenute in tutte le istanze *weight* delle unità, mediante la funzione *updateAllDerivateValues*;
3. modificare il valore dei pesi, ad epoca finita, invocando la funzione astratta *trainUnit* su tutte le unità del modello;
4. controllare il soddisfacimento di eventuali stop criteria e, in caso negativo, tornare al punto 1;

La funzione di modifica dei pesi invocata all'interno della funzione astratta *trainUnit* è praticamente l'unica che viene sovrascritta dalle classi che implementano i metodi del gradiente.

Ognuno di questi metodi differiscono solo dal modo in cui usano l'informazione sulla derivata parziale per aggiornare i pesi. Con questa organizzazione è stato possibile implementare un metodo del gradiente scrivendo unicamente la sua funzione di modifica dei pesi, ereditando tutto il resto da *iLearningAlgorithm*.

La funzione *start_training* viene, invece, sovrascritta per i metodi costruttivi, e lancia un'eccezione qualora la si invochi su un'istanza di *Pseudoinverse*.

Entrambi i metodi costruttivi implementati fanno uso di una struttura ausiliaria *poolCandidate* che permette l'addestramento di un'unità candidata in modo parallelo ed indipendente dalle altre. Per farlo, si è ricorsi alla programmazione concorrente mediante l'uso di *thread* che effettuano l'addestramento concorrente di tutte le unità candidate, sfruttando il fatto che durante l'addestramento dell'unità candidata tutti gli altri pesi del modello sono congelati, caratteristica comune ad entrambe le varianti di algoritmi costruttivi viste.

Oltre a quanto detto, gli algoritmi di apprendimento hanno la possibilità di conservare all'interno dell'istanza *results* un *log* analitico dell'andamento dell'addestramento, in modo che possa essere analizzato successivamente (vedere Sez. 4.3.2). Questa opzione prevede, specialmente per training molto lunghi, un incremento notevoli

le delle dimensioni del pacchetto dei risultati finali (vedere Sez. 4.1.5). Per questo motivo è possibile escludere le funzionalità di logging settando a *False* l'attributo *saveExtendedLogging* del tag *Learning* del descrittore.

4.1.4.1 Stop Criteria

I vari stop criteria usati negli algoritmi di apprendimento sono creati nel costruttore di *iLearningAlgorithm* usando l'abstract factory *StopCriteriaFactory*.

Tra le classi che implementano i diversi stop criteria, abbiamo:

SC_maxIterationReached che risulta soddisfatto al raggiungimento di una certa epoca massima, o di una certa unità nascosta, nel caso di algoritmi costruttivi. A seconda dei parametri passati, la soglia limite può essere incrementata dinamicamente, permettendo di eseguire un'incremental strategy;

SC_minimumMaximumAbsError che risulta soddisfatto al raggiungimento di un *MaxAbsE* minimo fissato;

SC_maximumTrainingResult che risulta soddisfatto al raggiungimento di un massimo valore dipendente dal task, come l'accuracy nel caso di classificazione;

SC_resultsStale che risulta soddisfatto al raggiungimento di uno stallo nel risultato;

SC_resultsMonotony che termina l'addestramento quando il risultato non mantiene un andamento monotono;

SC_minIterations questo controllo inibisce gli altri stop criteria, e permette di non fermare l'addestramento fino a che un numero minimo di epoche non sono passate, o un certo numero di unità nascoste non sono state create; Questo, pur non essendo un vero e proprio stop criteria, è stato implementato come tale per facilità di integrazione;

Ognuna di queste classi è una sottoclasse concreta della classe astratta *Stop Criteria*, ed implementa la funzione astratta pura *isMet*, invocata dall'algoritmo di apprendimento su ogni stop criteria che possiede. Di fatto, l'algoritmo di apprendimento non istanzia esplicitamente gli stop criteria, ma usando un'oggetto *StopCriteriaFactory* riempie un vettore di stop criteria posseduto localmente:

Listing 4.26: Vettore di Stop Criteria contenuto in *iLearningAlgorithm*

```
vector<StopCriteria*> stopCriteriaSet;
```

i cui elementi vengono controllati invocando *isMet* alla fine di ogni epoca.

4.1.5 Gestione dei Risultati

La classe *Result* implementa la gestione dei risultati di un processo di training.

Richiamando la funzione *gatherEpochResults* alla fine di un'epoca, all'interno della funzione *start_training* dell'algoritmo di apprendimento, la classe *Results* calcola la performance del modello nel training e validation set, e la memorizza usando apposite strutture *result_package*, le quali memorizzano le performance di un modello su una singola epoca.

La struttura *result_package* è estesa opportunamente per poter gestire in modo automatico tipi di task diversi, sovrascrivendo opportunamente funzioni di utilità come *str* e *getMainResult* in modo che restituiscano valori specifici del tipo di task.

I risultati di ogni epoca (o per ogni candidate unit inserita nel modello) vengono conservate in appositi vettori:

Listing 4.27: Vettori dei Risultati di Training e Validation

```
vector<result_package*> trainingResults;
vector<result_package*> validationResults;
```

con funzioni e strutture ausiliare per facilitare l'accesso agli ultimi risultati, come:

Listing 4.28: Strutture Ausiliare

```
result_package* lastTrainingResult;
result_package* lastValidationResult;
```

La gestione di addestramenti con molti trials o con CV, i quali generano molte istanze di *Result*, vengono gestite mediante opportune estensioni della classe base *ResultsTrialsCollector* e *ResultsCVCollector*. Queste classi raccolgono i vari oggetti *Result* prodotti dai training multipli, ad esempio i training paralleli effettuati in una CV, e li raccolgono conservando informazioni ausiliarie, come la media delle performance e il modello migliore generato. Queste classi sovrascrivono opportunamente le funzioni ausiliarie di salvataggio della classe *Result* inserendo nel pacchetto le informazioni riguardanti l'andamento dei vari training.

Il salvataggio viene eseguito mediante diverse funzioni, tutte invocate all'interno della funzione principale *save* di *Result*, che permette di creare e salvare il pacchetto dei risultati all'interno di una cartella avente per nome l'ultimo risultato ottenuto in validation, inserendola sotto il path indicato nel descrittore. In questo modo è possibile avviare molti training con diversi descrittori, e andare a controllare i risultati migliori semplicemente ordinando le cartelle generate da *Result*.

4.1.5.1 Pacchetto dei Risultati

I risultati prodotti dal training vengono salvati in un pacchetto che possiede il seguente formato:

Listing 4.29: Formato del Pacchetto dei Risultati

```
AAAA-MM-DD_hh-mm-ss_name.res
```

dove la prima parte rappresenta l'informazione temporale relativa al completamento dell'addestramento:

AAAA-MM-DD rappresenta l'anno a quattro cifre, seguito dal mese e dal giorno;

hh-mm-ss rappresenta l'ora, il minuto ed il secondo di terminazione dell'addestramento;

mentre *name* è il nome del descrittore espresso dall'attributo *name* del tag *TaskDescriptor*.

Questa rappresentazione, unita all'accortezza di inserire il pacchetto in una cartella nominata secondo le performance finali del modello, permette di localizzare rapidamente un singolo risultato, anche dopo diversi giorni di training su descrittori diversi.

Il pacchetto è in realtà costituito una serie di files compressi nel formato *tar*. L'estensione *.tar* viene rinominata in *.res* per poter facilitare l'associazione del pacchetto a programmi di analisi post-training (vedere Sez. 4.3.2), ma è sempre possibile decomprimerlo qualora si volesse accedere ai singoli files contenuti.

Tutti i files inclusi nel pacchetto hanno il nome del descrittore, come esempio useremo il termine *name*, come prefisso⁶, ed un identificativo univoco come suffisso. Tra questi troviamo:

⁶Avere il nome del descrittore come prefisso permette di limitare i danni qualora si decomprima accidentalmente una serie di pacchetti nella stessa directory.

- Il modello addestrato *name_model.xml*, salvato usando le metodologie descritte in Sez. 4.1.3.2;
- Il descrittore usato per l'addestramento *name_descriptor.cfg*, al quale viene aggiunto l'attributo *seed* nel tag *TaskDescriptor*, in modo da poter rigenerare lo stesso modello ex novo, usando lo stesso descrittore;
- Eventuali file di log, in formato *.txt*, presenti nel caso sia stata abilitata la funzionalità nel descrittore;
- Un file latex *name_report.tex* usato per generare un report dettagliato dal tool *felix-reader*;
- La risposta del modello ad ogni pattern del training set in un file *name_completeOutput.csv*, usata dal predittore (vedere Sez. 5) per valutare il comportamento del modello in presenza di molecole simili a quella da predire;
- Una serie di file dal suffisso variabile, con estensione *.dat*, che contengono i risultati del modello al variare delle epoche, o delle unità nascoste. Il suffisso è variabile a seconda se si tratta di una separazione Hold Out o CV, e il numero varia a seconda del numero di trials eseguiti;
- In caso di Double CV, saranno inoltre presenti ulteriori dati in formato *.csv* che presentano i risultati delle molteplici varianti, nel loop interno ed esterno;

I file degli addestramenti con CV e di quelli con un numero di trial superiore ad uno, vengono generati sfruttando le funzioni sovrascritte nelle sottoclassi di *Result*.

4.1.6 Descrittore

La selezione del modello empirica prevede di effettuare molti test con diversi iperparametri relativi al modello e all'algoritmo di apprendimento. Il numero e la complessità degli iperparametri rende difficile la gestione mediante l'uso di *flags* da impostare, o da far passare come parametri a riga di comando. Per questo motivo si è scelto di utilizzare un approccio che prevedesse l'uso di un file di configurazione dove fossero contenute tutte le informazioni necessarie all'addestramento.

Questo file, chiamato *descrittore*, è codificato in XML e contiene le informazioni principali del:

Task contenente le informazioni operative relative all'esecuzione dell'addestramento;

Data dove sono contenuti i path del dataset e della cartella in cui andranno salvati i risultati. In questa parte verranno inoltre definiti il tipo di dataset usato (flat o grafo) e il tipo di separazione;

Model dove viene espresso il tipo di modello e i suoi iperparametri. Il numero e tipo di iperparametri sono informazioni modello-specifiche, come il numero di unità nascoste e di layers per il Multilayer Perceptron, o il tipo di funzione di attivazione per le unità;

Learning dove vengono espressi tutti i parametri dell'algoritmo di apprendimento, come *learning rate* o *weight decay*, oltre che gli stop criteria usati;

Ad esempio, nella parte relativa al Task una delle informazioni che è possibile fornire è il numero di *trial* il cui valore, se maggiore di 1, permette di addestrare diverse istanze con pesi iniziali diversi, e conservare la migliore. In questo caso l'andamento di tutte le istanze addestrate è conservato nel file del risultato, e può essere analizzato separatamente mediante il tool *felix-reader*.

Ogni descrittore deve contenere queste parti, le cui informazioni verranno passate ai rispettivi componenti della libreria *felix* (vedere Sez. 4.1.1). La lettura, ed eventuale salvataggio, del descrittore è gestita mediante la classe *TaskDescriptor*, la quale contiene alcune funzioni di utilità e quattro istanze pubbliche di *DescriptorParameter* che contengono tutte le informazioni relative al training:

Listing 4.30: Strutture Relative alle Varie componenti

```
DescriptorParameter* taskParameters;  
DescriptorParameter* dataParameters;  
DescriptorParameter* modelParameters;  
DescriptorParameter* learningAlgorithmParameters;
```

La classe *DescriptorParameter* non è altro che una rappresentazione logica degli iperparametri, e segue da vicino la struttura ad albero del linguaggio XML. Definita ricorsivamente, la classe *DescriptorParameter* contiene al suo interno il nome (corrispondente al tag name XML), una lista di attributi, dei riferimenti ai parametri figli e una serie di funzionalità utili alla navigazione e al reperimento di un sottoparametro specifico. Gli attributi vengono gestiti mediante una classe *DescriptorAttribute* che, oltre alla coppia

di attributi *name* e *value*, presenta diverse sottoclassi specifiche, a seconda del tipo di attributo, come *DescriptorIntAttribute* e *DescriptorBoolAttribute*.

Dato che la libreria non ha controllo su quanto presente nel file del descrittore, queste sottoclassi permettono di effettuare una validazione del descrittore passato.

Ogni componente principale di *felix*, come i modelli, i vari algoritmo di apprendimento e dataset, riceve nel costruttore, dalla sua abstract factory, un'istanza di *DescriptorParameter* contenente tutti gli iperparametri necessari al componente per essere creato, come ad esempio il numero di unità e layers in un *MultiLayerPerceptron* o la learning rate di una classe *Backpropagation*.

Ognuno di questi componenti implementa una funzione statica *getParameters* che restituisce un'istanza di *DescriptorParameter*, contenente un *template* di tutti i parametri necessari alla classe per poter essere istanziata, e questo viene usato per effettuare una validazione dei parametri passati. L'idea di base è la seguente: una volta che si implementa un componente nuovo, si assume che l'istanza di *DescriptorParameter* passata contenga lo stesso numero e tipo di parametri presenti in quella restituita dalla funzione *getParameters* della stessa classe.

Durante l'utilizzo della libreria, il descrittore è la prima cosa che deve essere caricata, in quanto contenente tutte le informazioni necessarie ad istanziare le successive componenti. Una volta caricato il descrittore, verranno passati i relativi oggetti *DescriptorParameter*, contenenti i valori settati nel file, ai relativi componenti per mezzo del costruttore. Questo viene fatto esplicitamente, ad esempio quando si vuole istanziare un modello specifico, o per mezzo di un'abstract factory che, sempre seguendo l'esempio, selezionerà il modello corretto in base alle informazioni presenti nel descrittore.

Questo approccio permette di gestire tutti i parametri della libreria in un unico file lasciando, allo stesso tempo, totale libertà nella gestione dei parametri per i singoli componenti.

Un esempio completo di descrittore è presente in App. [C.3](#).

4.1.6.1 Descrittore Grid

Una variante del descrittore è rappresentata dal grid, che permette di generare facilmente una famiglia di descrittori, che rappresentano varianti di un singolo descrittore di partenza.

L'utilizzo principale del grid è quello di effettuare una Double CV specificando in modo semplice i parametri che devono essere messi a griglia, rispetto ad un certo descrittore, chiamato *descrittore originale*. Un descrittore grid si presenta come un file XML il cui tag principale porta il nome di *GridDescriptor*, e che possiede, all'interno dell'attributo *path* del tag *taskDescriptor*, un riferimento al file descrittore originale.

Oltre al tag *taskDescriptor*, presenta un tag *Data* che definisce il tipo di loop esterno da usare, con i suoi parametri, come ad esempio il numero di folds della CV esterna.

I successivi tag si limitano a specificare solamente quei parametri che devono essere variati, rispetto al descrittore originale. Per specificare un parametro oggetto della variazione, contenuto in un attributo del descrittore originario, il descrittore grid replicherà la struttura dei tag del descrittore originario, limitandosi a riportare i nomi dei tag (senza gli attributi) fino ad arrivare al tag che contiene l'attributo che si vuole modificare. A questo punto nel grid sarà inserito un tag figlio, avente come nome l'attributo da modificare, e come attributi le keywords *from*, *to* e *tick* che rappresentano la variazione da eseguire.

Ad esempio, se volessimo testare un descrittore variando la learning rate dell'algoritmo di apprendimento, magari testandola sull'intervallo $\eta \in [0.1, 0.5]$ con scatti di 0.1 è sufficiente inserire questo tag all'interno del descrittore grid:

Listing 4.31: Esempio di Variante Inserita nel Grid

```
<Learning>
<learningRate from="0.1" to="0.5" tick="0.1"/>
</Learning>
```

In questo modo, viene identificato nel descrittore originale l'attributo *learningRate* del seguente tag:

Listing 4.32: Attributo Individuato nel Descrittore

```
<Learning name="Backpropagation" learningRate="0.1" weightDecay="0.01"
saveExtendedLogging="True">
```

e il sistema provvederà a generare 5 istanze diverse di *TaskDescriptor*, ognuna rappresentante una variazione della learning rate dell'algoritmo di apprendimento.

Questo è gestito mediante la classe *GridDescriptor* che si occupa di caricare il file del descrittore grid, tipicamente avente estensione *.grid*, e di generare le istanze di *TaskDescriptor* corrispondenti.

4.1.7 Estendibilità

La libreria *felix* permette di estendere i suoi componenti principali mediante la creazione di ulteriori modelli, classi di dataset, algoritmi di apprendimento o stop criteria, integrandoli con le funzionalità esistenti.

Qualora volessimo creare un nuovo algoritmo di apprendimento, ad esempio implementando la classe *MyNewAlgorithm*, occorre definire la classe estendendo *iLearningAlgorithm*:

Listing 4.33: Definizione della classe

```
class MyNewAlgorithm : public iLearningAlgorithm
{
public:
    MyNewAlgorithm (DescriptorParameter*,
                    iDataset*, iDataset*,
                    iNeuralNet*, ostream* log=NULL) throw(runtime_error);

    MyNewAlgorithm (const MyNewAlgorithm &);

    static DescriptorParameter getParameters();

protected:
    float par1, par2;
};
```

dove il metodo *getParameters* deve restituire l'oggetto *DescriptorParameter* contenente il template di tutti i parametri necessari all'algoritmo, e gli attributi *par1* e *par2* sono due parametri che l'algoritmo usa durante il suo funzionamento⁷.

La funzione *getParameters* costruisce l'oggetto *DescriptorParameter* nel seguente modo:

Listing 4.34: Implementazione di *getParameters*

```
DescriptorParameter MyNewAlgorithm::getParameters()
{
    DescriptorParameter par=iLearningAlgorithm::getParameters();
    par.setSelfExistingAttributeValue("name", "MyNewAlgorithm");
}
```

⁷Parametri di questo tipo possono rappresentare, ad esempio, la learning rate e il weight decay nell'algoritmo *BackPropagation*.

```

    par <<DescriptorFloatAttribute("firstParameter", "0.4", 0.0)
        <<DescriptorFloatAttribute("secondParameter", "0.8", 0.0);

    return par;
}

```

In questo modo si impone un valore di default ai due parametri, 0.4 e 0.8 rispettivamente, e un valore minimo per entrambi a 0.

Il *DescriptorParameter* così generato corrisponderebbe al seguente tag nel descrittore:

Listing 4.35: Tag corrispondente nel descrittore

```

<Learning name="MyNewAlgorithm" firstParameter="0.4" secondParameter="
    0.8" />

```

Le firme di entrambi i costruttori rispecchiano quelle dei costruttori della classe *iLearningAlgorithm*, difatti la loro implementazione richiama il costruttore della superclasse, sia nel caso del copy constructor:

Listing 4.36: Implementazione copy constructor

```

MyNewAlgorithm::MyNewAlgorithm(const MyNewAlgorithm& mna)
    : iLearningAlgorithm(mna),
      par1(mna.par1),
      par2(mna.par2) { }

```

sia nel caso del primo costruttore presentato:

Listing 4.37: Implementazione del primo costruttore

```

MyNewAlgorithm::MyNewAlgorithm( DescriptorParameter* par,
    iDataset* trSet,
    iDataset* valSet,
    iNeuralNet* m,
    ostream* logg) throw(runtime_error)
    : iLearningAlgorithm(par, trSet, valSet, m, logg)
{
    par1 = str2number<double>(par->getSelfAttributeValue("firstParameter"));
    par2 = str2number<double>(par->getSelfAttributeValue("secondParameter"));
}

```

Da notare come, all'interno del costruttore, il valore dei parametri sia reperibile direttamente dal riferimento di *DescriptorParameter* passato⁸.

Con questa implementazione la classe ha pieno accesso a tutte le funzioni e strutture dati principali della classe *iLearningAlgorithm*, tra cui l'istanza del modello, accessibile mediante l'attributo *model* e quella del risultato, accessibile mediante l'attributo *result*.

L'implementazione effettiva dell'algoritmo vero e proprio avviene sovrascrivendo una tra le seguenti funzioni virtuali presenti in *iLearningAlgorithm*:

trainJustWeight che prende in ingresso un riferimento ad un oggetto di tipo *weight* e ne modifica il valore. Ad esempio, tutti gli algoritmi di discesa del gradiente implementati sovrascrivono unicamente questa funzione, ed utilizzano le informazioni contenute nella struttura *weight* per aggiornare il valore del peso;

trainUnit che prende in ingresso un riferimento ad un oggetto di tipo *iNeuralUnit* e ne aggiorna i pesi. Questa funzione offre un maggior controllo sull'addestramento, permettendo di lavorare su un'intera unità invece che su un generico peso. Nella libreria *felix* la classe *Pseudoinverse* sovrascrive questa funzione in modo da scorrere facilmente tutti i valori in ingresso all'unità di uscita da addestrare, e costruire le matrici necessarie all'algoritmo.

start_training la funzione che viene chiamata dall'utilizzatore di *felix* per avviare l'addestramento può essere sovrascritta per raggiungere il massimo controllo consentito dalla libreria, sul training. Sovrascrivendo questa funzione, la classe ha un controllo completo sul processo di training, controllando i pattern che vengono passati al modello e i valori che vengono salvati all'interno della struttura *results*. Le classi che implementano i metodi costruttivi in *felix* sovrascrivono proprio questa funzione per poter effettuare i sotto-training locali e gestire l'architettura della rete a runtime.

Questi metodi sono stati elencati seguendo un ordine crescente di controllo che viene fornito all'utente che implementa il nuovo algoritmo di apprendimento, e questa suddivisione consente di ridurre il codice necessario ad implementare un nuovo algoritmo.

⁸La funzione *str2number* fa parte di una serie di funzioni di utilità implementate in *felix* e non illustrate precedentemente.

Per utilizzare la nuova classe è sufficiente sostituire la creazione dell'algoritmo di apprendimento, a mezzo di abstract factory (ad esempio nel codice 4.7) con la seguente parte:

Listing 4.38: Creazione del nuovo algoritmo

```
learning = new MyNewAlgorithm(  
    descriptor->learningAlgorithmParameters,  
    training_set, validation_set,  
    model);
```

Qualora si voglia integrare completamente l'algoritmo nella libreria *felix*, di fatto estendendola, è sufficiente modificare il codice di *LearningAlgorithmsFactory*, inserendo la nuova classe.

Operazioni analoghe possono essere eseguite per gli altri componenti, permettendo di estendere modelli e nuove classi di dati.

4.2 Libreria *qfelix*

Progettata come wrapper della libreria *felix*, la libreria *qfelix* ne estende alcune parti per facilitare l'uso di alcune sue funzionalità in un ambiente che utilizza il supporto grafico della libreria *QT*.

Oltre a fornire alcune funzioni *inline* generiche di utilità, la libreria implementa le seguenti classi:

QTrainingPlot che consente la visualizzazione dell'andamento di un training di un certo modello e la classe;

QTaskDescriptorViewer che permette di visualizzare graficamente il contenuto di un descrittore;

QModelText che permette di visualizzare i dettagli di un modello;

Estendendo il widget QT *QwtPlot*, la classe *QTrainingPlot* prende un'istanza di *Result* nel costruttore, e costruisce le curve di training visualizzate nel widget.

Alternativamente, è possibile costruire le curve caricando i dati da un file, mediante la funzione *setFromFile*, o aggiungendoli uno alla volta mediante la funzione *addData*. Il numero di curve generate dipende dal tipo di separazione usata in fase di training: in caso di Hold Out il plot presenterà due curve, mentre in caso di CV ci saranno un

numero di curve pari al numero di fold usate, con opportuna legenda presente al bordo del grafo.

La classe *QTaskDescriptorViewer* implementa direttamente *QWidget*, ovvero la classe più generica della libreria QT, e passando un descrittore alla funzione *updateSettings* costruisce una rappresentazione grafica del descrittore organizzando i componenti del dataset, del modello e dell'algoritmo di apprendimento in tre tabs diverse. Per ognuna di queste componenti vengono mostrati i parametri sottoforma di un albero *QTreeWidget* non modificabile.

In modo simile è stata implementata la classe *QModelText* che permette di mostrare il valore dei pesi, e le connessioni tra le unità di un modello, inserendole in un *QTreeWidget*.

4.3 Strumenti di Addestramento e Analisi dei Modelli

In questa sezione descriveremo alcuni strumenti sviluppati che usano le librerie *felix* e *qfelix* per effettuare l'addestramento dei modelli, e la loro successiva analisi.

Il processo di selezione del modello può richiedere molto tempo, sia da parte della macchina che deve effettuare materialmente l'addestramento, sia da parte dello sviluppatore, che deve confrontare i risultati e cercare la migliore combinazione di iperparametri possibile per ottenere le migliori performance.

I seguenti strumenti sono stati progettati proprio nell'ottica di semplificare entrambi questi aspetti.

4.3.1 cougar

Questo tool prende in ingresso il path ad un descrittore ed effettua l'addestramento specificato al suo interno, salvando il risultato ad addestramento terminato. La flessibilità della libreria *felix* permette di limitare l'interazione dell'utente che usa *cougar* alla scrittura del descrittore, il quale include tutti i dettagli dell'addestramento e del salvataggio dei risultati.

Il tool *cougar* sfrutta direttamente tutte le funzionalità presenti nella libreria *felix* per effettuare l'addestramento di un modello, creando un oggetto *TaskDescriptor* ed usando i *DescriptorParameter* ivi contenuti per:

- caricare il dataset, eseguendone la separazione opportuna;
- creare l'istanza corretta del modello, usando la relativa abstract factory;
- passare l'istanza del modello e del dataset all'abstract factory dell'algoritmo di apprendimento, per generare un'istanza dell'algoritmo;
- eseguire l'addestramento invocando *start_training* sull'istanza dell'algoritmo;
- salvare i risultati secondo quando disposto dal descrittore;

Nel caso in cui il descrittore passato sia un grid, al posto di un'istanza di *TaskDescriptor*, verrà generata una della classe *GridDescriptor*, la quale verrà usata per effettuare molteplici training, a seconda del numero di varianti impostate nel grid descriptor.

Oltre al suddetto caso, è necessario eseguire molteplici training anche nel caso di separazione CV o di training a mezzo di diversi trial. In molti di questi casi *cougar* effettua i training in parallelo, usando diversi thread per effettuare i training in modo indipendente.

Durante l'esecuzione, il tool si occupa di fornire a schermo un sintetico riassunto dell'andamento dell'addestramento, con informazioni riguardanti:

- le performance correnti;
- il tempo impiegato:
 - per l'addestramento locale, come ad esempio per l'addestramento su una singola fold nel CV;
 - per l'addestramento complessivo;
- una stima del tempo di termine dell'addestramento, calcolato sulla media dei vari tempi locali;
- la memoria correntemente occupata dal processo⁹
- il risultato finale, qualora l'addestramento sia terminato;
- il percorso completo di salvataggio del pacchetto dei risultati;

⁹L'informazione sulla memoria che occupa un processo di training, sebbene non vitale per il processo in se, è stata utile per localizzare *memory leak* in fase di implementazione, specialmente in presenza di CV e grid dove molte istanze venivano create ed eliminate.

Questo che segue è un esempio di output con il processo di training di un grid in corso:

Listing 4.39: Esempio di Output di un Processo di Training

```

...
Hidden unit 19 trained: Training: MaE=0.0138446 MaxAE=0.0498693 R
    =0.999359 S=0.0178256 Validation: MaE=0.0219879 MaxAE=0.0477873 R
    =0.996515 S=0.0254891
Training terminated: Current MaxAbsError is equals or lesser than the
    mimimum one (0.0498693<=0.05)
Train fold 10 ended, time required 0:01:49 full ETA: 0:18:16

CV averaged Hidden Units: 21 time required: 0:01:49 results (Mem:
    117.67 Mb):
    Training: MaE=0.0130822 MaxAE=0.0479075 R=0.999375 S=0.0170038
    Validation: MaE=0.0176314 MaxAE=0.0550826 R=0.99801 S=0.0235522

Training 9-th terminated with 0.0176314 MaE on test fold, 25%
    completed (ETA 6:26:28) (Mem: 117.67 Mb)
It's worse than the best one, so this result will be discarded.
Training 11-th started, with descriptor maxAbsErr05QPwd001_WD=5e-05_TK
    =30_LA=0.05

```

Questo ci suggerisce che il processo di training ha finito l'addestramento di una CV, che il risultato finale è peggiore di uno precedente (quindi il modello non viene conservato), e che *cougar* ha appena avviato l'addestramento usando il descrittore successivo, generato da *GridDescriptor*.

Alla fine dell'addestramento, viene mostrato un riassunto con la media delle performance e il path dove è stato salvato il pacchetto del risultato:

Listing 4.40: Esempio di Risultato Finale

```

...
Results:
    Training: MaE=0.013672 MaxAE=0.0480365 R=0.999386 S=0.0176099
    Validation: MaE=0.0186312 MaxAE=0.0533141 R=0.996919 S=0.0240095
Training ended, completion time 10:23:19
Saving the result...
Averaged results saved in: /home/barontil/tesi/results/alkanes
    /0.0188522/2014-10-15_13-00-54_maxAbsErr05QPwd001_WD=0_TK=10_LA=0.
    res

```

4.3.2 felix-reader

Come descritto nella sezione 4.1.5.1, il risultato di un addestramento è conservato in un pacchetto *.res* contenente il modello, il descrittore usato per generarlo, e una serie di altre informazioni utili.

Queste informazioni, come i risultati parziali, possono essere indagate per individuare, ad esempio, una scelta di iperparametri che ottengano istanze di modelli con performance migliori.

Dato che l'analisi di molti dati post-training per diversi addestramenti, può essere un'operazione complessa, questo tool è stato sviluppato con l'intento di semplificare queste operazioni di analisi.

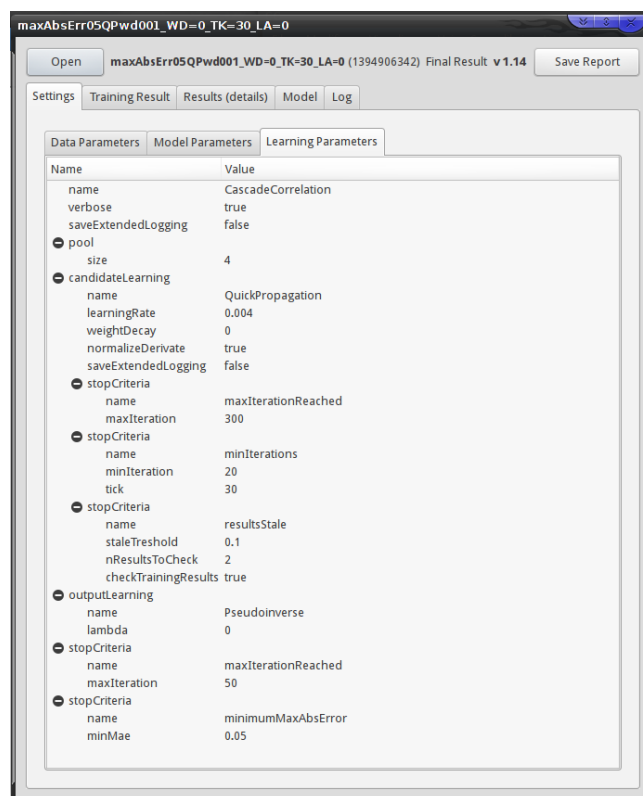


Figura 4.2: Gli iperparametri usati nel training vengono mostrati mediante un sistema di tabulazione.

Il tool *felix-reader* fa uso della libreria *qfelix* (vedere Sez. 4.2) per generare la maggior parte dei widget che compongono la finestra principale. La parte superiore presenta un pulsante *Open* che permette di caricare un pacchetto dei risultati, ed uno *Save Report* che permette di generare e salvare su file un report completo in formato

PDF. Sempre nella parte superiore sono presenti il nome del descrittore caricato e il seed (tra parentesi).

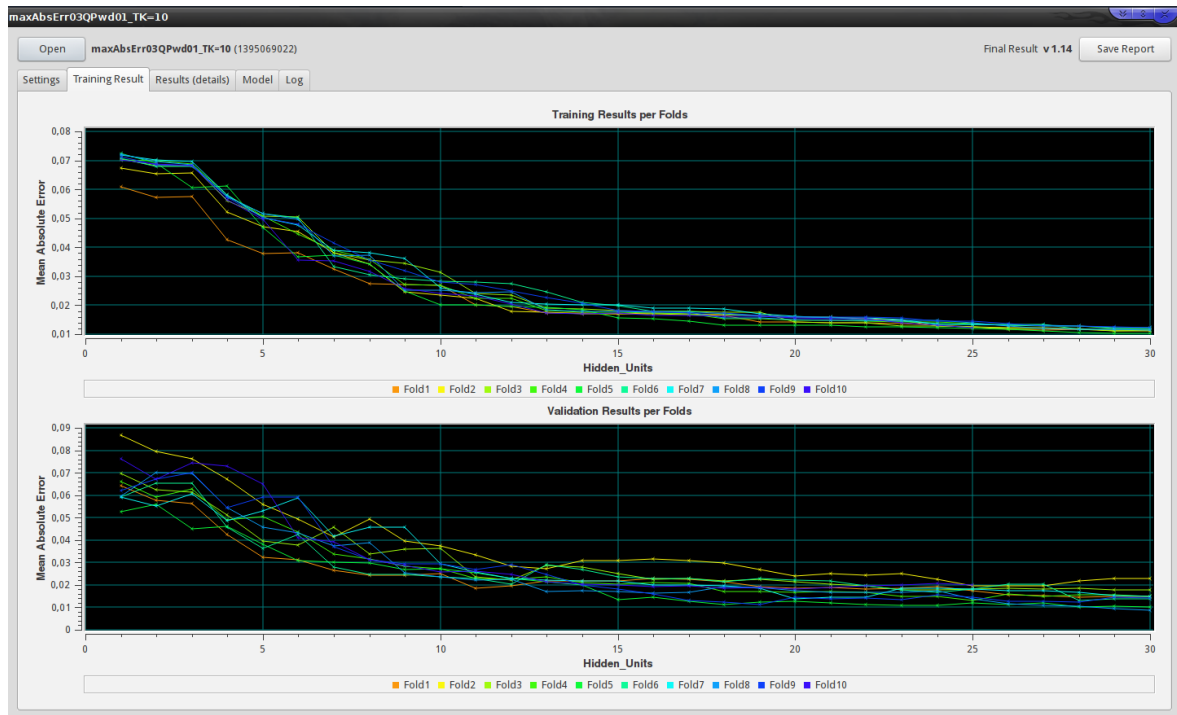


Figura 4.3: In questo esempio sono rappresentati gli andamenti di 10 training su altrettante folds di una separazione CV.

La parte centrale della finestra principale è occupata da un tabulatore che separa le informazioni riguardanti:

- i settings e gli iperparametri usati. Come è possibile osservare in Fig. 4.2, le varie parti del descrittore caricato vengono mostrate usando un'istanza di *QTaskDescriptorViewer*, e consentono di riassumere le condizioni sotto le quali l'addestramento è stato effettuato;
- l'andamento dell'addestramento. Visibile sia in formato grafico (vedere Fig. 4.3), che tabellare (vedere Fig. 4.4), è lo strumento di analisi principale offerto dal tool;
- il modello prodotto, visibile in Fig. 4.5a, permette di osservare l'architettura della rete generata e il valore dei pesi delle unità. Questo è utile per controllare alcune criticità come, ad esempio, un livello di saturazione elevato di alcune unità;
- l'eventuale log generato dall'algoritmo di apprendimento. Se presente, fornisce un dettaglio molto elevato su tutto il processo di training, come è visibile in Fig. 4.5b;

Hidden_Units	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6	Fold7	Fold8	Fold9	Fold10
1	0.0609182	0.0673127	0.0703649	0.0702961	0.0725296	0.0708198	0.0719133	0.07206	0.0712589	0.0701508
2	0.057294	0.0653182	0.0682471	0.0697441	0.0689684	0.0678904	0.0702768	0.0694623	0.0685017	0.0688354
3	0.0573735	0.0656288	0.0683273	0.0687965	0.060557	0.0678051	0.069525	0.0683795	0.0682562	0.0682355
4	0.0425495	0.0521268	0.0559734	0.0580077	0.061286	0.0575339	0.0579858	0.0573865	0.0571025	0.0562496
5	0.0377609	0.0471185	0.050847	0.0508513	0.0468736	0.0516701	0.0501614	0.0515732	0.0503163	0.0495336
6	0.0381276	0.0453449	0.0505612	0.0445141	0.0368365	0.0502119	0.0477155	0.0497084	0.0480749	0.0357213
7	0.032508	0.0382423	0.0379905	0.0389887	0.0372763	0.0334138	0.0388233	0.0371005	0.0413463	0.0354109
8	0.0274011	0.0341932	0.035489	0.0358742	0.0342431	0.030385	0.0380824	0.0372215	0.0355966	0.0317418
9	0.0270299	0.0246712	0.03441	0.0271858	0.0249657	0.0290813	0.0361932	0.02531	0.0318804	0.025798
10	0.0268179	0.0234218	0.0313581	0.0269072	0.0201713	0.0282494	0.0260067	0.0250853	0.0280765	0.0239436
11	0.0199994	0.0224856	0.0241184	0.0225072	0.0201008	0.0278923	0.0238633	0.0244666	0.0271585	0.0230958
12	0.0196881	0.0179351	0.0235903	0.0223469	0.0195083	0.0275289	0.0209861	0.024628	0.0248629	0.0203824
13	0.0173147	0.0174688	0.0189251	0.0181387	0.0191772	0.0245015	0.0204835	0.0183725	0.0227744	0.0174123
14	0.0173191	0.0169374	0.0187256	0.0176674	0.0184349	0.0209263	0.0200646	0.017818	0.0206801	0.0171681

Figura 4.4: In una tab apposita è possibile consultare il valore preciso dei risultati intermedi dell'addestramento.

Il sistema di log permette anche di associare diversi colori all'informazione da memorizzare, in modo da rilevare più efficacemente parti critiche nell'addestramento. Ad esempio, nella figura Fig. 4.5b si nota come l'algoritmo CC usato in quell'addestramento marchi in rosso le volte che il gradiente ha ottenuto un valore minore rispetto all'epoca precedente (nell'addestramento dell'unità candidata). Questo ha permesso di rilevare efficacemente situazioni in cui, ad esempio, la learning rate dell'algoritmo di apprendimento per l'unità candidata è troppo alta. Le figure 4.3, 4.4 e 4.5b mostrano un esempio di training mediante k -Folds CV. In questo caso, sono presentati due grafici, uno relativo ai risultati in training, ed uno ai risultati in validation, e la finestra di log è opportunamente separata per mostrare il dettaglio dell'addestramento di ognuna delle k folds.

Nel caso di training con diversi trial, un opportuno *QComboBox* sarà presente per poter scorrere tra i vari trial.

Il report viene generato mediante l'uso del file *.tex* presente nel pacchetto dei risultati, contenente molte delle informazioni suddette, sottoforma di macro $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$. Queste informazioni vengono usate da un file *template.tex*, incluso nel tool, il quale, includendo il file *.tex* presente nel pacchetto, dispone le informazioni secondo un layout prestabilito. Quando è richiesta la generazione del report, il tool si occupa di compilare il file

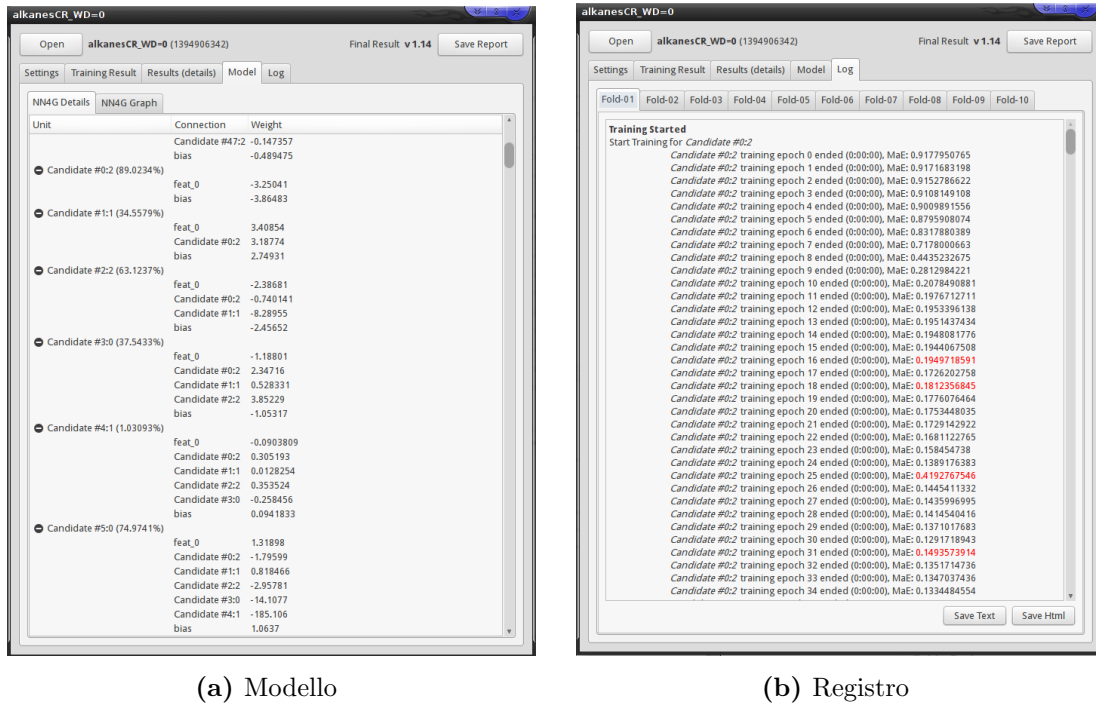
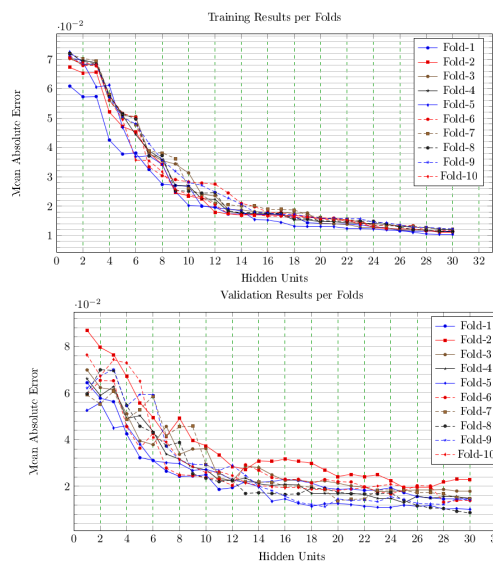


Figura 4.5: Esempio di modello, e di log, visualizzati da felix-reader.

template.tex in un file PDF che verrà salvato nella locazione scelta dall'utente. Un esempio di report generato è visibile in Fig. 4.6, mentre in App. F è riportato un estratto di un altro report generato da uno dei test di collaudo effettuati.

2 Training Results



Averaged Fold's Results				
	Mean Abs Error	Max Abs Error	R	S
Training	0.0116073	0.0458314	0.999515	0.0151146
Validation	0.015182	0.0464052	0.998609	0.0200898

- Averaged number of Hidden Units: 29

2.1 Fold-1 results

Averaged Final Results				
	Mean Abs Error	Max Abs Error	R	S
Training	0.0114339	0.047592	0.999418	0.0148185
Validation	0.0147097	0.0399738	0.997645	0.0205769

- Averaged number of Hidden Units: 1545626984

2.1.2 Fold-2 results

Final Results				
	Mean Abs Error	Max Abs Error	R	S
Training	0.0088295	0.0371343	0.999671	0.0119954
Validation	0.0216371	0.0944372	0.998649	0.0327994

Final stop criteria: Current MaxAbsError is equals or lesser than the minimum one (0.0371343 <= 0.04)

Here are the best/worst patterns on the Training dataset

Best				Worst			
Pattern	Target	Output	o - t	Pattern	Target	Output	o - t
1	-1.64	-1.64	3.67333e-06	130	1.7412	1.77833	0.0371343
88	1.65	1.64991	9.1128e-05	95	1.69	1.65333	0.0346749
24	1.1	1.09979	0.000211799	100	1.47	1.30414	0.034136
29	1.182	1.18168	0.000318066	135	1.6	1.36729	0.0327103
130	1.621	1.62144	0.00044691	128	1.6	1.63054	0.0303354

Here are the best/worst patterns on the Validation dataset

Best				Worst			
Pattern	Target	Output	o - t	Pattern	Target	Output	o - t
22	0.984	0.98549	0.00149016	2	-0.886	-0.980437	0.0944372
102	1.53	1.52822	0.00177756	142	1.5987	1.54709	0.0516107
52	1.24	1.23508	0.00491834	112	1.67	1.62515	0.0448497
32	1.156	1.16129	0.00529075	132	1.62	1.65109	0.0310942
42	1.33	1.32312	0.00688188	82	1.553	1.57818	0.0251836

2.1.3 Fold-3 results

Final Results				
	Mean Abs Error	Max Abs Error	R	S
Training	0.0111992	0.0397437	0.999536	0.0147136
Validation	0.0196098	0.0506348	0.998887	0.0244093

Final stop criteria: Current MaxAbsError is equals or lesser than the minimum one (0.0397437 <= 0.04)

Here are the best/worst patterns on the Training dataset

Best				Worst			
Pattern	Target	Output	o - t	Pattern	Target	Output	o - t
1	-1.64	-1.64	4.24174e-07	130	1.7412	1.78094	0.0397437
119	1.5704	1.57069	0.000292841	100	1.47	1.3082	0.0381977
106	1.528	1.52751	0.00048796	51	1.265	1.30067	0.0336677
46	1.415	1.41549	0.000489983	41	1.4027	1.36925	0.0334478

Figura 4.6: Estratto dal report in PDF generato dal tool.

Capitolo 5

Strumenti Sviluppati per la Valutazione di Molecole

Il predittore è stato sviluppato integrando il modello *NN4G* con l'intento di fornire uno strumento interattivo di supporto all'esperto che sia interessato a indagare su una proprietà fisico/chimica o biologica di una molecola. Questo strumento¹ è stato progettato integrando un sottoinsieme significativo delle funzionalità offerte dagli strumenti tossicologici disponibili (vedere Sez. 3.2), e fornendo alcune funzionalità sviluppate grazie all'approccio strutturale nella valutazione di molecole.

Molti predittori chimici, tra cui quelli visti in Sez. 3.2, basano la loro predizione principalmente sul valore di alcuni descrittori (vedere Sez. 4.1.6) o sulla presenza di determinati frammenti (vedere Sez. 3.1.1.1). Il predittore proposto, al contrario, usa principalmente le caratteristiche contestuali del modello *NN4G* (vedere Sez. 2.4.2) implementate dalla libreria *felix* (vedere Sez. 4.1) per effettuare predizioni di proprietà fisico/chimiche di una molecola, basandosi sulla sua struttura.

A differenza dei tool visti, questo approccio permette di sfruttare le caratteristiche contestuali del modello *NN4G* per identificare il contributo che hanno i singoli atomi di una molecola, nel generare il risultato del modello.

Questi *contributi* (illustrati dettagliatamente nella Sez. 5.3) sono uno dei principali aspetti innovativi del tool sviluppato, in quanto forniscono una maggiore trasparenza sulla generazione del risultato da parte del modello. Mostrati in modo sovrapposto alla rappresentazione grafica della molecola, i contributi permettono, all'esperto che fa

¹Il predittore conta circa 3000 righe di codice ed utilizza funzionalità implementate sia nella libreria *felix* che in *gfelix*.

uso del tool, di indagare sul contributo delle componenti strutturali della molecola in esame.

Oltre all'esito della predizione e alla distribuzione dei contributi, il sistema fornisce all'esperto informazioni sul dominio applicativo del modello in rapporto alla molecola da analizzare e permette di generare un report in formato PDF con tutte le informazioni riguardanti l'analisi della molecola. Un esempio di report generato è presente in App. E.

Il sistema permette l'utilizzo di un singolo modello *NN4G* per effettuare la predizione e fornire i contributi, oppure di un sistema binario, che utilizza due modelli, finalizzato a ridurre i contributi minori (vedere Sez. 5.4).

La sezione 5.3, verrà dedicata al modo in cui si è deciso di valutare i contributi dei singoli atomi rispetto alla predizione, e alle modalità di presentazione grafica. La sezione 5.1 descriveremo i files necessari al predittore per effettuare le predizioni, come il file del modello, e quelli opzionali come la lista delle structural alerts da individuare. Nella sezione 5.2, analizzeremo invece come vengono implementati gli aspetti sopra elencati, cercando di descrivere le interazioni delle strutture tool, di pari passo con il processo di interazione di un esperto del dominio con lo strumento. Nella sezione 5.4 daremo una breve descrizione delle classi principali, e delle loro interazioni.

5.1 File di Uso del Predittore

Il tool permette di effettuare predizioni sia di singole molecole, che di interi dataset, usando un modello *NN4G* implementato nella libreria *felix* (presentata in Sez. 4.1), addestrato mediante un tool opportuno, nel nostro caso *cougar* (presentata in Sez. 4.3.1).

A scopo dimostrativo, in questa sezione e nella Sez. 6.2, mostreremo il funzionamento del tool usando un modello addestrato con il tool *cougar* (presentata in Sez. 4.3.1) sul dataset Bursi, uno dei dataset usati come benchmark con target che mutagenico, descritto in Sez. 6.1.3. Di fatto, il tool può essere usato per effettuare predizioni su qualsiasi task, purchè sia presente un modello addestrato.

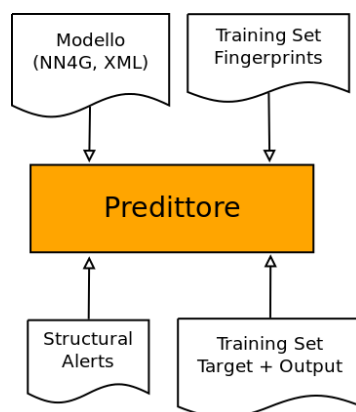


Figura 5.1: Lista dei files utilizzati dal predittore all'avvio

Il tool, per il suo funzionamento, necessita di alcuni files accessori (illustrati nel diagramma 5.1), descritti nei seguenti punti:

1. il modello, in formato XML, generato mediante la libreria *felix* (vedere Sez. 4.1), ed addestrato mediante il tool *cougar* (vedere Sez. 4.3.1) su un certo training set;
2. una serie di informazioni riguardo le structural alerts, opzionale e fornita dall'utente che usa il predittore;
3. la lista di elementi chimici presenti nel dataset su cui è stato addestrato il modello, identificati da un nome, dalla formula SMILES e dal fingerprint della molecola (vedere Sez. 3.1.1.1), il tutto separato da virgole, una molecola per linea. È opzionale e deve essere fornito da chi fornisce il modello addestrato;
4. la risposta del modello per ogni molecola del dataset usato per l'addestramento, assieme al rispettivo valore target;

Parte di questi files, come quelli descritti nei punti 1 e 4, sono specifici del modello addestrato, e possono essere recuperati dal pacchetto dei risultati prodotto da *felix* (vedere Sez. 4.1.5).

La lista di elementi del dataset di training, citata nel punto 3, deve essere ricavata del dataset usato per addestrare il modello, ed è usata dal sistema per estrapolare alcune informazioni utili a definire il dominio applicativo, come ad esempio la dimensione² di tutte le molecole presenti nel dataset.

²per dimensione si intende il numero di atomi presenti nella molecola.

I fingerprint delle molecole usate per addestrare il modello verranno usati per visualizzare le molecole più simili a quella fornita, assieme alla relativa risposta del modello fornita al punto 4. Qualora il file del fingerprint non sia presente, il sistema non potrà fornire questa informazione. Nel caso del modello usato come esempio in queste sezioni, i fingerprint delle molecole del dataset Bursi sono state estrapolate mediante uno script Python che sfrutta le funzionalità OpenBabel per estrapolare il fingerprint di una molecola.

Le informazioni riguardo le structural alerts, punto 2, possono essere fornite in un file di testo nel seguente formato:

$$\langle SAdoc \rangle ::= \langle SAinfo \rangle \langle SAdoc \rangle | \epsilon$$
$$\langle SAinfo \rangle ::= \langle name \rangle ':=' \langle formula \rangle '\n'$$

dove $\langle name \rangle$ è un qualsiasi nome identificativo della structural alerts, mentre $\langle formula \rangle$ è la formula della structural alerts in formato SMARTS. Queste informazioni non sono obbligatorie, e il file può essere lasciato vuoto, ma permettono di abilitare un utile strumento per l'esperto che voglia far evidenziare dal tool determinate structural alert o, più in generale, un certo insieme di sotto-strutture rilevanti per l'analisi di una molecola.

Ad esempio, durante gli esperimenti svolti nel corso di questo progetto, abbiamo inserito in quel file le informazioni riguardanti un insieme di di structural alerts mutageniche note (vedere Sez. 3.1.2), in modo da confrontarle rapidamente con la distribuzione dei contributi nella molecola (vedere Sez. 6.2).

La maggior parte di questi files devono essere modificati solo nel caso di modifica del modello usato dal predittore, mentre per la predizione effettiva occorre solo l'informazione riguardante la molecola da predire.

5.2 Utilizzo del Predittore

Il sistema prende in ingresso una singola molecola in formato SMILES o un intero dataset molecolare in formato SDF e, per ogni molecola da analizzare, esegue un processo di predizione effettuando i seguenti passi:

1. converte la molecola in una struttura intermedia *OBMol* di OpenBabel;
2. converte la molecola dalla struttura *OBMol* in una struttura *Graph* di *felix*;

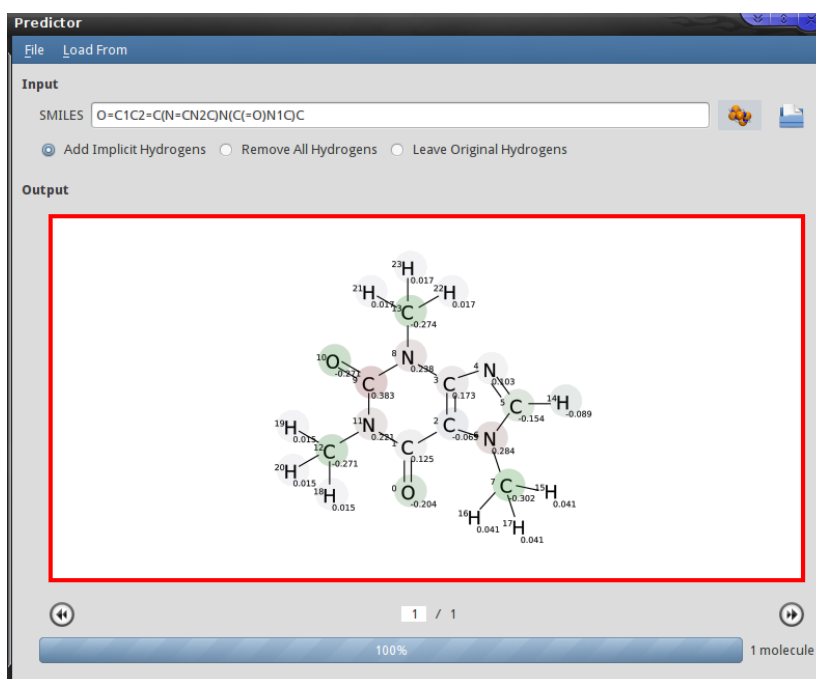


Figura 5.2: Schermata principale del predittore, in questo esempio si vede la conversione di una molecola da una formula SMILES, alla sua rappresentazione grafica

3. passa il grafo ottenuto al modello NN_4G caricato;
4. fornisce un responso positivo qualora la risposta del modello sia sopra il valore 0, negativa altrimenti;
5. mostra il responso finale assieme ad una rappresentazione grafica della molecola, con il valore dei contributi rappresentati graficamente sui singoli atomi, i quali sono marcati di rosso se forniscono un contributo positivo, e verde altrimenti;
6. fornisce all'utente finale, assieme al responso, il maggior numero di informazioni possibili, come quelle relative al dominio applicativo, finalizzate a valutare, da parte dell'esperto, la qualità della predizione.

Il passaggio intermedio di conversione della molecola in *OBMol* e successivamente in *Graph*, permette di astrarre dalla codifica originaria della stessa, senza fare distinzioni tra formato SDF e SMILES, e permettendo di estendere facilmente il tool, in futuro, per fornirli la possibilità di gestire ulteriori formati in ingresso.

Nel caso di predizione di molecole multiple, come il caricamento da SDF, il sistema prevede una disposizione grafica delle molecole a mosaico (come visibile in Fig. 5.3), ed un sistema a pagine per scorrere tra gruppi di 20 molecole alla volta. Questo sopperisce

al problema del caricamento di dataset grossi, dove il tempo computazionale necessario a convertire un grosso numero di molecole in forma grafica, potrebbe pregiudicare l'utilizzo.

Dal punto di vista grafico, la finestra principale, visibile in Fig. 5.2 è verticalmente suddivisa in 3 parti:

- La parte di *input*, la più alta, permette l'inserimento della formula SMILES e di avviare il processo di predizione premendo l'apposito pulsante. In questa parte è anche possibile effettuare il caricamento di un dataset SDF, il quale avvierà il processo di predizione per ogni molecola;
- La parte di *output*, al centro, è la zona dove verranno mostrate le molecole analizzate, sottoforma di anteprima;
- La parte terminale fornisce alcune informazioni sullo stato del sistema, come il numero di molecole caricate e il progresso nel caricamento della pagina successiva;

Le molecole mostrate in Fig. 5.2 e Fig. 5.3 rappresentano un'anteprima della predizione, dove è visibile la rappresentazione grafica della molecola, dei contributi, e del responso, rappresentato da un bordo verde se la predizione è negativa, rosso altrimenti. L'associazione del colore alla predizione è solo una convenzione dovuta al tipo di predizione che il tool deve effettuare, dove per una predizione positiva di tossicità si è deciso di far attribuire il colore rosso.

Una volta visualizzate le molecole, sia nel caso di predizione singola, che multi-molecolare, se dalla finestra principale si esegue un doppio click sull'anteprima della molecola, si apre una finestra con tutti i dettagli della predizione (visibile in Fig. 5.4, 5.5 e 5.6). La finestra dei dettagli è graficamente divisa in due parti:

- la metà superiore è la rappresentazione grafica della molecola, analoga a quella vista nella finestra principale, ma con la possibilità di effettuare zoom e panning. Per i dettagli del formato usato nella visualizzazione grafica della molecola vedere Sez. 5.3;
- la metà inferiore contiene diverse informazioni sulla predizione, tra cui le sezioni:

Data Info se la molecola proviene da un dataset, questa sezione contiene la lista di informazioni accessorie aggiunte dai creatori del dataset, per quella molecola, come ad esempio eventuali commenti;

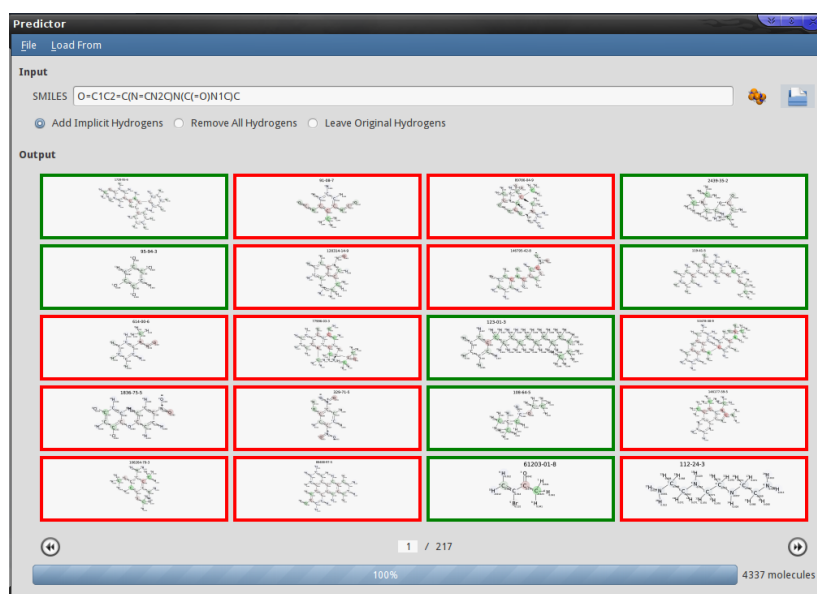


Figura 5.3: Visualizzazione multi-molecolare di un dataset caricato, con le molecole ordinate a mosaico e divise a gruppi di 20 per pagina.

Formula sezione che illustra una rappresentazione della funzione di uscita del modello, in funzione dei contributi dei singoli atomi della molecola analizzata;

Applicability Domain sezione che mostra una lista di informazioni riguardanti l'appartenenza della molecola da predire al dominio applicativo del modello usato;

Similar Mols che mostra una lista delle 5 molecole più simili a quella da predire, presenti nel dataset di training, con la relativa predizione del modello e valore target;

In alto è anche presente il responso in forma di etichetta (positiva/negativa) e in forma numerica, che rappresenta il valore effettivo di uscita del modello.

Entrambe queste parti possono essere ridimensionate usando la barra che le divide, ed è inoltre presente, nell'angolo in alto a destra, un pulsante *expand* che permette di espandere al massimo il grafico della molecola, per facilitare la ricerca di atomi specifici.

Sempre in alto a destra è presente un pulsante che permette di salvare un report della molecola mostrata (di cui è presente un esempio in App. E).

La rappresentazione grafica della molecola presenta le seguenti informazioni:

- l'indice dell'atomo relativo alla molecola, inserito in alto a sinistra rispetto all'etichetta dell'elemento. Questo indice è stabilito da *OpenBabel*, e viene usato per

qualsiasi informazione che fa riferimento diretto ai singoli atomi, come la formula esplicita, o le structural alerts rilevate (vedere 5.2.2);

- il contributo normalizzato dell'atomo, espresso in valore numerico in accordo al modello NN4G, inserito in basso a destra rispetto all'etichetta;
- un cerchio, o *marker*, colorato attorno all'elemento, di colore e trasparenza dipendenti unicamente dal valore del contributo dell'atomo stesso (per i dettagli, vedere 5.3);

Nelle prossime sezioni analizzeremo nel dettaglio le più importanti tra le informazioni visualizzate.

5.2.1 Formula

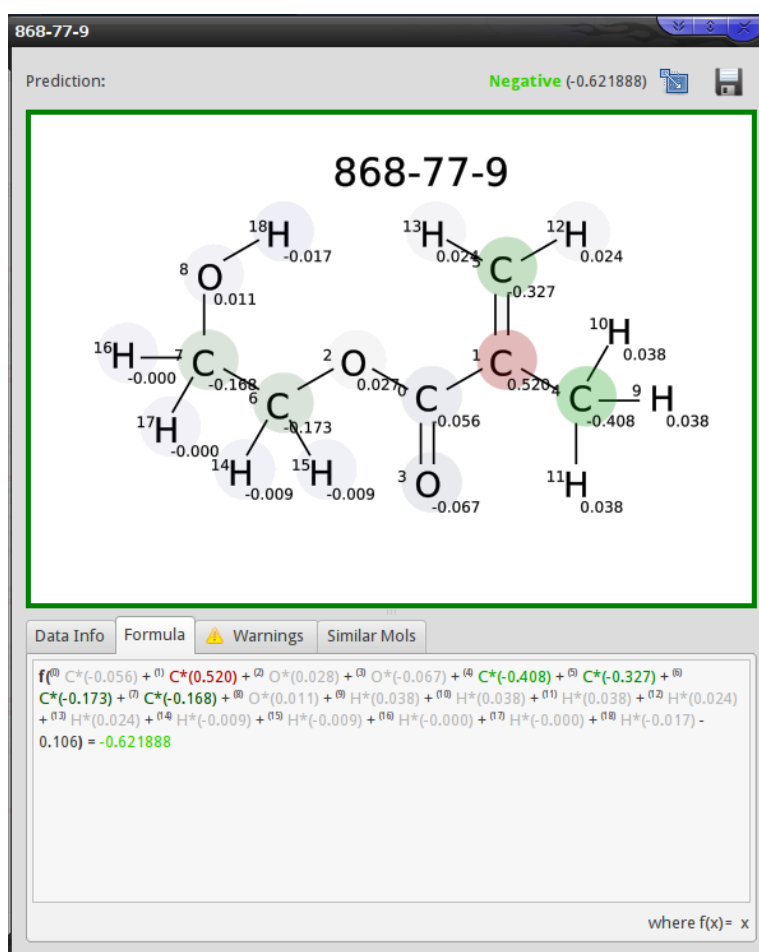


Figura 5.4: Formula estesa del modello.

La formula estesa, visibile in Fig. 5.4, permette all'esperto di osservare una formulazione della funzione di uscita del modello basata sui contributi dei singoli atomi di una molecola M . Data una molecola da predire, composta da n atomi, la formula viene presentata nella forma di:

$$f \left({}^{(0)}El_0 * (C_0(g_M)) + \dots + {}^{(i)}El_i * (C_i(g_M)) + \dots + {}^{(n)}El_n * (C_n(g_M)) + b \right) = Pr$$

dove El_i rappresenta l' i -esimo atomo della molecola, $C_i(g_M)$ il suo contributo rispetto al grafo g_M associato alla molecola M (vedere Sez. 5.3) e b il *bias* dell'unità di uscita del modello NN4G.

Il tipo di funzione f che permette di ottenere la predizione Pr è mostrata nell'angolo in basso a destra, e rappresenta la funzione di attivazione dell'unità di uscita del modello³. I colori di ogni coefficiente della formula rispecchiano quelli dei contributi, in modo analogo a quanto è visibile nella rappresentazione grafica.

5.2.2 Applicability Domain

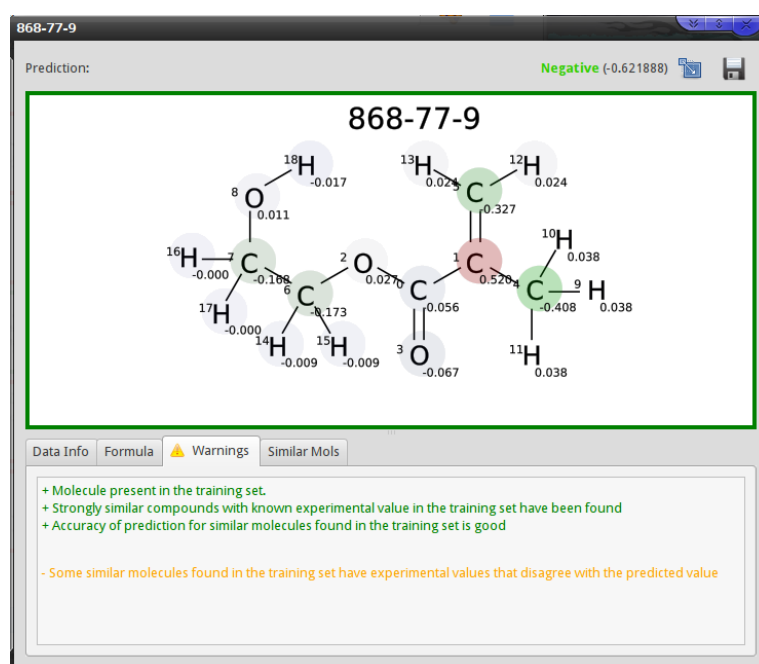


Figura 5.5: Dettagli sul dominio applicativo.

Il dominio applicativo è espresso mediante una serie di *warning* visibili nella tab *Applicability Domain*.

³nel caso dell'esempio in figura, l'uscita è lineare.

I warnings mostrati (vedere Fig. 5.5) rappresentano una serie di informazioni utili all'esperto per valutare l'attendibilità della predizione. Queste informazioni possono essere classificate nel seguente modo:

Similarity Warnings prendendo le più simili molecole presenti nel training set, rispetto a quella da analizzare (vedere Sez. 5.2.3), il predittore fornisce le seguenti informazioni:

- presenza della molecola predetta all'interno del training set⁴;
- presenza di molecole molto simili nel training set, valutando l'indice di Tanimoto T (vedere Sez. 3.1.1.1) della terza molecola tra le più simili e esprimendo una forte ($T > 0.8$), una moderata ($0.6 < T \leq 0.8$) o nessuna ($T \leq 0.6$) similarità della molecola da analizzare rispetto a quelle del training set;
- accuracy media delle tre molecole più simili, rispetto alla risposta del modello su quelle molecole, distinguendo tra un'accuracy media buona ($\mu_{Acc} > 0.9$), una non ottimale ($0.5 < \mu_{Acc} \leq 0.9$) o non adeguata ($\mu_{Acc} \leq 0.5$);
- errore medio delle tre molecole più simili, rispetto alla risposta del modello su quelle molecole, distinguendo tra valori sperimentali che concordano ($\mu_E > 0.9$), differiscono in parte ($0.5 < \mu_E \leq 0.9$) o totalmente ($\mu_E \leq 0.5$) con la risposta del modello;

Structural Warnings questi warnings vengono forniti analizzando alcune proprietà di tutte le molecole presenti nel training set, con lo scopo di evidenziare se, ad esempio, la molecola da analizzare ha un numero di atomi troppo alto rispetto a quelle presenti nel training set usato per addestrare il modello, o se presenta un elemento valutando:

- il numero di molecole nel training set che presentano lo stesso numero di atomi della molecola da analizzare, espresso in percentuale;
- la presenza, all'interno della molecola da analizzare, di elementi poco presenti ($< 10\%$ degli atomi complessivi) nel training set o, viceversa, l'assenza di elementi molto presenti ($> 70\%$ degli atomi complessivi) nel training set;

Structural Alerts Warnings vengono inoltre elencati tutti gli Structural Alerts trovati nella molecola, indicando il nome, e l'indice degli atomi coinvolti;

⁴per stabilire se due molecole sono effettivamente la stessa, si effettua una conversione di entrambe nel *canonical SMILES* e si confronta quel valore.

5.2.3 Similar Mols

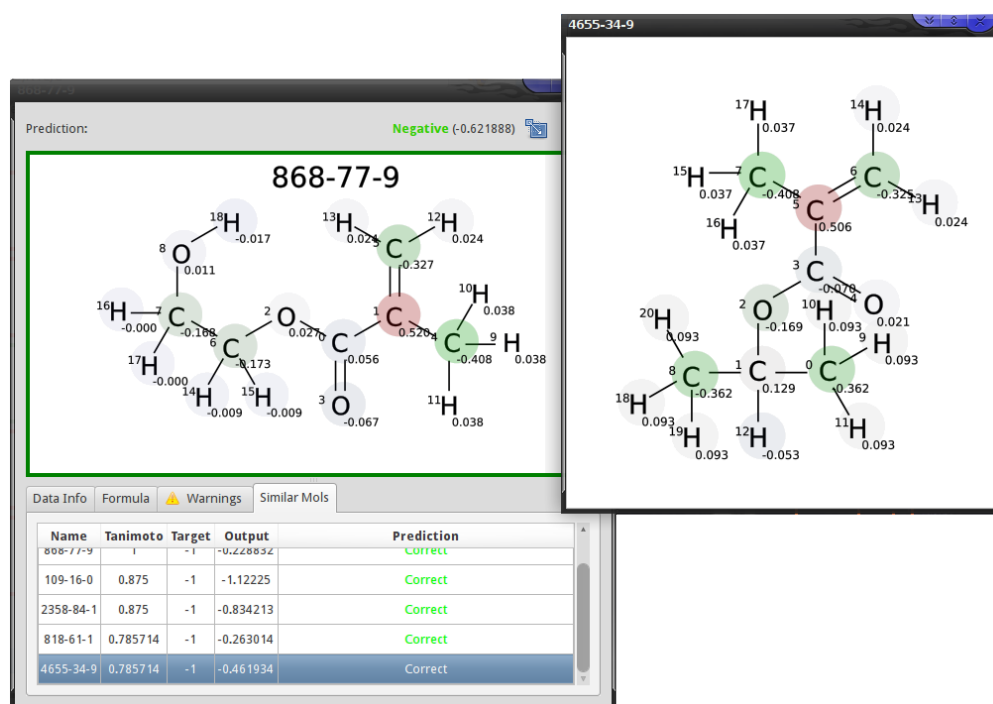


Figura 5.6: Lista di molecole simili.

Usando le informazioni sulle molecole del dataset di training, e l'indice di Tanimoto (vedere Sez. 3.1.1.1), vengono ricavate le 5 molecole più simili a quella da analizzare, e vengono mostrate (vedere Fig. 5.6) in una tabella contenente:

- il nome della molecola;
- l'indice di Tanimoto rispetto al fingerprint della molecola da analizzare;
- il valore target di quella molecola nel training set del modello;
- la risposta del modello a quella molecola;
- un'etichetta che riassume la correttezza della risposta del modello, rispetto al target, per quella molecola;

Eseguendo un doppio click su una di quelle righe, si apre una finestra contenente la rappresentazione grafica della molecola simile, con i vari contributi degli atomi.

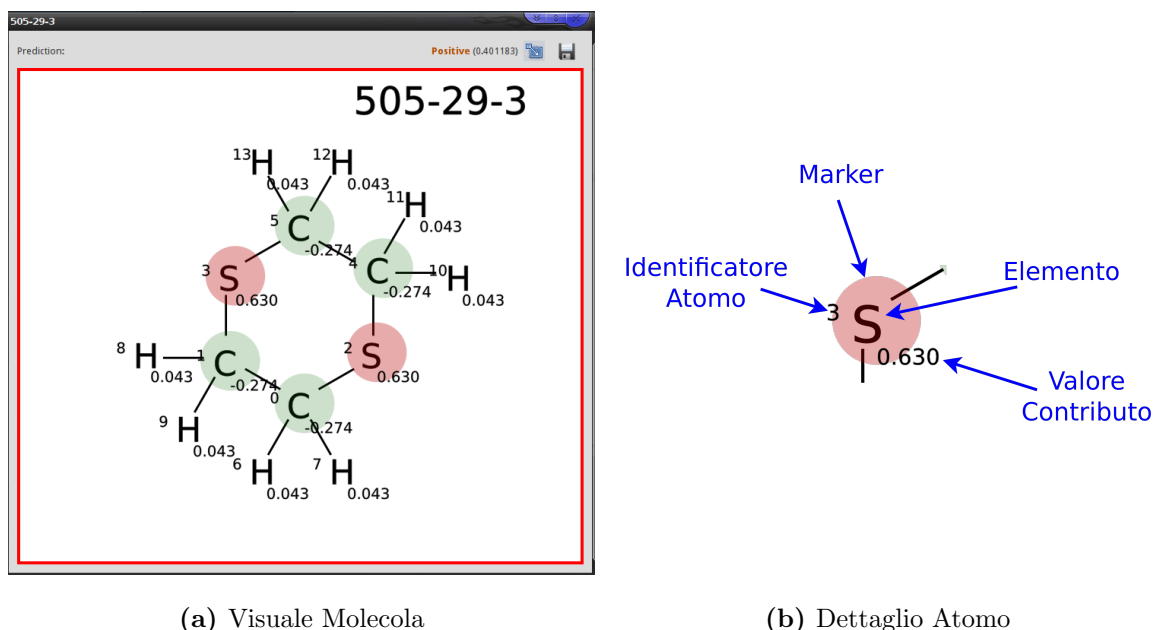


Figura 5.7: Dettaglio del formato usato per visualizzare le molecole.

5.3 Contributi Strutturali

Come accennato nell'introduzione del capitolo, la particolarità di questo tool è di poter evidenziare le sotto strutture che hanno contribuito maggiormente alla predizione, in modo da fornire un ulteriore supporto all'esperto che userà il sistema. Come è mostrato in Fig. 5.7, nella vista dettagliata di una molecola è possibile individuare le seguenti informazioni:

- Il nome identificativo della molecola;
- Un numero progressivo identificativo di ogni atomo;
- Il contributo numerico del singolo atomo;
- Un *marker* che indica il peso del contributo nel costituire la predizione;

Come descritto precedentemente (vedere Sez. 2.4) parlare del contributo degli atomi, o di gruppi di atomi, per generare la predizione di una proprietà di una molecola è equivalente, nel nostro approccio, a parlare del contributo dei vertici del grafo associato a quella molecola, all'interno del modello NN4G.

Per attribuire un valore numerico al contributo di ogni vertice di un grafo si è, quindi, partiti dall'equazione del layer di uscita (2.27) del modello NN4G, riscrivendola

in una forma che mettesse in risalto il contributo dei vertici

$$\begin{aligned} y_o(\mathcal{g}) &= f_o \left(\sum_{h=0}^{N_h-1} \tilde{w}_{oh} X_h(\mathcal{g}) + b_o \right) = f_o \left(\sum_{h=0}^{N_h-1} \tilde{w}_{oh} \cdot \frac{1}{k} \sum_{v \in \mathbf{v}^{\mathcal{g}}} x_h(v) + b_o \right) = \\ &= f_o \left(\sum_{v \in \mathbf{v}^{\mathcal{g}}} \frac{1}{k} \sum_{h=0}^{N_h-1} \tilde{w}_{oh} x_h(v) + b_o \right) \end{aligned}$$

Dato che nel nostro caso utilizziamo un task di classificazione binaria, il modello presenterà un'unica uscita. Da qui possiamo definire la funzione *contributo* $C_v(\mathcal{g})$, corrispondente al contributo del vertice v rispetto al grafo \mathcal{g}

$$C_v(\mathcal{g}) = \frac{1}{k} \sum_{h=0}^{N_h-1} \tilde{w}_h x_h(v) \quad (5.1)$$

dove $\tilde{\mathbf{w}}$ sono i pesi dell'unità di uscita del modello. Nelle istanze addestrate, ed usate per i test eseguiti con il predittore, il valore di normalizzazione k si è posto pari al massimo numero di vertici nei grafi del training set. Questa formulazione permette di avere la seguente equivalenza in modo da avere l'equivalenza:

$$f \left(\sum_{v \in \mathbf{v}^{\mathcal{g}}} C_v(\mathcal{g}) + b \right) = y(\mathcal{g})$$

Mediante l'equazione (5.1) è possibile definire l'uscita del modello, in risposta ad una certa molecola, in base ai contributi dei singoli atomi. Questa informazione viene fornita all'esperto nella rappresentazione grafica della molecola, in modo da permetterli di identificare agevolmente le sotto-strutture che contribuiscono maggiormente alla formulazione della predizione.

Seguendo lo stile di altri tool, come T.E.S.T. (vedere Fig. 3.4), il predittore sviluppato fornisce all'esperto un'equazione (vedere Fig. 5.4) che mette in risalto i coefficienti usati dal modello per effettuare la predizione. A differenza di altri tool, dove mostravano come coefficienti i descrittori usati, nel nostro caso saranno presenti i vari contributi strutturali, con una corrispondenza binaria con gli atomi visualizzati nella rappresentazione grafica.

La rappresentazione grafica della molecola (visibile in Fig. 5.7) permette, nel suo complesso, di avere una buona sintesi visuale del contributo degli atomi nel formulare l'esito della predizione, anche nel caso si osservino diverse molecole alla volta (come nel caso visibile in Fig. 5.3).

Il colore assegnato al marker del contributo di un atomo permette di riconoscere rapidamente le seguenti informazioni:

- se il contributo è positivo o negativo, in base al colore: rosso e verde, rispettivamente;
- la grandezza del contributo, rispetto ai contributi massimi e minimi riscontrati dal modello sul training set, rappresentato mediante una trasparenza del marker: più il marker è trasparente e minore sarà il peso relativo del contributo;
- se il contributo supera o meno la soglia del modello, ovvero il bias dell'unità di uscita, rappresentato da una componente blu aggiunta ai colori rosso e verde del segno del contributo;

Questo sistema è finalizzato ad evidenziare graficamente il peso che i contributi hanno, positivo o negativo, nel formulare la predizione, facendo passare in secondo piano i contributi di minore peso (sia in assoluto, che rispetto alla soglia del modello).

Nel dettaglio, il colore c del marker è espresso nella tupla dei suoi componenti $c = (c_r, c_g, c_b, c_\alpha)$, dove $c_r, c_g, c_b \in [0, 1]$ sono le sue componenti rosse, verdi e blu e $c_\alpha \in [0, 1]$ è il suo coefficiente di trasparenza. Queste componenti sono formate nel seguente modo:

- $c_r = \max (Norm (C_v(\mathcal{g})), 0)$
- $c_g = \max (-Norm (C_v(\mathcal{g})), 0)$
- $c_b = \max \left(1 - \frac{|C_v(\mathcal{g})|}{|b|}, 0 \right)$
- $c_\alpha = |Norm (C_v(\mathcal{g}))|$

dove b è il valore bias dell'unità di uscita del modello, e $Norm(C_v(\mathcal{g}))$ è definito nel seguente modo:

$$Norm (C_v(\mathcal{g})) = \begin{cases} 1 & \overline{Norm} (C_v(\mathcal{g})) > 1 \\ -1 & \overline{Norm} (C_v(\mathcal{g})) < -1 \\ \overline{Norm} (C_v(\mathcal{g})) & \text{altrimenti} \end{cases} \quad (5.2)$$

con:

$$\overline{Norm} (C_v(\mathcal{g})) = 2 \cdot \frac{C_v(\mathcal{g}) - C_{min}}{C_{max} - C_{min}} - 1$$

dove C_{max} e C_{min} sono, rispettivamente, il massimo e il minimo contributo valutato su tutti gli atomi del training set usato per addestrare il modello.

In caso di bias dell'unità di uscita nullo, il colore blu sarà nullo.

Dato che i coefficienti dei colori richiedono un valore normalizzato, è necessario eseguire una normalizzazione dei contributi. L'equazione (5.1) è una combinazione lineare di coefficienti limitati in un certo intervallo⁵, dunque risulta teoricamente limitata nel codominio. In pratica, tuttavia, questo limite si applica solo in limitate situazioni e una normalizzazione in funzione di contributi massimi e minimi riscontrati nel dataset permette di risaltare meglio il colore dei contributi.

Per questo motivo si è preferito effettuare la normalizzazione di $C_v(\mathcal{g})$ in funzione dei valori massimi e minimi riscontrati nel training set, mediante la funzione \overline{Norm} e limitare il valore mediante la funzione $Norm$.

Il caso del componente c_b è particolare, e viene usato direttamente il valore del contributo, invece del suo valore normalizzato, in quanto deve essere messo a rapporto con il bias, che è un valore non limitato. Il valore c_b risulterà comunque entro il range necessario al colore, per come è definita la sua funzione.

Con questa definizione è possibile sintetizzare qualitativamente i valori di c_r , c_g e c_α in funzione del modulo $|C_v(\mathcal{g})|$ e segno $sgn(C_v(\mathcal{g}))$ del contributo del vertice v , come mostrato nella tabella 5.1.

$ Norm(C_v(\mathcal{g})) $	$sgn(Norm(C_v(\mathcal{g})))$	c_r	c_g	c_α
Alto	Positivo	Alto	0	Alto
Alto	Negativo	0	Alto	Alto
Basso	Positivo	Basso	0	Basso
Basso	Negativo	0	Basso	Basso

Tabella 5.1: Tabella di sintesi qualitativa dei componenti rosso, verde e della trasparenza, rispetto al valore assoluto del contributo normalizzato, e al suo segno.

Il valore c_b dipende dalla grandezza relativa dei moduli del contributo non normalizzato e il bias dell'unità di uscita del modello, e il suo andamento qualitativo è osservabile nella tabella 5.2. Questo coefficiente è stato definito in questo modo per avere un riscontro visivo sul superamento, da parte del contributo dell'atomo, del bias dell'unità di uscita. Il confronto col bias, chiamato anche *soglia*, è importante, in quando una molecola risulta positiva (o negativa) in base al superamento, mediante la combinazione lineare dei contributi dei suoi atomi, di questa soglia.

⁵I modelli che abbiano trattato prevedono un'uscita non lineare e limitata per le funzioni di attivazione delle unità nascoste.

$\text{sgn}(C_v(\mathcal{g}) - b)$	$ C_v(\mathcal{g}) - b $	c_b
Negativo	Alto	Alto
Negativo	Basso	Basso
Positivo	-	0

Tabella 5.2: Tabella di sintesi qualitativa del componente blu, rispetto al valore assoluto della differenza del contributo col bias, e al suo segno.

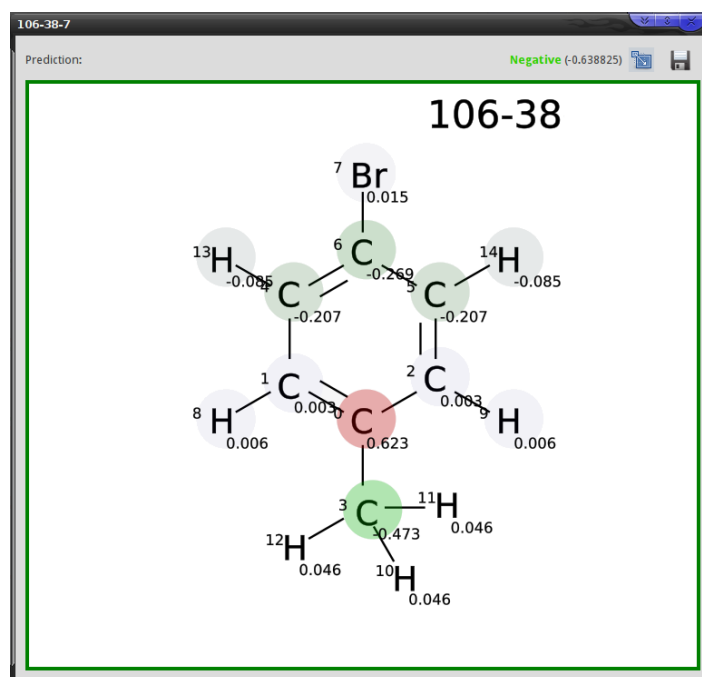


Figura 5.8: Un esempio di rappresentazione grafica di una molecola. È possibile notare come, a seconda del contesto, atomi di uno stesso elemento hanno un contributo molto diverso.

La molecola mostrata in Fig. 5.7, per le sue caratteristiche di simmetria, presenta lo stesso valore del contributo, per ogni atomo appartenente allo stesso elemento. Questo comportamento non è vero in generale, in quanto il valore del contributo si basa su un'informazione contestuale, e varia a seconda del tipo di struttura che viene analizzata.

Ad esempio, in Fig. 5.8 è possibile osservare la differenza del valore dei contributi del carbonio, ottenuta mediante la stessa istanza del modello usata per la Fig. 5.7. È possibile osservare che il contributo di un atomo viene assegnato dal modello in base all'informazione contestuale degli atomi vicini. A seconda del contesto in cui l'atomo di carbonio è inserito, nell'immagine di esempio, il valore del contributo varia, passando addirittura da positivo a negativo.

5.4 Struttura del Predittore

Come detto nella parte introduttiva del capitolo, l'implementazione del predittore fa uso degli strumenti sviluppati nel corso del progetto, *felix* per la gestione dei modelli, e *cougar* per il loro addestramento.

Per la parte grafica, la libreria fa uso degli strumenti messi a disposizione dalla libreria *Qt* per visualizzare le informazioni in maniera organica e dettagliata. In particolare, verrà sfruttata la separazione logica tra il contenuto di un'immagine *QGraphicScene* e la sua visualizzazione *QGraphicsView* utilizzata dalla libreria *Qt* per gestire la grafica. Questo dualismo permetterà di utilizzare la stessa scena per visualizzare la stessa molecola in diversi punti del programma, con notevole risparmio di memoria, specialmente nel caso di grandi dataset.

La libreria *OpenBabel* viene usata per la conversione delle molecole dai formati SMILES e SDF, per la loro conversione nel formato grafico vettoriale SVG, e per la ricerca di sotto-strutture molecolari mediante espressioni SMARTS.

La finestra principale del predittore è implementata dalla classe *PredictorWidget* che estende la classe *QMainWindow*, e si occupa di contenere e gestire le classi accessorie.

Nella fase di conversione di una molecola, questa viene convertita nella struttura *OBMol* di *OpenBabel*, la quale viene usata per facilitare la conversione in grafo da poter passare al modello, e per ottenere in modo semplice informazioni utili quali il suo fingerprint. La conversione avviene all'interno del costruttore di un'apposita classe *Molecule*, che implementa le seguenti le cui funzioni principali:

makePrediction funzione che si occupa di:

- convertire la struttura *OBMol* in un *Graph* della libreria *felix*;
- passare il grafo al modello e memorizzarne l'uscita;
- ottenere i contributi degli atomi e salvare in una stringa la relativa formula estesa del modello;
- salvare i risultati dell'analisi del dominio in funzione della molecola specifica;

addModelInfos funzione che prende la rappresentazione grafica della molecola, mediante stringa in formato vettoriale SVG, e la modifica. Per ogni atomo presente, viene aggiunto il suo contributo, sia come valore numerico che come marker.

Le operazioni di caricamento, conversione, analisi, e relativa rappresentazione grafica possono essere onerose per un'applicazione end-user interattiva, specialmente nel

caso di caricamento di dataset molto grandi. Per questo motivo si è scelto di suddividere le operazioni più onerose in diverse funzioni, ed optare per un'esecuzione dinamica delle stesse. Ad esempio, la scelta della visualizzazione delle molecole a mezzo di paginazione in un mosaico 5×4 , oltre a venire incontro a minimi criteri di leggibilità, ha l'importante scopo di permettere un caricamento *on-demand* delle molecole. A tempo di caricamento del dataset, vengono creati tanti oggetti *MoleculeWidget* quante sono le molecole presenti. La classe *MoleculeWidget*, essendo una sottoclasse di *Molecule*, ne erida il costruttore, il quale si limita a creare un oggetto *OBMol*.

Queste istanze di *MoleculeWidget* vengono inserite in una serie di strutture *MoleculePage*, le quali fungono da semplici contenitori. Quando una pagina deve essere caricata, nel caso in cui l'utente la selezioni con le apposite frecce, o perchè è la prima, viene invocata la funzione *showMols* della relativa *MoleculePage*, la quale si occupa di:

1. invocare la funzione *setGraphicalMolecule* di ogni istanza *MoleculeWidget* ivi contenuta, la quale, a sua volta:
 - I. invoca *makePrediction* della classe base *Molecule*;
 - II. crea la rappresentazione grafica della molecola in formato SVG, aggiungendo il marker dei contributi dei vari atomi;
 - III. aggiornare la *QGraphicsScene* con l'immagine della molecola, in modo da poterla visualizzare;
2. creare un'istanza della classe *MoleculeWidget_portrait* la quale, estendendo *QGraphicsView* permette la visualizzazione grafica dell'anteprima della molecola (visibile in Fig. 5.2 e Fig. 5.3);

Per motivi di efficienza, la classe *MoleculeWidget_portrait* userà come *QGraphicsScene* la stessa istanza contenuta nell'oggetto *MoleculeWidget* associato. Difatti le rappresentazioni grafiche dell'anteprima e del dettaglio della predizione sono le stesse, e vengono semplicemente rappresentate con due *QGraphicsView* diversi⁶.

Nella struttura del programma, la classe *PredictorWidget* implementa la finestra principale, la quale contiene tutte le altre strutture dati. Le informazioni dettagliate sulla singola molecola vengono mostrate grazie all'uso della classe *Molecule* che conserva tutte le informazioni molecolari ed implementa le operazioni di base della molecola, e *MoleculeWidget* che espone queste informazioni in maniera grafica.

⁶Il viewer usato per il dettaglio è un wrapper *MoleculeView* che permette lo zoom ed il panning, cosa non consentita nel caso dell'anteprima.

Le altre classi sono ausiliarie, e servono a rendere il sistema efficiente, come la classe *MoleculePage*, o graficamente accessibile, come la classe *MoleculeWidget_portrait*.

Molecule una delle classi principali, rappresenta tutte le informazioni relative ad una singola molecola, e conserva un riferimento al modello caricato. È responsabile di tutte le operazioni riguardanti la molecola, ed usa il riferimento al modello per effettuare la predizione;

MoleculeWidget estende *QWidget* e *Molecule* e conferisce alle strutture dati contenute nella classe *Molecule* la dovuta rappresentazione grafica. Questa classe implementa il dettaglio della molecola visibile in Fig. 5.7, e rappresenta graficamente tutte le informazioni contenute nella sua superclasse *Molecule*;

MoleculeView estende *QGraphicsView* aggiungendo semplicemente una funzione di *zoom* e *panning*;

MoleculeWidget_Portrait estende *QGraphicsView* e viene associato ad un'istanza di *MoleculeWidget* nel costruttore. Implementa la rappresentazione grafica di una molecola, nel formato “anteprima”, visualizzabile in *PredictorWidget* (vedere Fig. 5.2 e Fig. 5.3);

MoleculePage estende *QWidget* e contiene al suo interno diversi riferimenti di *MoleculeWidget* e *MoleculeWidget_portrait*. Si occupa della gestione delle pagine all'interno di *PredictorWidget* ed effettua il caricamento di tutte le molecole per cui è responsabile;

SimilarMoleculeWidget estende *MoleculeView* e *Molecule* e implementa la rappresentazione grafica visibile nel dettaglio delle molecole simili (vedere in Fig. 5.6);

PredictorWidget estende *QMainWindow* ed implementa la finestra primaria visibile in Fig. 5.2. Questa classe implementa le funzionalità di caricamento delle molecole, e contiene al suo interno un vettore di istanze di *MoleculePage* usato per visualizzare le molecole della pagina selezionata;

La differenza tra la rappresentazione grafica di una molecola visibile nell'anteprima della finestra principale (*PredictorWidget*) e quella visibile nel dettaglio (*MoleculeWidget*) è che quest'ultima viene implementata mediante *MoleculeView*, la quale permette lo *zoom* e il *panning* dell'immagine, cosa non permessa nell'anteprima.

A questa lista vanno aggiunte delle strutture dati ausiliarie che sono servite a gestire le informazioni del dataset di training, e la lista di structural alerts da visualizzare.

5.5 Rimozione Contributi Minori

Osservando la distribuzione dei contributi, ad esempio nella Fig. 5.8 si può notare che accanto a pochi atomi con contributi rilevanti, ve ne sono diversi con contributi di modulo ridotto. Ad esempio, sperimentalmente abbiamo notato che quasi tutti gli idrogeni delle molecole presenti in Bursi hanno associato un contributo in modulo ridotto. Questo in alcuni casi ha portato una distribuzione periferica di contributi, dove i contributi minori rischiavano di offuscare i contributi di sotto-strutture importanti per la predizione. Ad esempio, nella Fig. 5.9 è possibile notare come l'apparente

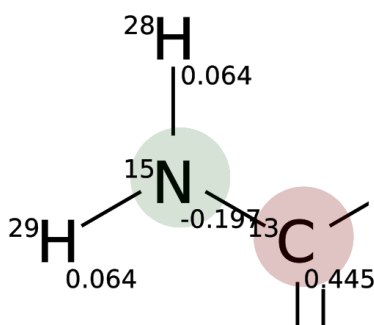


Figura 5.9: Il contributo negativo dell'azoto rende poco chiaro il contributo complessivamente neutro della componente NH_2

contributo negativo fornito dall'azoto nasconde il contributo sostanzialmente neutro della componente NH_2 , e nel complesso la componente CNH_2 risulta essere fortemente positiva.

Basandoci su queste osservazione, abbiamo ideato un sistema per addestrare un modello eliminando i contributi meno rilevanti, al fine di modificare la distribuzione dei contributi, togliendoli da atomi meno centrali per il task, come gli idrogeni, accentrandoli in strutture più critiche.

Per questo scopo abbiamo usato un'istanza di un modello addestrato \mathcal{M}_B , chiamato *modello base*, per addestrare una seconda istanza \mathcal{M}_D , chiamato *modello derivato* in modo che non considerasse i contributi sotto un certo *cutoff* ϵ , per generare l'uscita.

Per generare l'uscita, nel modello derivato \mathcal{M}_D sostituiamo la funzione di uscita (2.27) con la seguente:

$$y_o(\mathcal{g}) = f_o \left(\sum_{v \in \mathcal{V}^{\mathcal{g}}} \frac{1}{k} \sum_{h=0}^{N_h-1} \tilde{w}_{oh} I_h(v) + b_o \right) \quad (5.3)$$

dove:

$$I_h(v) = \begin{cases} x_h(v) & |C_v^{\mathcal{M}_B}(\mathcal{g})| > \epsilon \\ 0 & \text{altrimenti} \end{cases}$$

con $C_v^{\mathcal{M}_B}(\mathcal{g})$ contributo per il vertice $v \in \mathcal{g}$ calcolato sul modello base \mathcal{M}_B .

In questo modo si è implementato un sistema di riduzione dei contributi minori nel quale, il modello che effettua la previsione nel predittore \mathcal{M}_D usa le informazioni contenute nel modello base \mathcal{M}_B per stabilire quali atomi debbano essere considerati nel calcolare l'uscita. L'effetto è quello di ottenere un azzeramento dei contributi per gli atomi meno rilevanti e una redistribuzione dei valori nelle strutture principali.

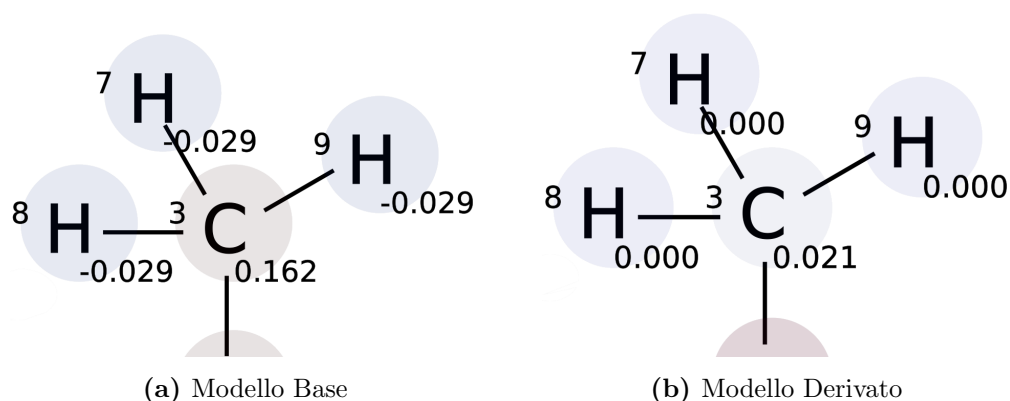


Figura 5.10: Esempio di ripartizione dei contributi nel caso si utilizzi il modello base (a) e nel caso del sistema di riduzione dei contributi minori con il modello derivato (b)

Nella Fig. 5.10 è mostrato come il valore del contributo di un carbonio risulti positivo, quindi a carattere tossico, mentre il gruppo CH_3 a cui appartiene risulti complessivamente neutro (vedere dettaglio di Fig. 5.10a). Utilizzando il modello derivato, ottenuto partendo dall'istanza che ha generato quella previsione, si nota come i contributi degli idrogeni vengano ripartiti, neutralizzando il gruppo CH_3 (vedere dettaglio di Fig. 5.10b). In Sez. 6.2.1 sono visibili ulteriori esempi, assieme ai dettagli dell'addestramento su alcune istanze.

Il modello \mathcal{M}_D viene creato copiando il modello base \mathcal{M}_B e addestrando, successivamente, solamente i pesi della sua unità di uscita, in base alla funzione (5.3) usando il calcolo della pseudoinversa.

Durante i test abbiamo selezionato il modello \mathcal{M}_D migliore, usando *felix* per addestrare diversi modelli:

$$\mathcal{M}_{\epsilon=0}, \dots, \mathcal{M}_{\epsilon=k}, \mathcal{M}_{\epsilon=k+\delta}, \dots, \mathcal{M}_{\epsilon=\max_{\epsilon}-1}$$

tutti basati sul modello base \mathcal{M}_B e ognuno addestrato con una soglia ϵ diversa, imposta in un certo range $[0, \max_{\epsilon}]$ e con un certo intervallo δ . Il modello \mathcal{M}_D ottenuto è stato selezionato, in questi test, prendendo il \mathcal{M}_i con migliore performance sul test set usato per valutare \mathcal{M}_B .

I risultati illustrati in Sez. 6.2.1 mostrano che, per ϵ bassi, oltre ad ottenere un accentrimento maggiore dei contributi in un numero più ridotto di atomi, questo sistema permette di ottenere in molti casi anche un incremento delle performance.

Capitolo 6

Risultati Sperimentali

In questo capitolo presenteremo i risultati di collaudo ottenuti mediante il sistema di training, e i risultati ottenuti mediante l'utilizzo del predittore, e del sistema di riduzione dei contributi minori sviluppato.

In Sez. 6.1 presenteremo una serie di test effettuati su benchmark noti in letteratura, costituiti da dataset chimici di differenti task e differenti endpoint. In Sez. 6.2 mostriamo come la distribuzione dei contributi abbia fatto emergere alcune corrispondenze tra sotto-strutture con contributi positivi, per un endpoint di mutagenicità, e alcune structural alerts note in letteratura per quell'endpoint. Mostriamo inoltre, come l'applicazione del sistema di riduzione dei contributi minori ridistribuisca i contributi in modo da far risaltare le sotto-strutture rilevanti per la predizione dell'endpoint.

6.1 Risultati di Collaudo del Sistema di Addestramento

Il sistema di gestione e addestramento di istanze del modello *NN4G* illustrato in Cap. 4 è stato sottoposto ad una serie di test di collaudo su alcuni dataset chimici noti in letteratura. I dataset presi in considerazione rappresentano task di regressione e classificazione, tutti con un singolo valore target. Nella maggior parte dei casi, i dataset erano presenti in formato SDF e sono stati convertiti in *.gph* per poter essere usati dal sistema, a mezzo di alcuni script Python non illustrati in dettaglio in questa tesi.

In particolare, i test svolti erano mirati a valutare le performance generali del sistema rispetto a benchmark in cui il modello era stato testato in precedenza, come nel

caso del dataset degli alcani, e su altri per cui erano disponibili risultati di altri modelli per grafi.

Tra i numerosi test svolti, si è scelto di rappresentare in questo contesto quelli funzionali ad una comparazione tra l'algoritmo di apprendimento *Cascade Correlation* (CC), visto in Sez. 2.3.1, e il *Cascade Residual* (CR), visto in Sez. 2.3.2. Questo confronto è particolarmente interessante, in quanto non esiste attualmente in letteratura un risultato riguardante l'applicazione dell'algoritmo costruttivo CR sul modello *NN4G*.

Molti test qui rappresentati prendono in considerazione anche l'applicazione dell'algoritmo *Ridge Regression*, visto in Sez. 2.2.2.2, per l'addestramento locale dell'unità di uscita nel caso dell'algoritmo CC.

Gli indici usati per rappresentare le performance dei modelli sono quelli descritti in Sez. 2.1.

6.1.1 Alcani

Questo dataset [2], presente in formato *.gph*, costituisce una collezione di 150 alcani (idrocarburi saturi) con task di regressione rappresentato dalla predizione della temperatura di ebollizione del composto. Il valore dell'endpoint dei composti presenti nel dataset, variabile da -164 a 174 gradi Celsius, viene scalato nel dataset nella forma di 10^{-2} gradi, arrivando quindi ad avere $t \in [-1.64, 1.74]$ per ogni valore target t del dataset. I vertici dei grafi presenti nel dataset rappresentano gruppi molecolari di tipo CH, CH_1, \dots, CH_4 e i vettori di feature dei vertici hanno dimensione unitaria con etichetta fissa al valore 1. Usando questa particolare codifica il modello, durante l'addestramento, fa totale affidamento sull'informazione strutturale della molecola per fornire il risultato, in quanto l'informazione distintiva dei vertici non è presente.

Algoritmo	Training	Validation	Test	Risultati Completi
CC	1.41 (± 0.13)	1.67 (± 0.03)	1.88 (± 0.7)	Sez. D.1.1
CR	4.06 (± 1.09)	4.7 (± 0.4)	5.04 (± 1.4)	Sez. D.1.2

Tabella 6.1: Riassunto dei risultati sugli alcani, al variare degli algoritmi usati. Il risultato rappresenta il $MaE \cdot 10^2$ ed è espresso in gradi centigradi.

I test sugli alcani sono stati eseguiti usando l'algoritmo QuickPropagation con $\eta = 0.004$ per l'addestramento delle unità candidate, con stop criteria globale raggiunto se il *MaxAbsErr* è inferiore a 5 gradi. Il test è stato eseguito su una DCV con 10 fold esterne e 10 interne. I parametri a griglia sono il coefficiente di penalizzazione del metodo weigh decay per quanto riguarda l'algoritmo di apprendimento delle unità candidate, l'intervallo dell'incremental strategy per lo stop criteria locale e, nel caso della variante con la Cascade Correlation, il parametro di regolarizzazione λ del calcolo della Ridge Regression (vedere Sez. 2.2.2.2) nell'algoritmo di apprendimento per l'unità di uscita. Nel caso di $\lambda = 0$ l'algoritmo di Ridge Regression per l'unità di uscita verrà sostituito dal calcolo della pseudoinversa (vedere Sez. 2.2.2.1). I risultati, espressi in gradi Celsius, sono presenti in forma sintetica nella Tab. 6.1.

Nei test effettuati, il risultato della variante CC dell'algoritmo usato per il modello *NN4G* implementato in *felix* risulta allineato a quanto presente in letteratura per quel modello [2] dove i migliori risultati in training e test erano 1.33 e 1.74 gradi, rispettivamente. Nel caso della variante CR otteniamo, invece, un netto peggioramento delle performance, dove l'errore medio assoluto è quasi triplicato rispetto alla variante CC. Inoltre i risultati mostrati in Tab. D.1 mostrano che l'uso della Ridge Regression porta, in questo caso, ad un deterioramento delle performance.

Le performance di una fold interna di questo test è stata riportata come esempio di report generato dal tool *felix-reader* in App. F.

6.1.2 Predictive Toxicology Challenge

Il dataset PTC (**P**redictive **T**oxicology **C**hallenge) [61] è rappresentato in formato SDF e conta 417 molecole, ognuna delle quali presenta fino a quattro endpoint diversi riguardanti la tossicità di alcuni roditori.

PTC.MM tossicità per il topo maschio (**M**ale **M**ice), comprendente 336 molecole;

PTC.FM tossicità per il topo femmina (**F**emale **M**ice), comprendente 349 molecole;

PTC.MR tossicità per il ratto maschio (**M**ale **R**at), comprendente 344 molecole;

PTC.FR tossicità per il ratto femmina (**F**emale **R**at), comprendente 351 molecole;

Come è possibile osservare, non tutte le molecole presentano risultati per tutti e quattro gli endpoint supportati dal dataset. Nella conversione è stata considerata la codifica *10K* dei 22 elementi presenti nel dataset, assieme ai valori di carica e radical dell'atomo

per costituire il vettore di feature per i vertici, Per quanto riguarda il target, i valori $\{+1, -1\}$ sono stati usati per codificare la tossicità e non tossicità, rispettivamente, della molecola per un dato endpoint.

Dataset	Algoritmo	Training	Validation	Test	Risultati Completi
<i>PTC.FM</i>	CC	0.8055 (± 0.004)	0.6513 (± 0.012)	0.5928 (± 0.095)	Sez. D.2.1
	CR	0.7281 (± 0.08)	0.6233 (± 0.024)	0.58445 (± 0.086)	Sez. D.2.2
<i>PTC.FR</i>	CC	0.8065 (± 0.005)	0.6825 (± 0.011)	0.6273 (± 0.116)	Sez. D.3.1
	CR	0.6726 (± 0.109)	0.6708 (± 0.009)	0.6045 (± 0.0908)	Sez. D.3.2
<i>PTC.MM</i>	CC	0.8032 (± 0.003)	0.6749 (± 0.011)	0.6540 (± 0.092)	Sez. D.4.1
	CR	0.7305 (± 0.052)	0.6593 (± 0.018)	0.6126 (± 0.076)	Sez. D.4.2
<i>PTC.MR</i>	CC	0.8081 (± 0.006)	0.6120 (± 0.016)	0.5521 (± 0.065)	Sez. D.5.1
	CR	0.7535 (± 0.057)	0.5936 (± 0.015)	0.6079 (± 0.097)	Sez. D.5.2

Tabella 6.2: Riepilogo accuracy media per training, validation e test per i vari task, con variante CC e CR

I test visibili in Tab. [6.2](#) sono stati eseguiti mediante una double cross validation, con 10 folds esterne, e 5 interne. Come parametri base abbiamo usato l'algoritmo Quickpropagation ($\eta = 0.1$) con Incremental Strategy da 20 a 100 epoche, per l'addestramento dell'unità candidata. Nel caso della Cascade Correlation abbiamo usato il metodo del Ridge Regression per l'addestramento dell'unità di uscita.

Per la griglia della double cross validation, abbiamo fatto variare il parametro di penalizzazione del Weight Decay delle unità candidate, da 0 a 0.1, l'intervallo dell'incremental strategy, corrispondente a valori da 10 a 30 epoche, e il parametro di regolarizzazione λ del metodo Ridge Regression per l'unità di uscita, da 0 a 0.1. Il parametro λ è presente solo nel caso della Cascade Correlation e, come nel caso degli

alcuni, un valore di $\lambda = 0$ corrisponde all'uso della pseudoinversa al posto del Ridge Regression, per l'unità di uscita.

Metodo	FM	FR	MM	MR
SVM	0.645	0.669	0.664	0.657
GraphESN	0.604 (± 0.009)	0.671 (± 0.001)	0.65 (± 0.007)	0.574 (± 0.033)
GraphESN-NG	0.625 (± 0.002)	0.667 (± 0.002)	0.648 (± 0.001)	0.612 (± 0.002)
NN4G (CC)	0.592 (± 0.095)	0.627 (± 0.116)	0.654 (± 0.092)	0.552 (± 0.065)
NN4G (CR)	0.584 (± 0.086)	0.604 (± 0.0908)	0.612 (± 0.076)	0.608 (± 0.097)

Tabella 6.3: *Comparativa delle performance di accuracy in test, per i quattro task del dataset PTC.*

Benchè un confronto completo con la letteratura vada al di là degli scopi di questo lavoro, i risultati ottenuti dalle due varianti, visibili in Tab. 6.3, sono comparati con i migliori risultati presenti in [62] che utilizzano un SVM con diversi kernel per grafi, il modello *GraphESN* visto in Sez. 2.4.1.7 ed una sua variante. I modelli basati su *GraphESN* [63] differiscono dall'uso di due varianti differenti di SMF. Nel caso della *GraphESN* indicata in Tab. 6.3 come SMF viene utilizzata una media degli stati, mentre nella variante *GraphESN-NG* viene utilizzata una SMF adattiva.

Su questi test entrambe le varianti dell'algoritmo di apprendimento del modello *NN4G* non mostrano, nei limiti delle prove effettuate, un miglioramento rispetto ai risultati presenti in letteratura. I test preliminari svolti su questo dataset hanno mostrato la tendenza del modello ad andare in overfitting dopo poche epoche, probabile causa delle performance inferiori rispetto alle altre soluzioni. A questo proposito è tuttavia utile notare che i risultati ottenuti con questo test (ad esempio quelli visibili in Tab. D.12) mostrano che il modello ha beneficiato, in alcuni casi, della presenza della regolarizzazione rappresentata sia dalla penalizzazione del weight decay, per le unità candidate, che dall'uso della Ridge Regression per l'unità di uscita.

6.1.3 Bursi

Il dataset Bursi [64, 65] è presente in formato SDF e presenta la valutazione dell'endpoint di mutagenicità valutato su 4337 molecole. Questo dataset è di particolare importanza per questo lavoro in quanto, oltre a costituire un ulteriore elemento di collaudo per il sistema di addestramento, è stato usato per l'addestramento delle istanze

usate dal predittore per i test mostrati in Sez. 6.2. Per questa parte, si è scelto di usare questo dataset in quanto erano presenti in letteratura informazioni riguardo structural alerts sull'endpoint di mutagenicità.

Oltre al grande numero di molecole presenti, il dataset è stato originalmente [65] separato in due dataset, uno di training e uno di test, prendendo in considerazione le caratteristiche chimiche dei composti rappresentati, in modo da avere una distribuzione uniforme. Per questa separazione, alcune molecole sono state rimosse, seguendo una procedura di controllo della qualità, portando il numero di molecole effettivamente usate per il training set e test set a 3367 e 837, rispettivamente, per un totale di 4204 molecole effettivamente usate.

Come valore delle feature dei vertici è stata usata la codifica 1oK degli 11 elementi presenti nel dataset, a cui è stato aggiunto il valore di carica e di *stereo parity* dell'atomo, codificati normalizzando i valori rispetto al valore massimo esprimibile in SDF, riguardo le rispettive proprietà. L'endpoint del dataset corrisponde alla mutagenicità sulla *salmonella typhimurium* ottenuta mediante il test di Ames [66] il quale possiede un'affidabilità stimata attorno all'85%. Il valore target è stato codificato con i valori $\{+1, -1\}$ che rappresentano i casi di molecola mutagenica e non mutagenica, rispettivamente.

Per l'esecuzione dei test preliminari si è preferito, visto la grande quantità di dati a disposizione, evitare l'onere di effettuare una DCV, come nel caso dei dataset precedenti. Inoltre, avere il dataset ripartito in modo bilanciato tra training e test set ci ha portato ad effettuare un lavoro di model selection preliminare sul training set per trovare gli iperparametri migliori, i quali sono stati usati per addestrare il modello su tutto il training set e valutare le performance sul test set.

Sebbene i risultati di alcune forme di regolarizzazione, come l'uso della Ridge Regression per l'unità di uscita e il weight decay per le unità candidate, abbiano deteriorato le performance sul dataset di regressione usato (vedere Sez. 6.1.1), i risultati ottenuti nel PTC suggeriscono che queste tecniche possono migliorare le performance in determinati casi. Per questo motivo, abbiamo voluto testare anche su questo dataset entrambe queste tecniche, negli esperimenti di model selection, per valutare il loro impatto sulle performance.

I test preliminari di model selection svolti per la variante CC dell'algoritmo di apprendimento del modello NN_4G sono stati eseguiti mediante una CV da 3 folds nel training set di Bursi. I risultati ritenuti più significativi sono mostrati in Tab. 6.4 e

sono stati ottenuti variando il parametro di penalizzazione del weight decay e il coefficiente λ della Ridge Regression. Come nei casi precedenti, un valore $\lambda = 0$ sta a significare l'uso della pseudoinversa al posto del metodo del Ridge Regression per l'unità di uscita. Gli altri parametri sono stati lasciati fissati, e comprendono l'uso della Quickpropagation con $\eta = 0.1$, un'incremental strategy da 20 a 200 epoche con un intervallo di 10 per l'unità candidata, uno stop criteria globale fissato ad 0.85 di accuracy in training e 150 unità massime. Riguardo gli stop criteria globali, l'accuracy massima è stata scelta per le limitazioni teoriche del test di Ames su cui è basato l'endpoint del dataset, mentre il numero massimo di unità è derivato da alcuni test preliminari, che hanno mostrato una tendenza del modello ad andare in overfitting quando il numero di unità superava una certa soglia. I test preliminari hanno mostrato una certa variabilità del risultato nel caso in cui si usi il numero massimo di vertici (variante k_{norm}) per il coefficiente k dell'equazione (2.28) del modello NN_4G e nel caso in cui si usi la media dei vertici (variante k_{avg}). Per questo motivo si è ritenuto opportuno effettuare i test su entrambe queste varianti. I risultati presenti in Tab. 6.4 mostrano un generale mi-

		k_{norm}		k_{avg}	
Weight Decay	λ	Acc	Unità	Acc	Unità
0	0	0.74	87	0.78	91
0	0.01	0.75	79	0.76	131
0.01	0	0.75	96	0.76	83
0.01	0.01	0.74	80	0.77	150

Tabella 6.4: Risultati della model selection effettuata sul dataset di training, per la variante CC dell' algoritmo di apprendimento, rispetto alla variante normalizzata e media del valore k . Il valore di accuracy e del numero di unità è una media delle varie fold di validation.

glioramento delle performance nel caso in cui si usi la media dei vertici nella funzione (2.28) rispetto alla normalizzazione mediante il numero massimo di vertici. Tuttavia, è opportuno notare che la variante k_{norm} beneficia maggiormente della regolarizzazione imposta dalle tecniche di weight decay e di Ridge Regression, rispetto alla variante k_{avg} , senza che questa rallenti sensibilmente l'addestramento.

I risultati ottenuti hanno portato ad usare la variante k_{avg} senza regolarizzazione per il test finale, il quale è stato eseguito usando la separazione originaria del dataset e la variante CC dell' algoritmo, con i parametri espressi in precedenza. Le performance

Trial	<i>MaE</i>	<i>MaxAbsErr</i>	Unità	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
<i>tr</i> ₁	0.544631	2.20598	150	0.847755	0.86313	0.864965	0.825911
<i>tr</i> ₂	0.576597	2.22848	89	0.850729	0.8705	0.861244	0.837382
<i>tr</i> ₃	0.628142	2.39464	150	0.804639	0.812564	0.845827	0.752362
<i>tr</i> ₄	0.545086	2.32658	113	0.853405	0.878821	0.855928	0.850202
<i>tr</i> ₅	0.556509	2.64224	84	0.851918	0.868015	0.867092	0.832659
Media	0.570	2.360	117.200	0.842	0.859	0.859	0.820
Dev. Std.	0.035	0.175	31.886	0.021	0.026	0.009	0.039

Tabella 6.5: *Performance di training della variante CC al variare dei trial*

Trial	<i>MaE</i>	<i>MaxAbsErr</i>	Unità	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
<i>tr</i> ₁	0.691283	3.29412	150	0.765755	0.793478	0.781585	0.745989
<i>tr</i> ₂	0.674882	2.66338	89	0.782402	0.807359	0.798715	0.762032
<i>tr</i> ₃	0.66738	3.03332	150	0.78478	0.793033	0.828694	0.729947
<i>tr</i> ₄	0.65044	4.08677	113	0.816885	0.838013	0.830835	0.799465
<i>tr</i> ₅	0.663763	6.03417	84	0.772889	0.792373	0.800857	0.737968
Media	0.670	3.822	117.200	0.785	0.805	0.808	0.755
Dev. Std.	0.015	1.342	31.886	0.020	0.020	0.021	0.028

Tabella 6.6: *Performance di test della variante CC al variare dei trial*

finali sono state ottenute eseguendo una media tra i risultati derivati dall'esecuzione di 5 trial differenti, visibili in Tab. 6.5 e Tab. 6.6, che corrispondono a 5 addestramenti di altrettante istanze del modello, usando gli stessi parametri di training e variando l'inizializzazione dei pesi. È possibile osservare l'andamento dell'addestramento delle varie istanze sul training set, in Fig. 6.1, e sul test set, in Fig. 6.2.

Per quanto riguarda la variante CR dell'algoritmo di apprendimento, i risultati ottenuti sul dataset degli alcani e sul PTC hanno mostrato un comportamento generalmente peggiore in quasi tutte le occasioni in cui è stato confrontato col la variante CC. Risultati ottenuti da test preliminari svolti sul dataset Bursi hanno confermato questa tendenza anche per questo dataset, ottenendo performance peggiori e una convergenza generalmente più lenta. Per motivi di completezza si è deciso di effettuare comunque un test finale con questa variante, usando parametri di addestramento simili a quelli usati per il test finale della variante CC, aumentando il numero di epoche massime a 200 al fine di compensare la lentezza della convergenza. Il risultato mostrato in Tab. 6.7, conferma anche per questo dataset il peggioramento delle performance, rispetto alla variante CC, e la lentezza della convergenza del risultato, fermandosi su

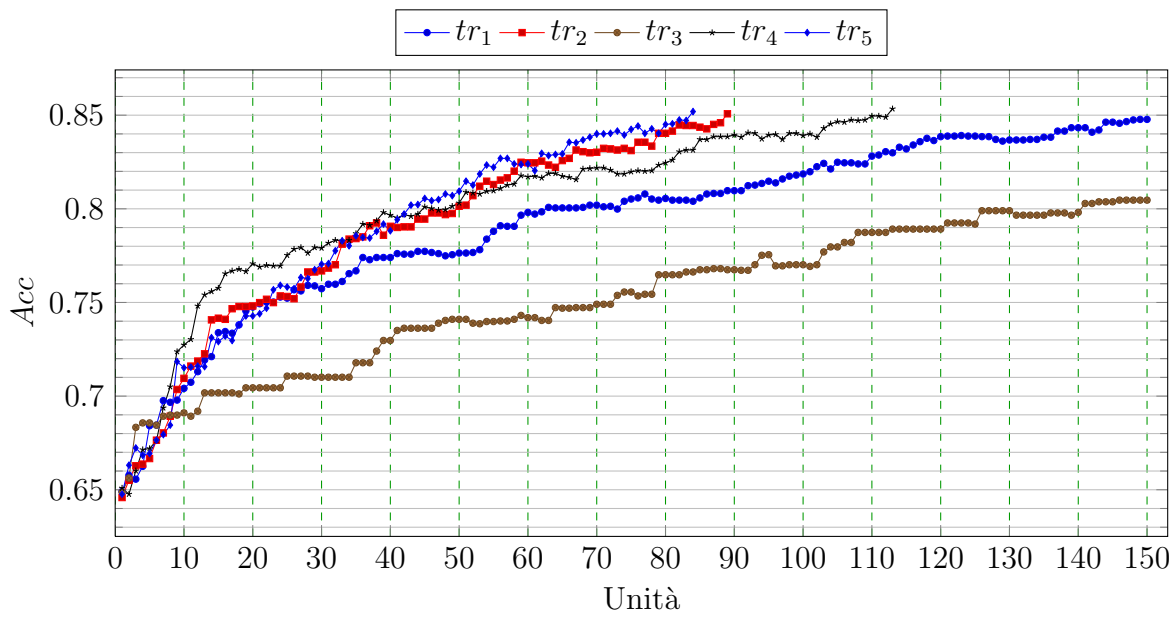


Figura 6.1: Andamento dei vari trial sul training set, all'aumentare del numero di unità

un'accuracy di 0.73 in training al raggiungimento del numero massimo di unità fissate.

Dataset	<i>MaE</i>	<i>MaxAbsErr</i>	Unità	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
Training	0.739682	2.4357	200	0.734761	0.75582	0.776715	0.681511
Test	0.73739	2.39681	200	0.728894	0.749478	0.768737	0.679144

Tabella 6.7: Performance della variante CR per il training e test set

Metodo	<i>Acc_{test}</i>
SVM	0.832
GraphESN-NG	0.7924 (± 0.06)
NN4G (CC)	0.785 (± 0.02)
Benigni/Bossa	0.783
GraphESN	0.758 (± 0.05)
NN4G (CR)	0.728
lazar	0.694

Tabella 6.8: Confronto dei risultati finali di collaudo sul dataset Bursi, rispetto allo stato dell'arte.

Anche in questo contesto, un confronto completo con la letteratura andrebbe oltre gli scopi di questo lavoro, tuttavia i risultati ottenuti dal sistema per le due varianti di algoritmi di apprendimento CC e CR, visibili in Tab. 6.8, sono confrontati con un

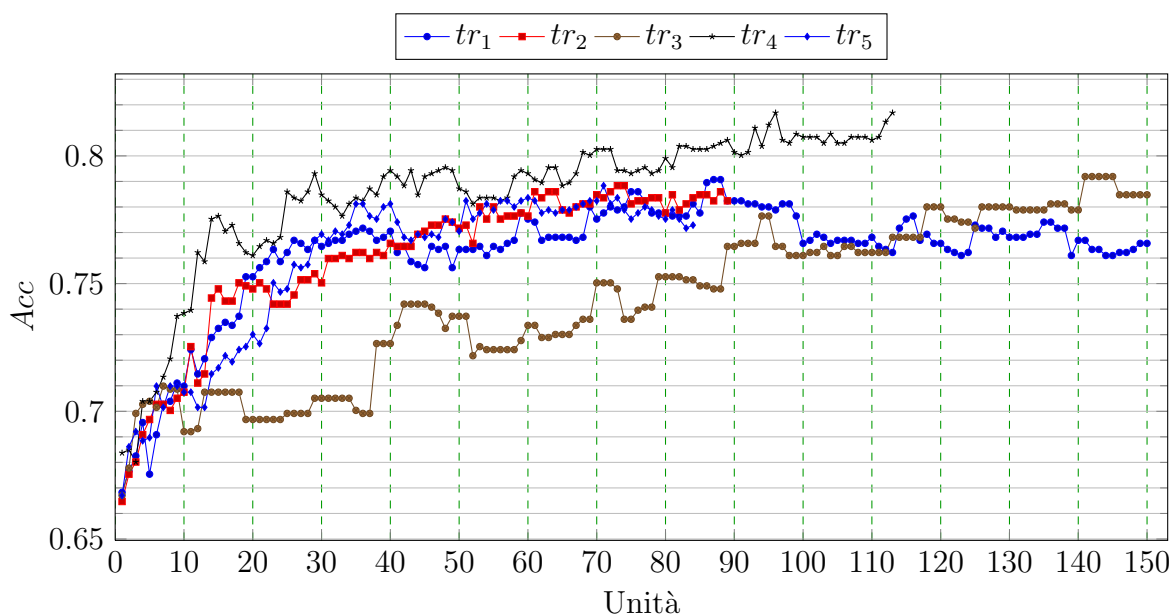


Figura 6.2: Andamento dei vari trial sul test set, all'aumentare del numero di unità

sistema che fa uso di una SVM, i due modelli basati sul *GraphESN* visti nel caso del dataset PTC, il sistema *lazar* [67] e il sistema rulebase Benigni/Bossa [51], basato sul riconoscimento delle SA. In questo caso, il sistema basato su SVM [68] fa uso di un insieme di 27 descrittori selezionati mediante un processo di features selection specifico per il task in questione. Il metodo *lazar* è un sistema che basa la sua predizione su un certo numero di molecole simili a quella da predire, in base ad un criterio di similarità basato su frammenti.

La variante CC dell'algoritmo di apprendimento usato per il modello *NN4G* permette di ottenere performance comparabili con lo stato dell'arte. In particolare, il modello risulta essere vicino alle performance variante NG della *GraphESN*, ottenendo risultati migliori della versione base del modello *GraphESN*, del sistema Benigni/Bossa e del sistema *lazar*. Anche in questo caso la variante CR mostra performance decisamente inferiori rispetto alla variante CC, riuscendo ad ottenere performance migliori solamente rispetto al sistema *lazar*.

6.2 Risultati del Predittore

La capacità di identificare una distribuzione dei contributi sugli atomi di una molecola, vista in Sez. 5.3, ci ha permesso di analizzare il contributo delle sotto-strutture molecolari nella formulazione dell'esito di tossicità del modello.

Analizzando diverse istanze di modelli addestrati con *cougar* (vedere Sez. 4.3.1) sul dataset Bursi (vedere Sez. 6.1.3) abbiamo trovato, in diversi casi, una corrispondenza tra la distribuzione di alcuni contributi positivi e alcune structural alert note in letteratura (vedere Sez. 3.1.2). Queste sotto-strutture sono state usate dal modello per formulare una predizione positiva della molecola che le contiene, quindi associandole ad una caratteristica tossica, senza che avesse alcuna informazione specifica in fase di training.

Le immagini presenti in questa sezione sono state ottenute mediante l'addestramento di 4 istanze di uno stesso modello le cui performance raggiunte, visibili in Tab. 6.9, fanno riferimento al training set e test set del dataset Bursi (vedere Sez. 6.1.3).

Istanza	Acc_{tr}	Acc_{test}	Saturazione
\mathcal{M}_1	0.815938	0.770511	0.867829
\mathcal{M}_2	0.819209	0.772889	0.865841
\mathcal{M}_3	0.820398	0.7717	0.873144
\mathcal{M}_4	0.798692	0.751486	0.84268

Tabella 6.9: Performance delle varie istanze addestrate, con i valori di Acc di training e test, e la saturazione media delle unità del modello.

I risultati sperimentali hanno mostrato che il modello, indipendentemente dalle istanze, riesce ad isolare alcune sotto-strutture molecolari note in letteratura, come alcune structural alerts (SA) citate in Sez. 3.1.2, come è possibile osservare in Fig. 6.3. Un esempio riportato in questa sezione è quello della SA 21 visibile in Fig. 6.4a, che è stata rilevata in diverse molecole, visibili in Fig. 6.4b e 6.4c, in modo indipendente dall'istanza addestrata. Diverse istanze hanno mostrato di ripartire i contributi in modo diverso, ma tutte attribuiscono ad alcune sotto-strutture molecolari, in particolare quelle corrispondente a structural alert, una componente fortemente positiva. Questo suggerisce la tendenza del modello a focalizzare i contributi positivi in alcune sotto-strutture, per le quali è stata rilevata una corrispondenza con la conoscenza degli esperti del dominio in analisi, senza che il modello avesse ricevuto alcuna informazione di questo tipo in fase di addestramento. La dispersione dei contributi rispetto all'istanza è spiegabile dal fatto che il modello NN4G, pur utilizzando l'informazione contestuale, non viene addestrato basandosi sul riconoscimento di sotto-strutture ma, in modo più generale, utilizza le informazioni dei singoli vertici, e del contesto strutturale, per

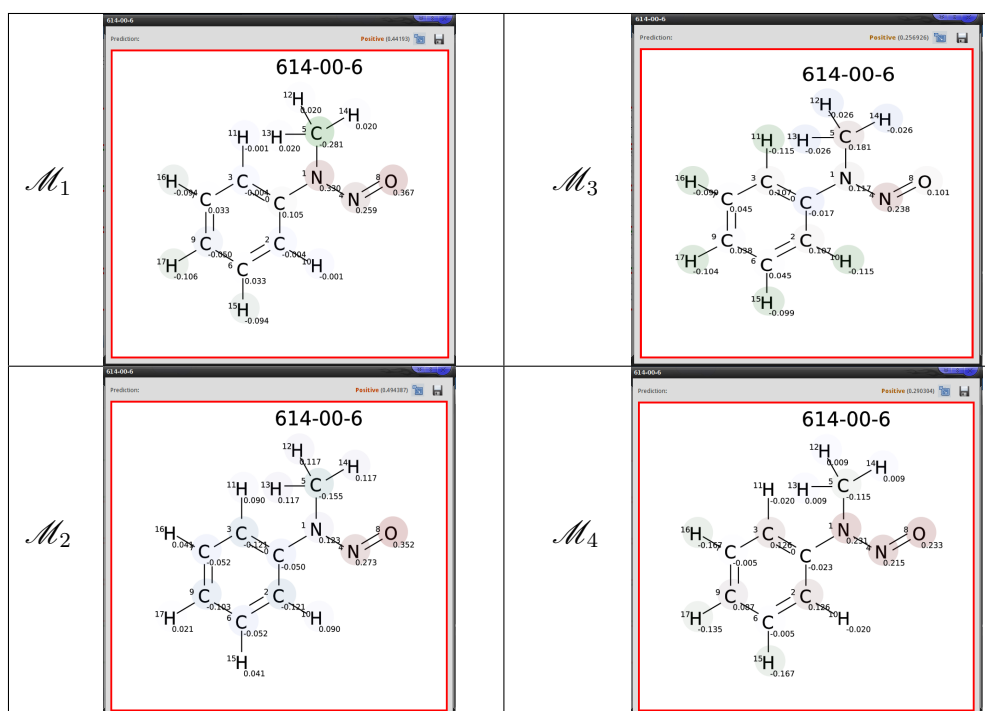


Figura 6.3: Esempio di SA 21 identificata dalle varie istanze del modello.

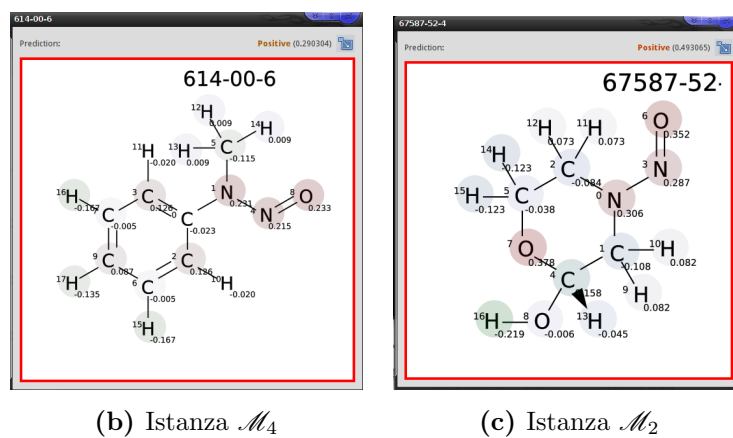
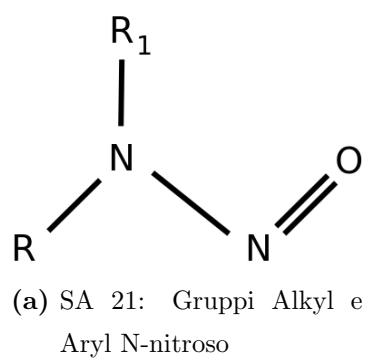
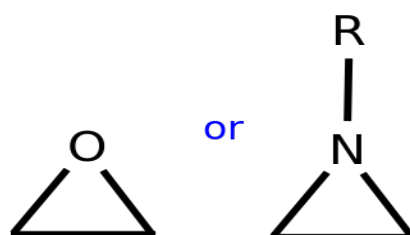


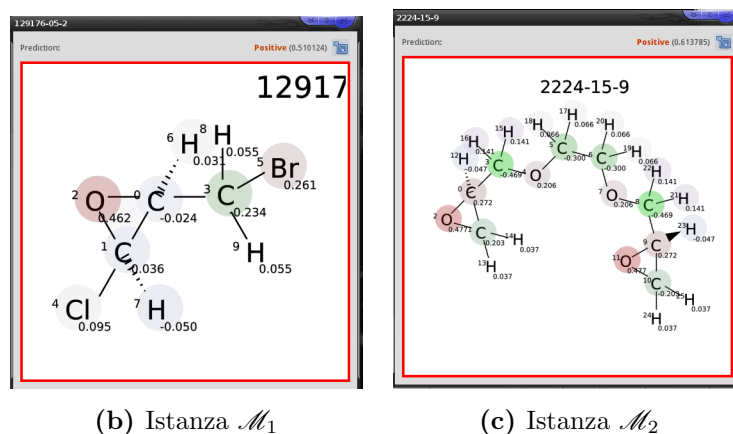
Figura 6.4: La SA 21 è stata rilevata in diverse molecole, indipendentemente dall'istanza addestrata.

formulare la predizione con una ampia libertà di attribuzione dei contributi alle varie componenti del contesto.

Spesso il contributo positivo è presente nella maggior parte degli atomi costituenti la SA, ma in alcuni casi è stata osservata un'attribuzione della componente positiva limitatamente ad una parte della SA, o anche ad un solo atomo, permettendo comunque di identificare nella grafica la zona molecolare di rilievo. Ad esempio, come è osservabile in Fig. 6.5, la componente positiva della SA 7 viene attribuita all'ossigeno del composto OC_2 .



(a) SA 7: Epoxides e Aziridines

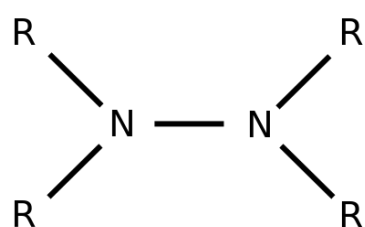


(b) Istanza \mathcal{M}_1

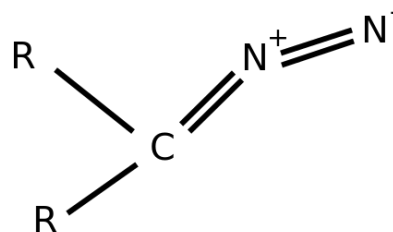
(c) Istanza \mathcal{M}_2

Figura 6.5: Il contributo positivo di alcune structural alert viene attribuito a sotto-strutture, identificate dal modello come caratterizzanti.

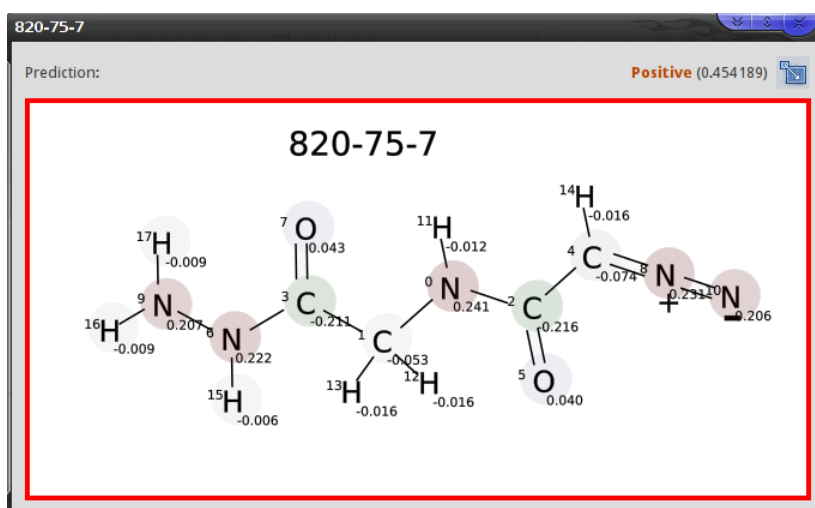
Per finire, si riportano casi in cui il modello attribuisce dei contributi positivi a structural alerts multiple presenti nella stessa molecola, come si vede in Fig. 6.6. Questo mostra le capacità del modello di ripartire i contributi in diverse sotto-strutture di una stessa molecola, qualora vengano considerate entrambe rilevanti ai fini della predizione.



(a) SA 13: Gruppo Hydrazine



(b) SA 14: Gruppi Aliphatic Azo e Azoxy



(c) Molecola con Entrambe le SA

Figura 6.6: Molecola che presenta contemporaneamente la SA 13 e la SA 14, rilevate entrambe da \mathcal{M}_4 .

6.2.1 Variante con Rimozione dei Contributi Minori

In questa sezione, mostreremo i risultati ottenuti applicando il sistema di riduzione dei contributi minori illustrato in Sez. 5.5. A titolo di esempio mostreremo i risultati ottenuti mediante l'addestramento del sistema di riduzione dei contributi minori usando come modelli base le istanze $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4$ addestrate per ottenere i risultati visti nella precedente sezione.

Come prova preliminare, abbiamo valutato il comportamento del modello base senza alcun ulteriore addestramento, al variare del *cutoff* ϵ . I risultati, visibili in Fig. 6.7 sono stati valutati sul training set usato per l'addestramento del modello base,

estrapolato dal dataset Bursi, e sul relativo test set. Le performance originarie delle istanze, analoghe al caso $\epsilon = 0$, costituiscono la nostra *baseline* e sono riportate sui grafi mediante linee orizzontali. Questo test sperimentale ha mostrato che eliminando sem-

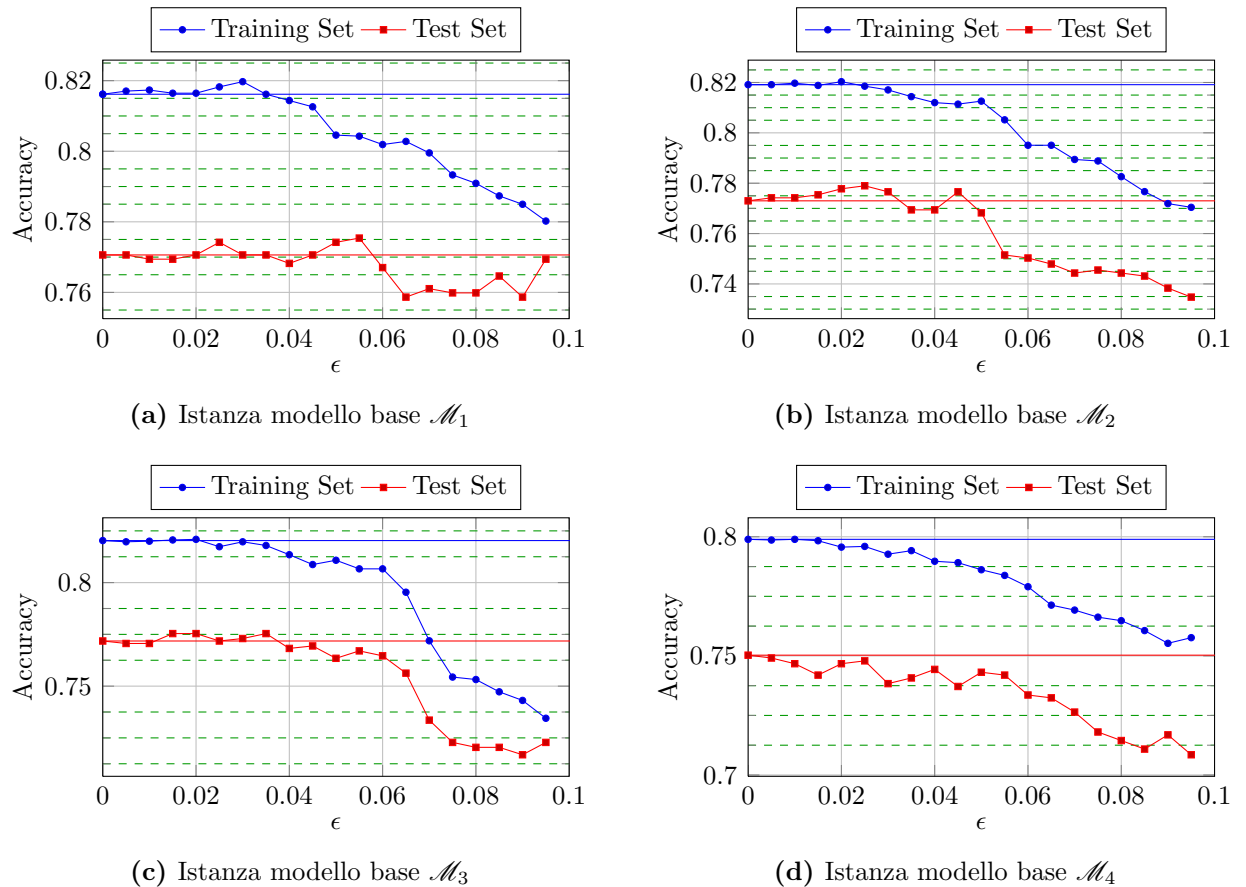


Figura 6.7: Risultati dei modelli base, al variare del cutoff ϵ . Le barre blu e rosse rappresentano la baseline dell'istanza per il training e test set, rispettivamente.

plicemente i contributi minori, senza quindi la generazione del modello derivato basato sul cutoff, le performance in test non calano molto, mostrando in limitati casi perfino un miglioramento. Le performance in training, tuttavia, si deteriorano rapidamente in tutti i test effettuati.

Questo deterioramento delle performance in training è mitigato quando viene effettuato un riaddestramento del modello, usando il calcolo della pseudoinversa, generando quindi i max_ϵ modelli derivati, addestrati basandosi sulla rimozione dei contributi minori valutati sul modello base. Come è visibile in Fig. 6.8, modelli derivati addestrati con un $\epsilon < 0.1$ mostrano un decadimento minore delle performance, arrivando addirittura a migliorare in test, segno che i vertici a cui corrispondevano contributi in modulo

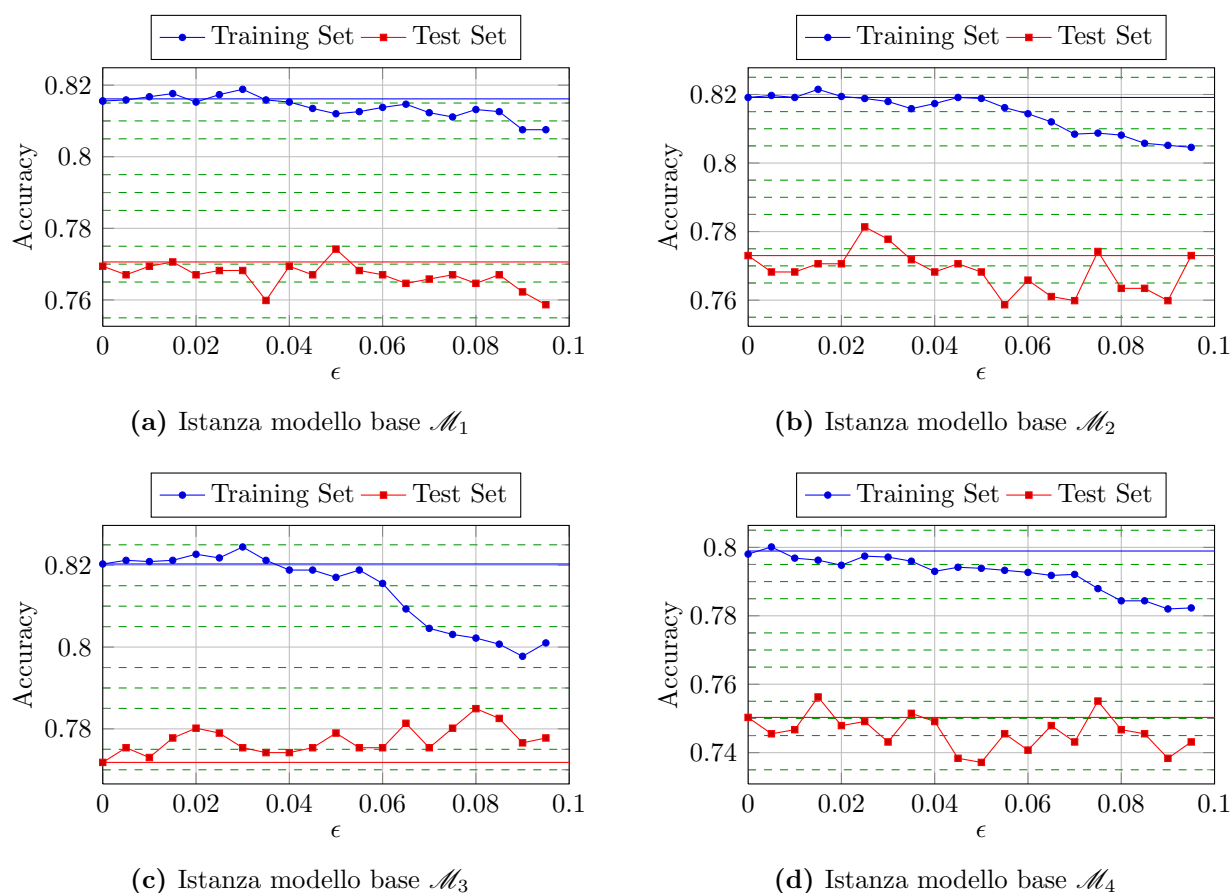


Figura 6.8: Risultati dei modelli addestrati, partendo dal modello base, al variare del cutoff ϵ . Le barre blu e rosse rappresentano la baseline dell'istanza per il training e test set, rispettivamente

ridotto creavano *noise* durante l'addestramento del modello.

Sebbene non sia mostrato nel grafo, si è notato un drastico peggioramento delle performance per $\epsilon \geq 0.1$. Questo è probabilmente dovuto al numero di atomi che complessivamente vengono rimossi. Come è possibile osservare in Fig. 6.9, il numero medio di atomi rimossi, in percentuale, segue un andamento lineare rispetto al cutoff. Quel grafico mostra che ad un $\epsilon = 0.095$ corrisponde una rimozione media del 60% degli atomi, sia in training che in test, nella formulazione del valore di uscita. Il grafico è mostrato per il training set e la singola istanza \mathcal{M}_1 , ma risultati del tutto analoghi sono stati riscontrati nel test set e in tutte le istanze testate.

I grafici in Fig. 6.10 mostrano la percentuale di atomi appartenenti ad un certo elemento che vengono rimossi al variare del cutoff. Tra le varie istanze si nota una certa differenza nel tipo di elementi rimossi, al variare del cutoff, segno che la distribuzione

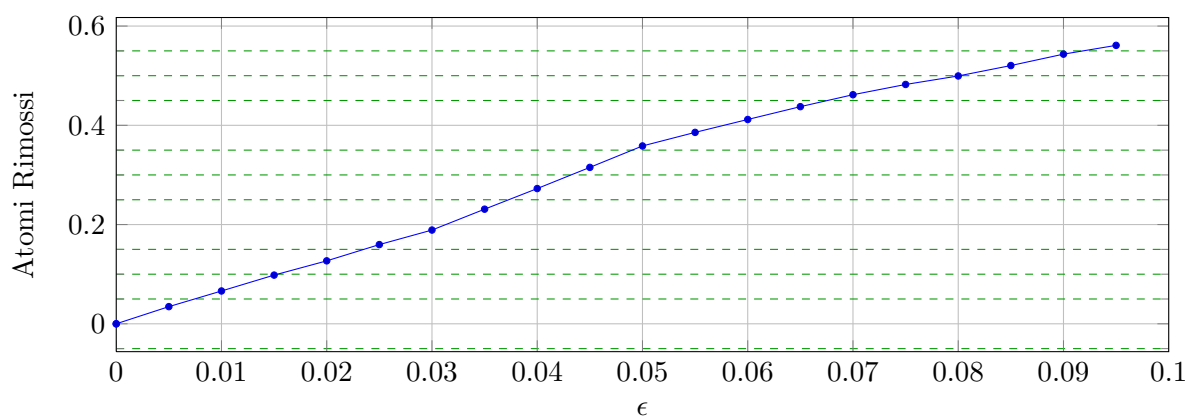


Figura 6.9: Percentuale, normalizzata, di atomi rimossi rispetto al cutoff nel training set.

dei contributi rispetto agli elementi non è sempre uniforme nelle istanze dei modelli base. Tuttavia, sono stati riscontrati alcuni tratti che accomunano le varie istanze dei modelli derivati, come il numero di atomi di idrogeno che venivano ignorati nella generazione del risultato, che in quasi tutte le istanze prodotte arrivavano a dimezzarsi per $\epsilon \approx 0.05$. In maniera analoga, anche se in forma minore, abbiamo osservato un fenomeno simile anche per gli atomi di carbonio. La cosa appare maggiormente evidente se normalizziamo i valori visti in precedenza, in funzione del numero massimo di atomi, come è visibile per il caso del modello \mathcal{M}_1 in Fig. 6.11).

Queste osservazioni suggeriscono che il modello NN_4G , mediante l'addestramento, abbia relegato gli atomi appartenenti a questi elementi, ad un semplice ruolo di complemento strutturale e non particolarmente rilevante ai fini della predizione. In questo senso, la rimozione di buona parte di questi atomi può giustificare l'incremento di performance osservato nell'addestramento delle istanze derivate.

La redistribuzione dei contributi ottenuta mediante l'eliminazione dei contributi minori ha permesso di eliminare, sia dalla rappresentazione grafica, che dalla formula visibile dall'esperto del dominio, gli atomi a cui era associato un contributo meno rilevante nel modello base, rendendone ancora più semplice la lettura da parte dell'esperto del dominio (come è visibile in Fig. 6.12).

Come ci aspettavamo, la redistribuzione dei contributi ha inoltre permesso di mettere in maggior risalto le strutture principali della molecola. In alcuni casi, come quello visibile in Fig. 6.13a la molecola presenta contributi misti, con gli idrogeni del gruppo aromatico a forte carattere negativo e il carbonio nel gruppo CH_3 positivo. Usando il sistema con il modello derivato, visibile in Fig. 6.13b, abbiamo una sostanziale

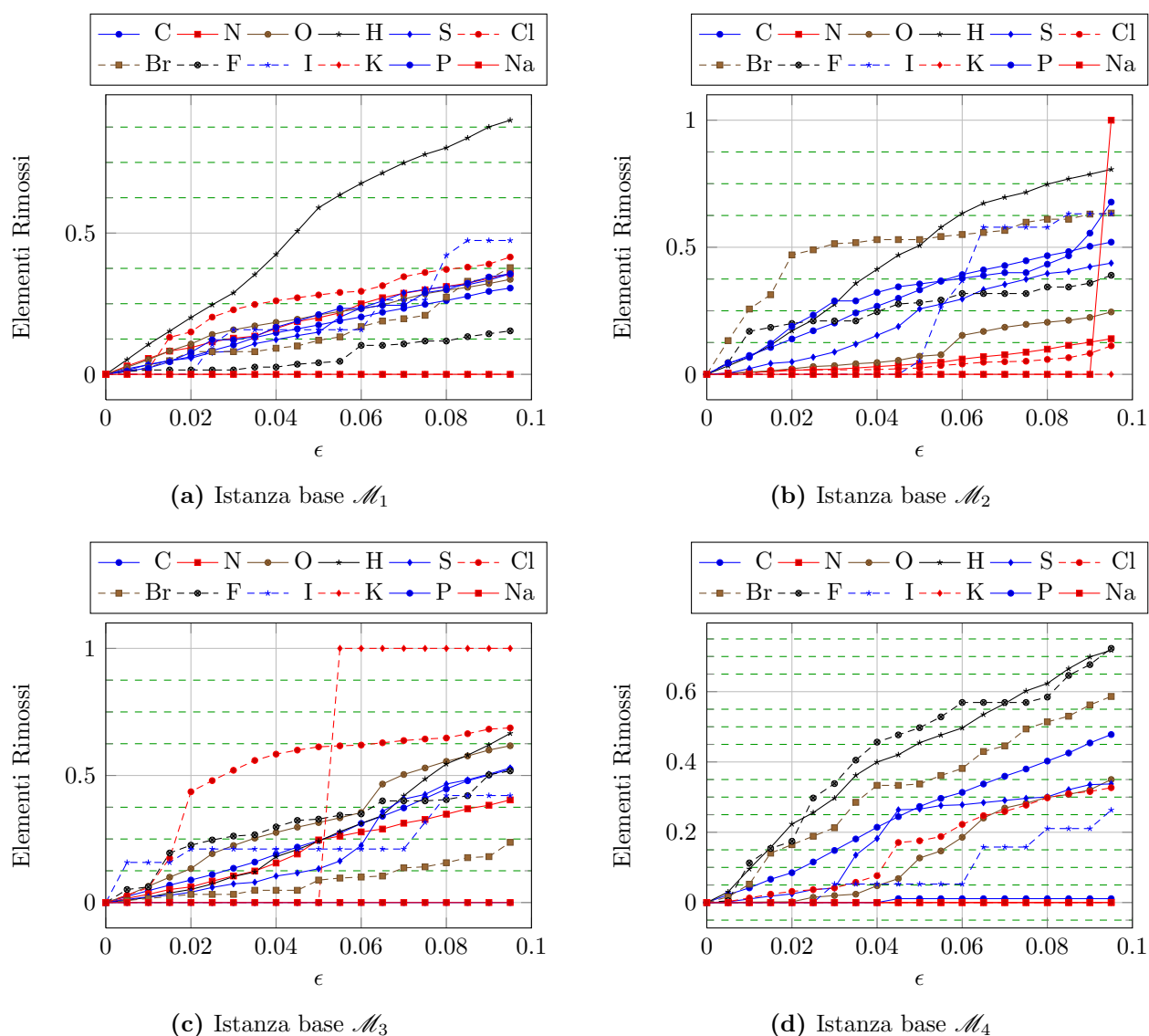


Figura 6.10: Percentuale (normalizzata) di atomi rimossi, divisi per elemento, rispetto alle istanze dei modelli derivati considerate.

neutralizzazione di tutti i componenti, tranne l'azoto appartenente alla SA 21.

Un altro esempio è rappresentato Fig. 6.14, ha permesso di dare maggior risalto a entrambe le SA presenti nella molecola, la SA 13 e 14. Come si vede dalla figura, in entrambi i casi il sistema di riduzione dei contributi minori ha eliminato i contributi dell'idrogeno e il carbonio (visibili nel dettaglio (c) e della figura) rendendo più evidente la parte centrale della struttura (visibile nel dettaglio (d) e (f)).

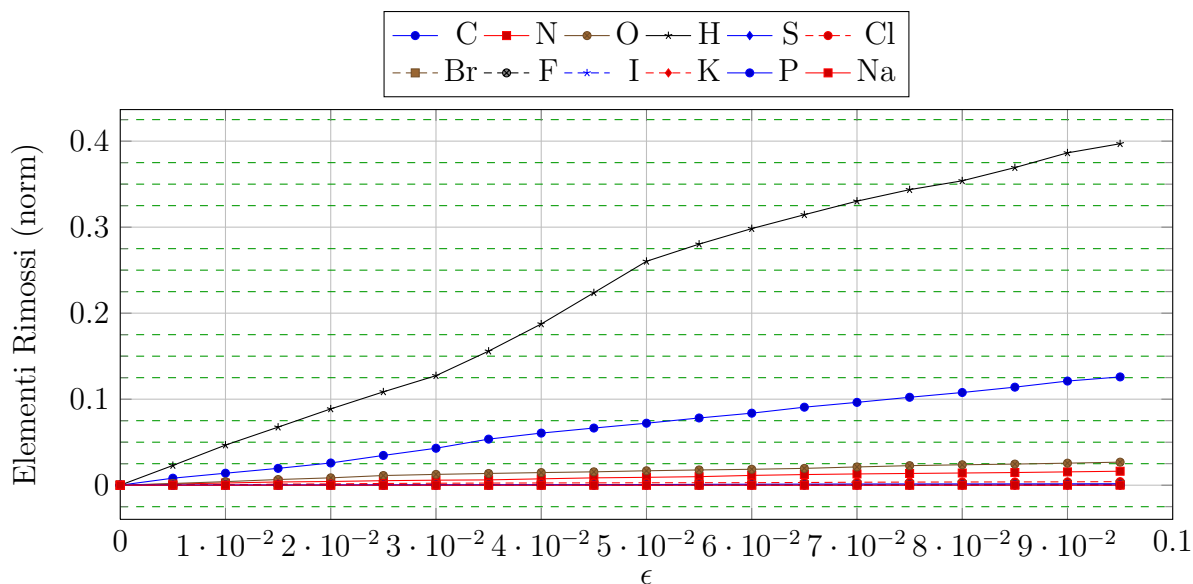
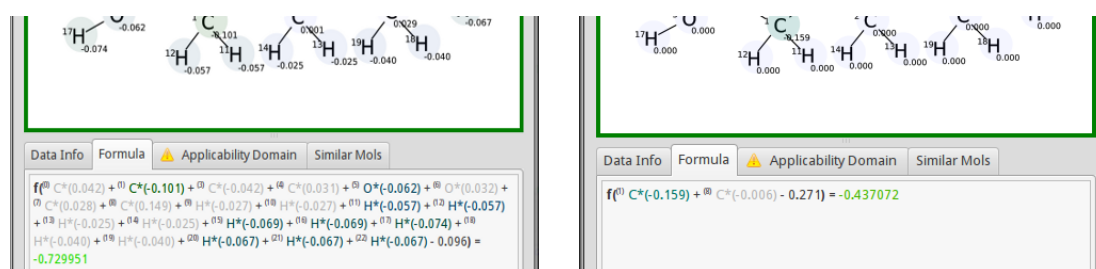


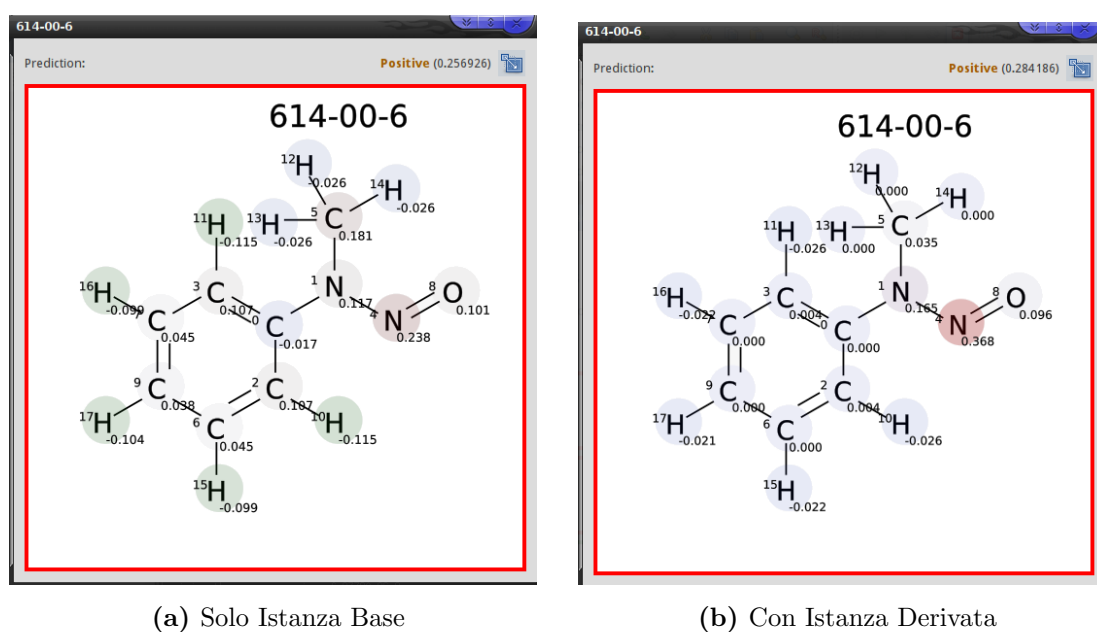
Figura 6.11: Percentuale degli atomi rimossi per elemento, normalizzati sul massimo numero di atomi, sul training set, al variare del cutoff ϵ



(a) Formula Ricavata dal Modello Base

(b) Formula Ricavata dal Sistema Binario

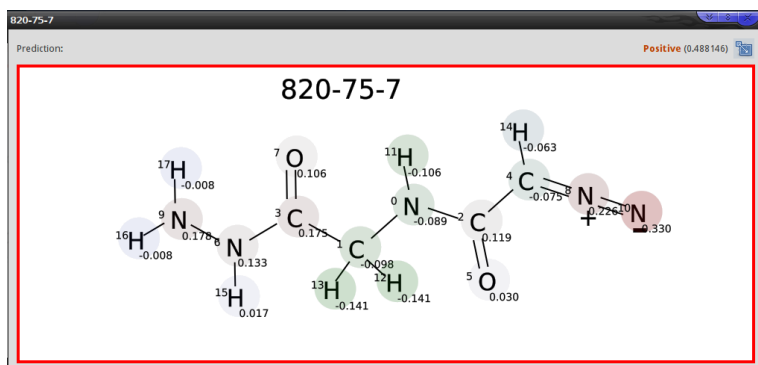
Figura 6.12: Dettaglio sulla formula dei contributi, nel caso del modello base e con l'ausilio del modello derivato, sulla stessa molecola.



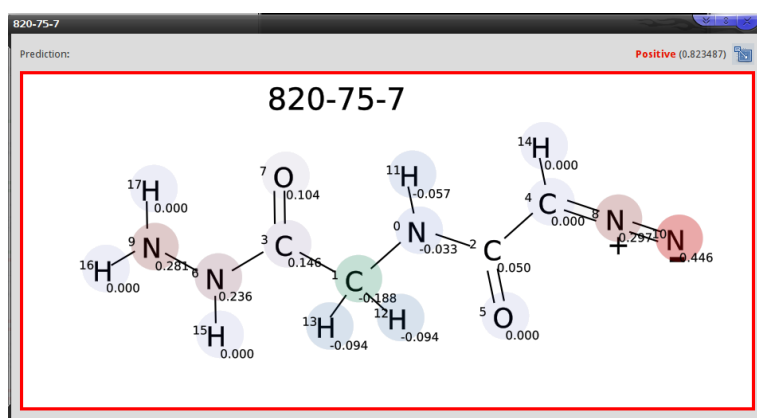
(a) Solo Istanza Base

(b) Con Istanza Derivata

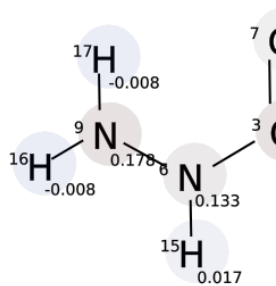
Figura 6.13: Esempio di SA evidenziata dalla redistribuzione dei contributi, relativamente al modello base \mathcal{M}_3 e al sistema di riduzione dei contributi minori con l'istanza derivata.



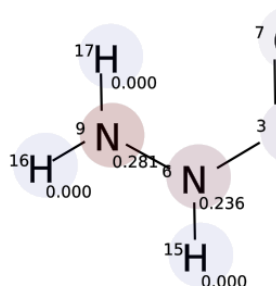
(a) Solo Istanza Base (IB)



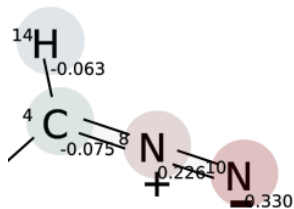
(b) Con Istanza Derivata (ID)



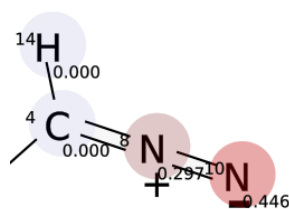
(c) Dettaglio SA 13 (IB)



(d) Dettaglio SA 13 (ID)



(e) Dettaglio SA 14 (IB)



(f) Dettaglio SA 14 (ID)

Figura 6.14: Esempio di entrambe le SA presenti nella molecola, evidenziate dalla redistribuzione dei contributi.

Capitolo 7

Conclusioni

Il lavoro di questa tesi ha prodotto un tool innovativo per la predizione molecolare, basato sull'analisi di grafi. Il tool si basa su librerie e strumenti di addestramento e analisi di modelli *NN4G* e permette di mostrare, all'esperto del dominio, le sotto-strutture molecolari che concorrono alla formulazione di una predizione positiva o negativa.

Il primo obiettivo realizzato è stato dunque la creazione di un sistema completo per l'addestramento di modelli per grafi, di tipo *NN4G*, e di analisi post training. Questo sistema, comprendente due tool e due librerie, consente l'uso di molteplici algoritmi di apprendimento ed è stato progettato usando criteri di estendibilità e facilità d'uso.

Questa prima parte del lavoro ci ha permesso di sperimentare nuovi algoritmi di apprendimento per il modello *NN4G*, come l'algoritmo costruttivo CR e il metodo ridge regression per il layer di uscita del modello.

L'utilizzo del sistema di costruzione del modello, basato sulla libreria *felix* e il tool *cougar*, ha permesso di effettuare sperimentazioni sistematiche, agevolando la gestione dei risultati. La libreria *felix* supporta l'addestramento sia di modelli per dati flat, come il *MLP*, che per grafi, come il *NN4G*, e la sua progettazione modulare ha permesso di sviluppare agilmente diversi algoritmi di apprendimento. Il tool *felix-reader*, derivato dalla libreria *qfelix*, estensione grafica di *felix*, ha permesso di analizzare le istanze addestrate, mediante grafici dell'andamento del training, informazioni sul modello e sul descrittore usato per generarlo, e la visualizzazione del *log* generato. La capacità di questo tool di generare report in PDF con gran parte di queste informazioni ha fornito un utile supporto nella selezione degli iperparametri migliori.

Sebbene le varianti degli algoritmi di apprendimento introdotte non abbiano apportato dei miglioramenti significativi sui benchmark sperimentati, il loro utilizzo testimonia la completezza della libreria *felix* e le sue capacità di estensione.

Le librerie *felix* e *gfelix*, parte del sistema di costruzione del modello, sono state la base del successivo sviluppo di un tool completo di predizione di proprietà molecolari. Il tool, a differenza di altri strumenti di predizione molecolare, sfrutta le caratteristiche della *NN4G* per effettuare la predizione senza passare dai descrittori, ma convertendo la molecola in un grafo e usare un'istanza addestrata del modello *NN4G* per ottenere il responso.

Questo approccio si differenzia dagli approcci a descrittori in quanto effettua la predizione basandosi direttamente sulla struttura molecolare, senza passare da una selezione di un sottoinsieme significativo di descrittori molecolari, variabile a seconda dell'endpoint da predire, i quali possono codificare solo un'informazione parziale della molecola.

Il predittore sviluppato è in grado di effettuare predizioni su un endpoint definito, fornendo all'esperto informazioni riguardo il dominio applicativo del modello, ovvero un insieme di caratteristiche delle molecole usate per l'addestramento. Queste informazioni permettono all'esperto di valutare l'affidabilità della predizione, in relazione alla molecola analizzata.

Il predittore sviluppato ha la caratteristica innovativa di sfruttare una riformulazione della funzione di uscita del modello *NN4G* per rilevare le sotto-strutture molecolari che maggiormente hanno contribuito alla formulazione della predizione. Questo ha permesso di far emergere questa informazione fino all'esperto del dominio, sia come rappresentazione grafica sovrapposta alla rappresentazione molecolare, sia sottoforma di formula esplicita dell'uscita del modello. La possibilità di mostrare quali sotto-strutture molecolari hanno contribuito maggiormente a formulare, ad esempio, un responso di tossicità, permette all'esperto di indagare sulle componenti molecolari maggiormente rilevanti. Questo aspetto è stato sviluppato seguendo le necessità di trasparenza del modello espresse nelle direttive della norma REACH.

Analizzando diverse istanze del modello *NN4G* addestrate su un dataset con endpoint di mutagenicità, abbiamo osservato una corrispondenza tra sotto-strutture fortemente positive e alcune structural alerts note in letteratura. È importante precisare che questo risultato è emerso senza che il modello abbia avuto alcuna informazione specifica sulle structural alerts.

Effettuando diversi esperimenti, abbiamo notato che il modello distribuiva molti contributi di modulo ridotto su molti vertici, spesso corrispondenti ad elementi specifici, come il carbonio e l'idrogeno. Questo fenomeno aveva l'inconveniente principale

di disperdere i contributi, ad esempio in atomi di carbonio o idrogeno, presenti in numero elevato nelle molecole analizzate, rendendo meno riconoscibili le sotto-strutture molecolari caratterizzanti per l'endpoint da predire.

Per ovviare a questo inconveniente, è stato sviluppato un sistema per far attribuire un contributo maggiore alle sotto-strutture più rilevanti per l'endpoint usato.

A questo scopo è stata modificata la formulazione del modello NN_4G in modo che escluda, dalla formulazione della predizione, contributi ritenuti poco rilevanti da un'altra istanza NN_4G . Questo sistema di riduzione dei contributi minori utilizza un'istanza addestrata di NN_4G per selezionare i contributi più rilevanti all'interno di un grafo, mentre una seconda istanza di NN_4G viene addestrata usando solo i contributi selezionati come rilevanti, in base ad una soglia definita, dalla prima istanza.

Incorporando questa variante del modello NN_4G nel predittore abbiamo riscontrato un miglioramento sotto diversi aspetti. In primo luogo, l'eliminazione dei contributi minori ha portato ad un accentramento dei contributi in sotto-strutture maggiormente definite all'interno della molecola, e potenzialmente più critiche per l'endpoint predetto. Inoltre l'eliminazione dei contributi minori ha, in molti casi, ridotto drasticamente il numero di coefficienti della formula esplicita della funzione di uscita visibile dall'esperto, rendendola più semplice e chiara. Per finire, l'eliminazione della parte di informazioni strutturali poco influente ha portato, in diversi casi, ad un aumento di performance del sistema di riduzione dei contributi minori, rispetto all'uso del solo modello base.

Sia il sistema di training, che il predittore, offrono diversi spunti per sviluppi futuri. Ad esempio è possibile sfruttare le capacità di estendibilità di *felix* per ideare nuove varianti del NN_4G , per creare modelli del tutto nuovi o implementare nuovi algoritmi di apprendimento per i modelli presenti.

Riguardo il modello NN_4G è possibile proseguire la strada tracciata da questo lavoro, ad esempio implementando un sistema che iteri il processo di training basato sui contributi, eliminando progressivamente i contributi ritenuti meno rilevanti fino ad uno stop criteria fissato.

Sperimentalmente è stato osservato che alcune istanze del modello NN_4G sono molto concordi nel riconoscere alcune structural alerts specifiche. È possibile sfruttare questa particolarità per usare un comitato che selezioni automaticamente alcune sotto-strutture su cui ci sia un forte consenso tra diversi modelli.

Infine, sebbene in questo progetto ci siamo concentrati sullo sviluppo di modelli e un tool per la predizione tossicologica, di fatto il predittore può essere facilmente

riadattato per l'utilizzo in altri campi dove i dati possono essere modellati per mezzo di grafi e dove l'individuazione di sotto-strutture critiche nella formulazione di una predizione risulti essere un aspetto cruciale.

Appendice A

Formato SDF

```
benzene
ACD/Labs0812062058

6 6 0 0 0 0 0 0 0 0 0 1 V2000
  1.9050 -0.7932  0.0000 C  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  1.9050 -2.1232  0.0000 C  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0.7531 -0.1282  0.0000 C  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0.7531 -2.7882  0.0000 C  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 -0.3987 -0.7932  0.0000 C  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 -0.3987 -2.1232  0.0000 C  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2  1  1  0  0  0  0
3  1  2  0  0  0  0
4  2  2  0  0  0  0
5  3  1  0  0  0  0
6  4  1  0  0  0  0
6  5  2  0  0  0  0
M  END
$$$$
```

Il cui formato è composto¹ da:

Intestazione (linee 1-3) comprendente:

- il nome della molecola (benzene) (linea 1)
- le informazioni su Utente/Programma/Data/ecc (linea 2)
- il commento (linea 3)

¹le linee sono relative a questo esempio, di fatto il Ctab può contenere un numero di linee, per il blocco degli atomi e dei legami, arbitrario.

Tabella delle Connessioni (Ctab) (linee 4-17) contenente:

- la linea dei contatori: 6 atomi, 6 legami, . . . , V2000 standard (linea 4)
- il blocco degli Atomi (1 linea per ogni atomo): x, y, z, elemento, ecc. (linee 5-10)
- il blocco dei Legami (1 linea per ogni legame): primo atomo, secondo atomo, tipo, ecc. (linee 11-16)
- il blocco delle Proprietà (linea 17)

Appendice B

Formato Gph Esteso (Gph+)

Il formato gph è una codifica che permette di salvare su memoria permanente dati strutturati come grafi e loro sottoclassi, come DPAG e alberi radicati.

In questo lavoro si è adottato il formato gph come formato standard per la libreria felix e come formato target per la conversione dei dataset molecolari, codificati in SDF. Per gli scopi del progetto si è scelto di estendere la notazione classica del formato, effettuando modifiche minori, come il cambiamento della label *TreeNum* in quella più generica *GraphNum*, assieme all'aggiunta di nuove funzionalità espressive, quali:

- La possibilità di aggiungere features specifiche dei singoli vertici. Il formato gph standard permette l'utilizzo di sole features globali legate alle etichette impostate nell'header, mentre il formato esteso prevede l'aggiunta di *VertexSpecificFeatures* (vedere App. B.4);
- La possibilità di aggiungere le features degli archi, possibilità non contemplata nel formato standard;

Queste aggiunte vanno ad ampliare il numero di informazioni che possono essere codificate con il formato, e in questa appendice presentiamo il formato esteso completo. La possibilità di aggiungere feature associate agli archi era legata ad alcuni sviluppi non inseriti in questo progetto.

Nelle definizioni delle sezioni successive vengono usati i seguenti simboli terminali:

int numero intero, usato in genere per gli indici dei vertici;

real numero reale, usato in genere per le features;

chars sequenza di caratteri, usato in genere per le etichette descrittive;

empty la stringa vuota;

Dove non e' specificato diversamente, tutti gli elementi delle regole grammaticali sono da considerarsi separati da spazi, nelle relative produzioni. Alcune regole (*VertexGlobalAriety*, *VertexGlobalIdx*, ...) sono ridondanti, e sono state inserite solo per semplificare la lettura della grammatica.

B.1 Definizione del Documento

Il documento gph si presenta in due parti:

- Un *header* contenente le informazioni sulle etichette dei vertici, assieme al valore delle features globali¹;
- Il *corpo* del documento, contenente una sequenza di grafi;

$$\langle GphDocument \rangle ::= \langle Header \rangle \mid \langle Graphs \rangle$$

$$\langle Graphs \rangle ::= \langle Graph \rangle \mid \langle Graph \rangle \backslash n \langle Graphs \rangle$$

B.2 Definizione dell'Intestazione

L'intestazione, o *header*, contiene tutte le informazioni di carattere globale, come il numero di grafi presenti nel documento, il grado massimo, e le features globali associate alle diverse etichette.

$$\langle Header \rangle ::= \text{'SymbolNum'} \langle int \rangle \backslash n \text{'LabelDim'} \langle int \rangle \backslash n \langle VerticesGlobalInfos \rangle \backslash n \text{'GraphNum'} \langle int \rangle \backslash n \text{'MaxAriety'} \langle int \rangle \backslash n$$

$$\langle VerticesGlobalInfos \rangle ::= \langle VertexGlobalInfos \rangle \mid \langle VertexGlobalInfos \rangle \langle VerticesGlobalInfos \rangle$$

$$\langle VertexGlobalInfos \rangle ::= \backslash t \langle VertexGlobalIdx \rangle \langle VertexGlobalLabel \rangle \langle VertexGlobalAriety \rangle \langle GlobalFeats \rangle \backslash n$$

$$\langle VertexGlobalIdx \rangle ::= \langle int \rangle$$

$$\langle VertexGlobalLabel \rangle ::= \langle chars \rangle$$

$$\langle VertexGlobalAriety \rangle ::= \langle int \rangle$$

$$\langle GlobalFeats \rangle ::= \langle real \rangle \mid \langle real \rangle \langle GlobalFeats \rangle$$

¹In questo lavoro le etichette rappresentavano gli elementi atomici e le features globali la relativa codifica loK

B.3 Grammatica del Grafo

La codifica del grafo parte con alcune informazioni utili ad identificarlo rapidamente, come il nome, i valori target separati da uno spazio² e prosegue con la codifica di tutti i vertici del grafo.

$$\langle \text{Graph} \rangle ::= \text{'Name'} \langle \text{chars} \rangle \text{'\n'} \text{'GraphDim'} \langle \text{int} \rangle \text{'\n'} \text{'Target'} \langle \text{Target} \rangle \text{'\n'} \langle \text{VerticesGraphInfo} \rangle \text{'\n'}$$

$$\langle \text{Target} \rangle ::= \langle \text{real} \rangle \mid \langle \text{real} \rangle \langle \text{Target} \rangle$$

$$\langle \text{VerticesGraphInfo} \rangle ::= \langle \text{VertexGraphInfo} \rangle \mid \langle \text{VertexGraphInfo} \rangle \langle \text{VerticesGraphInfo} \rangle$$

B.4 Grammatica del Vertice

Il singolo vertice di un grafo contiene in una sola riga le informazioni riguardo:

- Un valore progressivo che rappresenta il suo indice;
- L'indice dei vicini, fino a riempire il grado massimo, riempiendo i vicini “mancanti” con valori nulli “-1”;
- L'etichetta globale associata al vertice;
- Le features associate agli archi, se presenti;
- Le features locali del vertice, se presenti;

$$\langle \text{VertexGraphInfo} \rangle ::= \langle \text{id} \rangle \langle \text{Neigh} \rangle \langle \text{VertexGlobalLabel} \rangle \langle \text{VertexGlobalIndx} \rangle \langle \text{VertexSpecificFeatures} \rangle \text{'\n'}$$

$$\langle \text{VertexSpecificFeatures} \rangle ::= \langle \text{empty} \rangle$$

$$\mid \text{'\n'} \langle \text{EdgeFeats} \rangle$$

$$\mid \text{'\n'} \langle \text{VertexFeats} \rangle$$

$$\mid \text{'\n'} \langle \text{EdgeFeats} \rangle \text{'\n'} \langle \text{VertexFeats} \rangle$$

$$\langle \text{id} \rangle ::= \langle \text{int} \rangle$$

$$\langle \text{Neigh} \rangle ::= \langle \text{int} \rangle \mid \langle \text{int} \rangle \langle \text{Neigh} \rangle$$

$$\langle \text{EdgeFeats} \rangle ::= \text{'-'} \mid \text{'('} \langle \text{EF} \rangle \text{'\n'} \mid \text{'('} \langle \text{EF} \rangle \text{'\n'} \langle \text{EdgeFeats} \rangle$$

$$\langle \text{EF} \rangle ::= \langle \text{real} \rangle \mid \langle \text{real} \rangle \langle \text{EF} \rangle$$

$$\langle \text{VertexFeats} \rangle ::= \langle \text{real} \rangle \mid \langle \text{real} \rangle \langle \text{VertexFeats} \rangle$$

B.5 Esempi

Qui di seguito vengono illustrati due esempi pratici di formato gph+.

²Nel lavoro proposto abbiamo sempre usato un singolo valore target

B.5.1 Esempio di Intestazione

SymbolNum 14

LabelDim 16

```

0 C 3 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 N 3 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
2 O 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
3 H 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
4 S 5 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
5 Cl 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
6 Br 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
7 F 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
8 I 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
9 K 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
10 P 5 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
11 Na 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
12 Li 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
13 Ca 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

```

GraphNum 4204

MaxAriety 6

B.5.2 Esempio di Grafo

Name 95-94-3

GraphDim 12

Target -1

```

0 1 2 3 -1 -1 -1 C 0 ; (0 1) (0 1) (0 0) - - - - : 0.0 0.53
1 0 4 5 -1 -1 -1 C 0 ; (0 1) (0 1) (0 0) - - - - : 0.0 0.234
2 0 6 10 -1 -1 -1 C 0 ; (0 1) (0 1) (0 0) - - - - : 0.0 0.0
3 0 -1 -1 -1 -1 -1 Cl 5 ; (0 0) - - - - : 0.0 0.16
4 1 7 11 -1 -1 -1 C 0 ; (0 1) (0 1) (0 0) - - - - : 0.47 0.0
5 1 -1 -1 -1 -1 -1 Cl 5 ; (0 0) - - - - : 0.0 0.0
6 2 8 7 -1 -1 -1 C 0 ; (0 1) (0 0) (0 1) - - - - : 0.0 0.0
7 4 9 6 -1 -1 -1 C 0 ; (0 1) (0 0) (0 1) - - - - : 0.0 0.34
8 6 -1 -1 -1 -1 -1 Cl 5 ; (0 0) - - - - : 0.0 0.0
9 7 -1 -1 -1 -1 -1 Cl 5 ; (0 0) - - - - : 0.38 0.0
10 2 -1 -1 -1 -1 -1 H 3 ; (0 0) - - - - : 0.0 0.0
11 4 -1 -1 -1 -1 -1 H 3 ; (0 0) - - - - : 0.0 0.27

```

Appendice C

Esempi di Files

C.1 Esempio di File del Modello

Questo è un esempio di modello NN4G salvato su file. La maggior parte delle informazioni contenute nel file sono necessarie a ricreare esattamente il modello salvato, in modo da eseguire predizioni o ulteriori esperimenti. Tra le informazioni contenute, ve ne sono alcune relative al training set, come il path contenuto nell'attributo *trainedOn*, e il valore massimo e minimo dei contributi ottenuti su quel dataset.

Il modello in esempio consiste di tre unità nascoste e una unità di uscita. I collegamenti tra le unità sono visibili nel tag *hiddenWeights*, mentre in *inputWeights* sono contenuti i valori dei pesi di input. La funzione d'attivazione usata dall'unità, assieme ai suoi parametri, è espressa nel tag *activationFunction*, mentre nell'attributo *saturationLevel* viene segnalato il livello di saturazione raggiunto da quell'unità durante l'addestramento.

```
<?xml version="1.0"?>

<Model name="NN4G" trainedOn="/home/barontil/tesi/data/bursi/bursi.gph
">
  <weights minWeightValue="-0.5" maxWeightValue="0.5"/>
  <normalization type="Normalized"/>
  <maxVertexNumber number="417"/>
  <contributions>
    <maxContribution name="133561-39-4: 4" value="0.692368"/>
    <minContribution name="80734-02-7: 30" value="-0.569537"/>
  </contributions>
```

```
<NN4GUnit name="Candidate #0:2" type="Hidden" saturationLevel="
  0.980434">
<activationFunction name="Sigmoidal" min="-1" max="1" slope="0.4"/>
<inputWeights>
3.7729,-3.22652,6.64258,6.41053,16.9392,1.59806,1.18479,-0.755965,
7.05397,-2.40019,-0.372821,1.64104,-3.468,0.782916,6.19697,11.7393,-8.20156
  </inputWeights>
<bias>-2.80814
</bias>
</NN4GUnit>
<NN4GUnit name="Candidate #1:3" type="Hidden" saturationLevel="
  0.999922">
<activationFunction name="Sigmoidal" min="-1" max="1" slope="0.4"/>
<inputWeights>
-0.0324791,5.81862,0.100559,-0.112426,3.38586,0.596681,0.478867,1.11495,
0.0553254,1.56433,-0.412405,-0.592605,-0.102662,0.833191,10.9647,2.42677,12.1916
  </inputWeights>
<hiddenWeights>(1.94956,Candidate #0:2)
</hiddenWeights>
<bias>10.7809
</bias>
</NN4GUnit>
<NN4GUnit name="Candidate #2:2" type="Hidden" saturationLevel="0.7347"
  >
<activationFunction name="Sigmoidal" min="-1" max="1" slope="0.4"/>
<inputWeights>
7.14142,-1.44644,8.19724,-1.16645,-2.80486,0.186277,0.215389,29.7969,
-0.726816,8.21934,-0.267049,-0.856916,18.1097,21.5542,-5.49237,3.38262,24.8982
  </inputWeights>
<hiddenWeights>(10.1344,Candidate #0:2);(-5.93275,Candidate #1:3)
</hiddenWeights>
<bias>-2.78997
</bias>
</NN4GUnit>
<NN4GUnit name="Candidate #3:1" type="Hidden" saturationLevel="
  0.919242">
<activationFunction name="Sigmoidal" min="-1" max="1" slope="0.4"/>
<inputWeights>
4.45211,29.1307,5.39407,-0.46784,24.0281,-0.0189667,-0.528488,2.66094,
-0.292564,-2.12008,-0.479483,-1.3016,0.952678,-0.317546,8.55076,-2.70928,0.22952
```

```

    </inputWeights>
<hiddenWeights>(1.16725,Candidate #0:2);(-4.23228,Candidate #1:3)
    ;(7.76084,Candidate #2:2)
</hiddenWeights>
<bias>-2.66024
</bias>
</NN4GUnit>
<NN4GUnit name="Output #0" type="Output" saturationLevel="0">
<activationFunction name="Linear"/>
<hiddenWeights>(-58.6647,Candidate #0:2);(-20.7776,Candidate #1:3)
    ;(87.7393,Candidate #2:2);(19.7523,Candidate #3:1)
</hiddenWeights>
<bias>-0.134534
</bias>
</NN4GUnit>
</Model>

```

C.2 Esempio Completo di Training

Il seguente codice rappresenta un esempio completo di training eseguito usando le funzionalità della libreria *felix*.

Listing C.1: Esempio di training completo di un modello

```

try{
    /// load the task descriptor
    TaskDescriptor* descriptor = new TaskDescriptor(argv[1]);
    /// Make the abstract factories
    DatasetFactory dsf;
    ModelsFactory mfactory;
    LearningAlgorithmsFactory lfactory;
    /// load the datasets
    iDataset* training_set;
    iDataset* validation_set;
    vector<vector<iDataset*> > mainDatasets = dsf.makeConcreteInstances(
        descriptor->dataParameters);
    unsigned int nFolds=mainDatasets[0].size();
    iNeuralNet* model = NULL;
    iLearningAlgorithm* learning = NULL;
    Result* final_result = NULL;

```

```
if(nFolds==1){
    /// this is an Hold Out separation
    training_set = mainDatasets[0][0];
    validation_set = mainDatasets[1][0];
    /// make the model
    model = mfactory.makeConcreteInstance(descriptor->modelParameters,
        training_set);
    /// make the learning algorithm
    learning = lfactory.makeConcreteInstance(descriptor->
        learningAlgorithmParameters,training_set,validation_set,model);
    /// perform the training
    learning->start_training();
    /// keep the results
    final_result=learning->results;
}else{
    /// this is a CV separation
    vector<felix::Result*> partial_results;
    for (unsigned int i=0;i<nFolds;++i){
        /// delete the previous learning
        delete learning;
        /// reset the seed at the original one
        descriptor->resetRandomToCurrentSeed();
        /// get the i-th training and validation set
        training_set = mainDatasets[0][i];
        validation_set = mainDatasets[1][i];
        /// make the model
        model = mfactory.makeConcreteInstance(descriptor->
            modelParameters,training_set);
        /// make the learning algorithm
        learning = lfactory.makeConcreteInstance(descriptor->
            learningAlgorithmParameters,training_set,validation_set,model
        );
        /// perform the training
        learning->start_training();
        /// save a result's copy
        partial_results.push_back(learning->results->clone());
    }
    /// make the final result
    final_result = new felix::ResultsCVCCollector(partial_results);
    /// delete all the partial results
    for (unsigned int i=0;i<partial_results.size();++i){
```



```

        if(partial_results[i]->model != final_result->model)
&delete partial_result[i]->model;
        delete partial_results[i];
    }
}
/// print the final results
string training_str, validation_str;
final_results->lastResults2str(training_str, validation_str);
cout &<< "Final Results:" << endl
&<< "Training: " << training_str << endl
&<< "Validation: " << validation_str << endl;
/// save the results
string resultsPath=final_result->save(descriptor);
cout<< "Results saved in: "<<resultsPath<<endl;
/// Delete all the data structures
delete descriptor;
delete model;
delete learning;
for (unsigned int i=0;i<mainDatasets.size();++i){
    for (unsigned int j=0;j<mainDatasets[i].size();++j){
        delete mainDatasets[i][j];
    }
}
}catch(string e){
    cerr<<"ERROR: "<<e<<endl;
}
}

```

C.3 Esempio di File del Descrittore

Il descrittore presentato in questo esempio descrive l'addestramento di un modello NN_4G , con singolo trial, sul dataset *bursi.gph* usando una separazione Hold Out.

L'algoritmo di apprendimento da usare è il *Cascade Correlation* il quale usa un *QuickPropagation* per le 4 unità nascoste del pool, con un'*Incremental Strategy* come stop criteria locale, e il calcolo della pseudoinversa in uscita.

Il processo termina con il raggiungimento dell'85% di accuracy, o delle 200 unità nascoste, e il pacchetto dei risultati viene salvato sotto il path presente nell'attributo *mainPath* del tag *results*.

```
<?xml version="1.0" ?>
```

```
<TaskDescriptor name="bursiCCwd01" version="1.14">
<Task trials="1"/>
<Data name="GraphDataset" type="gph" path="/home/barontil/tesi/data/
  bursi/bursi.gph" shuffle="false">
<separation type="HoldOut" validationCut=".8"/>
<results mainPath="/home/barontil/tesi/results/bursi/"/>
</Data>
<Model name="NN4G">
<weights minWeightValue="-0.5" maxWeightValue="0.5"/>
<outputActivationFunction name="Linear"/>
<hiddenActivationFunction name="Sigmoidal" max="1" min="-1" slope="0.4
  "/>
<normalization type="Normalized"/>
</Model>
<Learning name="CascadeCorrelation" verbose="True" saveExtendedLogging
  ="True">
<pool size="4"/>
<candidateLearning name="QuickPropagation" learningRate="0.1"
  weightDecay="0.01" normalizeDerivate="True" saveExtendedLogging="
  True">
<stopCriteria name="maxIterationReached" maxIteration="200"/>
<stopCriteria name="minIterations" minIteration="20" tick="10"
  upperBound="200"/>
</candidateLearning>
<outputLearning name="Pseudoinverse" lambda="0.0" saveExtendedLogging=
  "False"/>
<stopCriteria name="maxIterationReached" maxIteration="200"/>
<stopCriteria name="maxTrainingResult" maxResult="0.85"/>
</Learning>
</TaskDescriptor>
```

C.3.1 Esempio di File Grid

In questo file grid di esempio prevede la generazione di diversi descrittori a partire da quello espresso dal valore dell'attributo *path* del tag *taskDescriptor*.

I descrittori che verranno generati, saranno corrispondenti alle permutazioni ottenute dalle variazioni su:

- learning rate η dell'algoritmo per l'unità candidata, variabile da 0.01 a 0.05 con intervallo di 0.02;
- weight decay λ dell'algoritmo per l'unità candidata, variabile da 0.0 a 0.2 con intervallo di 0.05;
- incremental strategy dell'algoritmo per l'unità candidata, i cui scatti vanno da 10 a 30 con intervallo di 15;
- parametro λ del calcolo della pseudoinversa per l'unità di uscita, variabile da 0.0 a 0.1 con intervallo di 0.05;
- risultato¹ massimo da raggiungere, come stop criteria globale, variabile da 0.7 a 0.9 con intervallo di 0.05

Questo grid descriptor prevede che il training, eseguito ad esempio dal tool cougar, debba essere fatto con una Double-CV con 10 folds esterne.

```
<?xml version="1.0"?>

<GridDescriptor version="1.0">
<taskDescriptor path="/home/barontil/tesi/default_descriptor.cfg"/>
<Data>
<separation type="CrossValidation" nFolds="10" altPicking="True"/>
</Data>

<Learning>
<candidateLearning>

<learningRate from="0.01" to="0.05" tick="0.02"/>
<weightDecay from="0.0" to="0.2" tick="0.05"/>

<stopCriteria name="minIterations">
<tick from="10" to="30" tick="15"/>
</stopCriteria>

</candidateLearning>

<outputLearning>
```

¹Il tipo di risultato è in questo caso l'accuracy, ma in generale è dipendente dal task in questione e non è deducibile dal descrittore del grid.

```
<lambda from="0.0" to="0.1" tick="0.05"/>
</outputLearning>

<stopCriteria name="maxTrainingResult">
<maxResult from=".7" to=".9" tick=".05"/>
</stopCriteria>
</Learning>
</GridDescriptor>
```

Appendice D

Lista Completa dei Risultati

In questa appendice presentiamo la lista completa dei risultati sui dataset usati per il collaudo del sistema. I risultati sono rappresentati mediante etichette in accordo alle definizioni date in Sez. [2.1](#).

D.1 Alcani

In questa sezione verranno mostrati i risultati, in forma estesa, dei test sul dataset degli alcani.

D.1.1 Cascade Correlation

I risultati della griglia sono visibili in Tab. [D.1](#), mentre i risultati delle migliori configurazioni in training e test, sulle fold esterne della DCV, sono visibili rispettivamente in Tab. [D.3](#) e Tab. [D.4](#).

Weight Decay	Intervallo	λ	MaE_{tr}	MaE_{val}
0	10	0	0.0132782	0.0176109
0	10	0.05	0.0386705	0.0423986
0	10	0.1	0.0493344	0.0521369
0	30	0	0.0133554	0.0183067
0	30	0.05	0.0325871	0.0387731
0	30	0.1	0.0431564	0.0466892
5e-05	10	0	0.0135267	0.018784
5e-05	10	0.05	0.0391407	0.0429237
5e-05	10	0.1	0.048019	0.0512279
5e-05	30	0	0.0136141	0.0186015
5e-05	30	0.05	0.0310697	0.0372884
5e-05	30	0.1	0.0416094	0.04551
0.0001	10	0	0.0133035	0.0180702
0.0001	10	0.05	0.0383363	0.0416274
0.0001	10	0.1	0.0496852	0.052375
0.0001	30	0	0.0132998	0.0184342
0.0001	30	0.05	0.0315348	0.0363504
0.0001	30	0.1	0.0427935	0.0461996

Tabella D.1: Risultati alcani CC mediati sul loop interno della DCV, delle configurazioni a griglia

Weight Decay	Intervallo	λ	Unità	MaE	$MaxAbsErr$	R	S
0	10	0	23	0.0121569	0.0414067	0.999362	0.0155182
0.0001	30	0	20	0.0151057	0.0463075	0.999151	0.0192508
0	10	0	25	0.0138119	0.0499931	0.999341	0.0175364
0	30	0	22	0.0153868	0.0491123	0.999238	0.01921
0	10	0	24	0.0140076	0.0494452	0.999347	0.0178137
0.0001	10	0	24	0.012518	0.0439342	0.999499	0.015616
0	10	0	17	0.0158719	0.0482089	0.999246	0.019361
0.0001	30	0	20	0.0158913	0.04969	0.999207	0.019879
0.0001	10	0	26	0.0135492	0.0499696	0.999424	0.0170219
0.0001	30	0	17	0.013672	0.0480365	0.999386	0.0176099
Media			21.8	0.0141	0.0476	0.999	0.017
Deviazione Std			3.1902	0.0013	0.00289	0.0001	0.0015

Tabella D.2: Risultati alcani CC best training

Weight Decay	Intervallo	λ	Unità	MaE	$MaxAbsErr$	R	S
0	10	0	23	0.0167722	0.0553637	0.998412	0.0221658
0.0001	30	0	21	0.0170474	0.0496358	0.998036	0.0224932
0	10	0	24	0.0166623	0.0514088	0.99824	0.0220648
0	30	0	23	0.0165704	0.0545962	0.998267	0.0219143
0	10	0	25	0.0165548	0.0471836	0.998476	0.0211731
0.0001	10	0	23	0.0167603	0.0488988	0.998272	0.0217309
0	10	0	23	0.0165846	0.0455235	0.99853	0.020735
0.0001	30	0	22	0.0168664	0.0468473	0.998242	0.0217734
0.0001	10	0	24	0.0164056	0.0503976	0.998261	0.0214535
0.0001	30	0	20	0.0175313	0.0579729	0.998135	0.0234833
Media			22.8	0.0167	0.0507	0.9982	0.0218
Deviazione Std			1.4757	0.0003	0.00406	0.00014	0.0007

Tabella D.3: Risultati alcani CC best validation

Weight Decay	Intervallo	λ	Unità	MaE	$MaxAbsErr$	R	S
0	10	0	23	0.0140434	0.0350527	0.997977	0.01907
0.0001	30	0	20	0.0382228	0.307433	0.991424	0.0824929
0	10	0	25	0.0189081	0.059187	0.998905	0.0242122
0	30	0	22	0.0145669	0.0452747	0.999022	0.0192571
0	10	0	24	0.0156329	0.0389056	0.999053	0.0187579
0.0001	10	0	24	0.0180503	0.0516354	0.998527	0.022834
0	10	0	17	0.019488	0.0612183	0.997858	0.0238065
0.0001	30	0	20	0.0151481	0.0293538	0.998818	0.0172744
0.0001	10	0	26	0.0158303	0.0435832	0.998144	0.0197358
0.0001	30	0	17	0.0186312	0.0533141	0.996919	0.0240095
Media			21.8	0.0188	0.0724	0.9976	0.0271
Deviazione Std			3.1902	0.007	0.0831	0.0022	0.0196

Tabella D.4: Risultati alcani CC best test

D.1.2 Cascade Residual

I risultati della griglia sono visibili in Tab. D.5, mentre i risultati delle migliori configurazioni in training e test, sulle fold esterne della DCV, sono visibili rispettivamente in Tab. D.6 e Tab. D.8.

Weight Decay	Intervallo	MaE_{tr}	MaE_{val}
0	10	0.00678669	0.0726284
0	30	0.00637269	0.0721711
5e-05	10	0.0445461	0.0483281
5e-05	30	0.0725845	0.0743592
0.0001	10	0.0454972	0.0497559
0.0001	30	0.0713543	0.0729342

Tabella D.5: Risultati alcani CR, mediati sul loop interno della DCV, delle configurazioni a griglia

Weight Decay	Intervallo	Unità	MaE	$MaxAbsErr$	R	S
0	30	36	0.0151646	0.0449809	0.99907	0.01873
5e-05	10	75	0.0311185	0.304167	0.995067	0.0463698
0.0001	10	75	0.0394144	0.37248	0.993449	0.055226
5e-05	10	75	0.0493502	0.819174	0.982533	0.0916111
5e-05	10	75	0.0474951	0.732789	0.985305	0.0841791
5e-05	10	75	0.0456167	0.758329	0.985378	0.0841028
0.0001	10	75	0.0439609	0.620774	0.989036	0.0736274
0.0001	10	75	0.0437947	0.84916	0.983272	0.0909324
5e-05	10	75	0.037245	0.429245	0.993594	0.0566673
5e-05	10	75	0.0527205	0.716737	0.985149	0.0862854
Media		71.1000	0.0406	0.5648	0.9892	0.0688
Deviazione Std		12.3329	0.0109	0.2648	0.0057	0.0239

Tabella D.6: Risultati alcani CR best training (esterno)

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>R</i>	<i>S</i>
0	30	38	0.0385701	0.135325	0.989926	0.0535478
5e-05	10	75	0.0435238	0.114803	0.986009	0.0541339
0.0001	10	75	0.0494091	0.159375	0.984286	0.0658925
5e-05	10	75	0.0485616	0.13193	0.987386	0.0596514
5e-05	10	75	0.0453495	0.108298	0.989041	0.055468
5e-05	10	75	0.0463983	0.15449	0.983565	0.0630733
0.0001	10	75	0.0494277	0.167647	0.986082	0.0666632
0.0001	10	75	0.052624	0.145296	0.984133	0.0663923
5e-05	10	75	0.0455564	0.126909	0.988258	0.0581227
5e-05	10	75	0.0506775	0.15844	0.984726	0.0666209
Media		71.3000	0.0470	0.1403	0.9863	0.0610
Deviazione Std		11.7004	0.0040	0.0200	0.0022	0.0054

Tabella D.7: *Risultati alcani CR best validation (interno)*

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>R</i>	<i>S</i>
0	30	36	0.0322951	0.0765166	0.992704	0.0361703
5e-05	10	75	0.0322424	0.0777457	0.997847	0.0413975
0.0001	10	75	0.0516143	0.158342	0.991369	0.067856
5e-05	10	75	0.0439426	0.0921512	0.993538	0.0494331
5e-05	10	75	0.0796648	0.496903	0.944536	0.141546
5e-05	10	75	0.0435662	0.11448	0.992145	0.052637
0.0001	10	75	0.0494097	0.117595	0.986242	0.060164
0.0001	10	75	0.0613826	0.338033	0.962042	0.0969787
5e-05	10	75	0.0528753	0.291518	0.96253	0.0878909
5e-05	10	75	0.0567754	0.120423	0.974852	0.0682151
Media		71.1000	0.0504	0.1884	0.9798	0.0702
Deviazione Std		12.3329	0.0140	0.1407	0.0179	0.0315

Tabella D.8: *Risultati alcani CR best test (esterno)*

D.2 PTC.FM

In questa sezione verranno mostrati i risultati, in forma estesa, dei test sul task **FM** del dataset PTC.

D.2.1 Cascade Correlation

I risultati di tutte le combinazioni dei parametri della griglia, mediati su tutte le fold di validation interne alla DCV e per la variante CC, sono visibili in Tab.D.9. Nelle Tab.D.10, D.11 e D.12 sono presentati i migliori risultati, rispettivamente, del training e test sul loop esterno e validation, sul loop interno, delle migliori configurazioni.

Weight Decay	Intervallo	λ	Acc_{tr}	Acc_{val}
0	10	0	0.807781	0.592371
0	10	0.05	0.805715	0.596918
0	10	0.1	0.805632	0.602586
0	25	0	0.809057	0.604839
0	25	0.05	0.806669	0.608935
0	25	0.1	0.807144	0.608971
0	30	0	0.808978	0.617542
0	30	0.05	0.806509	0.595346
0	30	0.1	0.805237	0.600374
0.05	10	0	0.811523	0.593103
0.05	10	0.05	0.805793	0.602309
0.05	10	0.1	0.805873	0.608669
0.05	25	0	0.810249	0.615991
0.05	25	0.05	0.806908	0.603241
0.05	25	0.1	0.807225	0.605796
0.05	30	0	0.808342	0.592985
0.05	30	0.05	0.807701	0.620527
0.05	30	0.1	0.806108	0.612186
0.1	10	0	0.808659	0.59787
0.1	10	0.05	0.80683	0.619467
0.1	10	0.1	0.808258	0.607409
0.1	25	0	0.807626	0.59595
0.1	25	0.05	0.807226	0.613088
0.1	25	0.1	0.805871	0.600087
0.1	30	0	0.808343	0.588684
0.1	30	0.05	0.807225	0.601301
0.1	30	0.1	0.806589	0.596022

Tabella D.9: Risultati CC del PTC.FM, mediati sul loop interno della DCV, delle configurazioni a griglia

Weight Decay	Intervallo	λ	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.1	25	0	72	0.616154	2.01394	0.802548	0.761194	0.772727	0.824176
0	30	0	72	0.673051	1.90921	0.805732	0.828571	0.669231	0.902174
0.05	10	0.05	72	0.66963	1.61622	0.802548	0.758333	0.733871	0.847368
0.1	10	0.05	72	0.666748	1.70631	0.802548	0.793103	0.707692	0.869565
0	25	0	72	0.675613	1.76691	0.815287	0.80531	0.716535	0.882353
0	30	0.05	72	0.677012	1.77848	0.802548	0.777778	0.742424	0.846154
0.1	10	0.05	72	0.664264	1.78509	0.805732	0.796296	0.688	0.883598
0	25	0.1	72	0.693134	1.73052	0.802548	0.782258	0.734848	0.851648
0	10	0.1	72	0.671133	1.92222	0.802548	0.787611	0.700787	0.871658
0.05	25	0.1	73	0.646612	1.90542	0.812698	0.789916	0.734375	0.86631
Media			72.1	0.6653	1.8134	0.8055	0.7880	0.7200	0.8645
Deviazione Std			0.3162	0.0208	0.1207	0.0047	0.0205	0.0299	0.0226

Tabella D.10: Risultati PTC.FM migliori configurazioni CC training (esterno)

Weight Decay	Intervallo	λ	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.1	25	0	62	0.880151	2.50814	0.655965	0.596774	0.560606	0.725275
0	30	0	62	0.975356	3.57042	0.652791	0.580153	0.584615	0.701087
0.05	10	0.05	62	0.87033	2.80215	0.640195	0.551402	0.475806	0.747368
0.1	10	0.05	62	0.902386	2.14424	0.652791	0.586777	0.546154	0.728261
0	25	0	62	0.90488	3.48487	0.646339	0.566667	0.535433	0.721925
0	30	0.05	62	0.912663	2.69447	0.624168	0.567308	0.44697	0.752747
0.1	10	0.05	62	0.8629	2.57962	0.665489	0.602041	0.472	0.793651
0	25	0.1	62	0.909745	2.81471	0.649565	0.59322	0.530303	0.736264
0	10	0.1	62	0.884411	2.53528	0.655914	0.581197	0.535433	0.737968
0.05	25	0.1	63	0.838365	2.66207	0.669841	0.596774	0.578125	0.73262
Media			62.1000	0.8941	2.7796	0.6513	0.5822	0.5265	0.7377
Deviazione Std			0.3162	0.0370	0.4375	0.0128	0.0163	0.0467	0.0243

Tabella D.11: Risultati PTC.FM migliori configurazioni CC validation (interno)

Weight Decay	Intervallo	λ	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.1	25	0	35	0.971719	3.40991	0.714286	0.533333	0.727273	0.708333
0	30	0	35	1.05252	3.04589	0.6	0.454545	0.384615	0.727273
0.05	10	0.05	35	1.10023	2.46386	0.485714	0.555556	0.263158	0.75
0.1	10	0.05	35	1.053	4.20107	0.571429	0.416667	0.384615	0.681818
0	25	0	35	0.904661	2.72944	0.628571	0.6	0.5625	0.684211
0	30	0.05	35	0.97227	2.37989	0.714286	0.533333	0.727273	0.708333
0.1	10	0.05	35	1.14959	4.71039	0.457143	0.466667	0.388889	0.529412
0	25	0.1	35	0.949649	2.5831	0.657143	0.466667	0.636364	0.666667
0	10	0.1	35	0.829487	2.58664	0.628571	0.578947	0.6875	0.578947
0.05	25	0.1	34	0.982905	1.95061	0.470588	0.411765	0.466667	0.473684
Media			34.9000	0.9966	3.0061	0.5928	0.5017	0.5229	0.6509
Deviazione Std			0.3162	0.0945	0.8647	0.0952	0.0671	0.1673	0.0918

Tabella D.12: Risultati PTC.FM migliori configurazioni CC test (esterno)

D.2.2 Cascade Residual

I risultati di tutte le combinazioni dei parametri della griglia, mediati su tutte le fold di validation interne alla DCV, sono visibili in Tab.D.13. Nelle Tab.D.14, D.15 e D.16 sono presentati i migliori risultati, rispettivamente, del training e test sul loop esterno e validation, sul loop interno, delle migliori configurazioni.

Weight Decay	Intervallo	Acc_{tr}	Acc_{val}
0	10	0.814709	0.580988
0	25	0.812797	0.573973
0	30	0.813679	0.571142
0.05	10	0.733758	0.586662
0.05	25	0.729941	0.593379
0.05	30	0.723327	0.588346
0.1	10	0.682418	0.573641
0.1	25	0.686324	0.579365
0.1	30	0.683438	0.577788

Tabella D.13: Risultati CR del PTC.FM, mediati sul loop interno della DCV, delle configurazioni a griglia

Weight Decay	Intervallo	Unità	MaE	$MaxAbsErr$	Acc	$Prec$	Rec	$Spec$
0.1	25	80	0.808382	1.94433	0.726115	0.705357	0.598485	0.818681
0.1	30	80	3.5982	20.3127	0.535032	0.448052	0.530769	0.538043
0	10	15	0.676982	2.20994	0.805732	0.831579	0.637097	0.915789
0.05	25	80	0.72423	1.67697	0.738854	0.7	0.646154	0.804348
0	10	15	0.657288	2.34323	0.805732	0.794643	0.700787	0.877005
0.05	25	80	0.789147	1.93056	0.687898	0.651786	0.55303	0.785714
0.05	30	80	0.75399	1.71912	0.732484	0.678261	0.624	0.804233
0.05	30	80	0.752724	1.83069	0.726115	0.701754	0.606061	0.813187
0.05	25	80	0.799029	1.9764	0.713376	0.703297	0.503937	0.855615
0	10	16	0.627639	1.96048	0.809524	0.788136	0.726562	0.86631
Media		60.6000	1.0188	3.7904	0.7281	0.7003	0.6127	0.8079
Deviazione Std		31.2382	0.9084	5.8088	0.0802	0.1056	0.0706	0.1031

Tabella D.14: Risultati PTC.FM migliori configurazioni CR training (esterno)

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.1	25	80	0.91823	2.13307	0.621096	0.565657	0.424242	0.763736
0.1	30	80	1.00107	3.08894	0.598617	0.516949	0.469231	0.690217
0	10	12	0.987392	3.98395	0.617921	0.517544	0.475806	0.710526
0.05	25	80	0.933233	2.54368	0.607988	0.530973	0.461538	0.711957
0	10	13	0.902009	2.78813	0.630261	0.545455	0.519685	0.705882
0.05	25	80	0.98244	3.02728	0.579826	0.5	0.439394	0.681319
0.05	30	80	0.886653	2.3897	0.668664	0.59633	0.52	0.767196
0.05	30	80	0.895144	2.43229	0.633794	0.577982	0.477273	0.747253
0.05	25	80	0.899983	2.14401	0.633743	0.56383	0.417323	0.780749
0	10	12	0.914823	3.16867	0.64127	0.567568	0.492188	0.743315
Media		59.7000	0.9321	2.7700	0.6233	0.5482	0.4697	0.7302
Deviazione Std		32.6872	0.0424	0.5685	0.0245	0.0310	0.0356	0.0347

Tabella D.15: Risultati PTC.FM migliori configurazioni CR validation (interno)

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.1	25	80	0.90494	1.73479	0.6	0.421053	0.727273	0.541667
0.1	30	80	6.93313	102.036	0.685714	0.555556	0.769231	0.636364
0	10	15	0.866106	2.46909	0.628571	0.8	0.421053	0.875
0.05	25	80	1.04965	2.48909	0.457143	0.25	0.230769	0.590909
0	10	15	0.951309	2.77225	0.657143	0.625	0.625	0.684211
0.05	25	80	0.804288	1.86468	0.628571	0.416667	0.454545	0.708333
0.05	30	80	1.04193	2.47426	0.428571	0.444444	0.444444	0.411765
0.05	30	80	0.913053	3.55867	0.657143	0.461538	0.545455	0.708333
0.05	25	80	0.91807	1.54316	0.542857	0.5	0.5	0.578947
0	10	16	0.972838	1.89201	0.558824	0.5	0.466667	0.631579
Media		60.6000	1.5355	12.2834	0.5845	0.4974	0.5184	0.6367
Deviazione Std		31.2382	1.8980	31.5414	0.0868	0.1449	0.1576	0.1222

Tabella D.16: Risultati PTC.FM migliori configurazioni CR test (esterno)

D.3 PTC.FR

In questa sezione verranno mostrati i risultati, in forma estesa, dei test sul task **FR** del dataset PTC.

D.3.1 Cascade Correlation

I risultati di tutte le combinazioni dei parametri della griglia, mediati su tutte le fold di validation interne alla DCV e per la variante CC, sono visibili in Tab.D.17. Nelle Tab.D.18, D.19 e D.20 sono presentati i migliori risultati, rispettivamente, del training e test sul loop esterno e validation, sul loop interno, delle migliori configurazioni.

Weight Decay	Intervallo	λ	Acc_{tr}	Acc_{val}
0	10	0	0.811408	0.641047
0	10	0.05	0.805871	0.634375
0	10	0.1	0.805952	0.634732
0	25	0	0.809355	0.63122
0	25	0.05	0.806819	0.631205
0	25	0.1	0.806978	0.637852
0	30	0	0.808725	0.637892
0	30	0.05	0.807692	0.649841
0	30	0.1	0.806268	0.647693
0.05	10	0	0.809985	0.644812
0.05	10	0.05	0.808484	0.658447
0.05	10	0.1	0.807691	0.636582
0.05	25	0	0.810301	0.624251
0.05	25	0.05	0.80785	0.637505
0.05	25	0.1	0.806348	0.642232
0.05	30	0	0.809197	0.634678
0.05	30	0.05	0.809034	0.637877
0.05	30	0.1	0.807692	0.636265
0.1	10	0	0.810063	0.648314
0.1	10	0.05	0.806584	0.639058
0.1	10	0.1	0.807849	0.63814
0.1	25	0	0.808559	0.632123
0.1	25	0.05	0.807691	0.635645
0.1	25	0.1	0.807851	0.619469
0.1	30	0	0.811094	0.627088
0.1	30	0.05	0.807058	0.631513
0.1	30	0.1	0.806029	0.631518

Tabella D.17: Risultati **CC** del **PTC.FR**, mediati sul loop interno della DCV, delle configurazioni a griglia

Weight Decay	Intervallo	λ	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.1	25	0.05	29	0.630329	1.78141	0.806349	0.810811	0.560748	0.932692
0	30	0.05	47	0.599114	1.65852	0.803797	0.8	0.581818	0.92233
0.1	30	0	23	0.638038	1.95979	0.813291	0.776596	0.657658	0.897561
0.05	30	0	21	0.652194	1.80481	0.803797	0.805556	0.54717	0.933333
0	25	0	24	0.613781	2.02995	0.813291	0.793103	0.627273	0.912621
0.05	10	0.05	33	0.653598	1.76502	0.803797	0.876923	0.513514	0.960976
0.1	10	0.1	30	0.669693	1.71171	0.800633	0.820895	0.518868	0.942857
0	30	0.1	33	0.626309	1.62475	0.806962	0.795181	0.6	0.917476
0	25	0.05	40	0.642736	1.69114	0.813291	0.825	0.594595	0.931707
0.05	10	0.05	30	0.634473	1.76087	0.800633	0.782051	0.570093	0.91866
Media			31	0.636027	1.7788	0.806584	0.808612	0.577174	0.927021
Deviazione Std			7.9162	0.02	0.1279	0.005	0.0285	0.0453	0.0174

Tabella D.18: Risultati PTC.FR, sul loop esterno della DCV, delle migliori configurazioni CC training

Weight Decay	Intervallo	λ	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.1	25	0.05	24	0.841513	2.69204	0.701587	0.591549	0.392523	0.860577
0	30	0.05	27	0.825718	2.54313	0.689782	0.578947	0.4	0.84466
0.1	30	0	21	0.886798	2.82504	0.683482	0.571429	0.396396	0.839024
0.05	30	0	21	0.829876	2.78563	0.689931	0.55	0.415094	0.828571
0	25	0	24	0.867262	3.22186	0.664435	0.522222	0.427273	0.791262
0.05	10	0.05	27	0.883036	2.38478	0.667659	0.539474	0.369369	0.829268
0.1	10	0.1	34	0.836278	2.5296	0.680556	0.533333	0.377358	0.833333
0	30	0.1	33	0.853133	2.37452	0.680506	0.556962	0.4	0.830097
0	25	0.05	32	0.864083	3.54677	0.674107	0.555556	0.36036	0.843902
0.05	10	0.05	24	0.815327	2.30545	0.693006	0.589286	0.308411	0.889952
Media			26.7	0.850302	2.72088	0.682505	0.558876	0.384678	0.839065
Deviazione Std			4.8085	0.0245	0.3975	0.0115	0.0236	0.0335	0.0251

Tabella D.19: Risultati PTC.FR, sul loop interno della DCV, delle migliori configurazioni CC validation

Weight Decay	Intervallo	λ	Unità	MaE	$MaxAbsErr$	Acc	$Prec$	Rec	$Spec$
0.1	25	0.05	29	0.994516	2.29503	0.444444	0.25	0.214286	0.590909
0	30	0.05	47	0.830054	2.54961	0.685714	0.5	0.363636	0.833333
0.1	30	0	23	0.980483	3.01728	0.571429	0.307692	0.4	0.64
0.05	30	0	21	0.727066	1.82911	0.685714	1	0.266667	1
0	25	0	24	0.9632	2.1866	0.571429	0.3	0.272727	0.708333
0.05	10	0.05	33	0.768391	1.96569	0.828571	0.833333	0.5	0.96
0.1	10	0.1	30	0.961182	2.14414	0.6	0.6	0.2	0.9
0	30	0.1	33	0.821044	2.51102	0.742857	0.6	0.545455	0.833333
0	25	0.05	40	0.945057	3.31973	0.657143	0.4	0.4	0.76
0.05	10	0.05	30	0.913288	1.5534	0.485714	0.333333	0.285714	0.619048
Media			31	0.890428	2.33716	0.627301	0.512436	0.344849	0.784496
Deviazione Std			7.9162	0.0959	0.5349	0.1164	0.2487	0.1171	0.1443

Tabella D.20: Risultati PTC.FR, sul loop esterno della DCV, delle migliori configurazioni CC test

D.3.2 Cascade Residual

I risultati di tutte le combinazioni dei parametri della griglia, mediati su tutte le fold di validation interne alla DCV e per la variante CC, sono visibili in Tab.D.21. Nelle Tab.D.22, D.23 e D.24 sono presentati i migliori risultati, rispettivamente, del training e test sul loop esterno e validation, sul loop interno, delle migliori configurazioni.

Weight Decay	Intervallo	Acc_{tr}	Acc_{val}
0	10	0.813778	0.622351
0	25	0.812755	0.623368
0	30	0.812202	0.620119
0.05	10	0.716665	0.633805
0.05	25	0.727522	0.641989
0.05	30	0.722056	0.636592
0.1	10	0.666939	0.631215
0.1	25	0.693008	0.648314
0.1	30	0.695476	0.640992

Tabella D.21: Risultati CR del PTC.FR, mediati sul loop interno della DCV, delle configurazioni a griglia

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.1	25	80	1.07312	2.0521	0.384127	0.355482	1	0.0673077
0.1	25	80	0.834041	1.83181	0.731013	0.719298	0.372727	0.92233
0.1	10	80	0.855348	1.69687	0.670886	0.606061	0.18018	0.936585
0	30	12	0.638774	2.08057	0.800633	0.765432	0.584906	0.909524
0.05	25	80	0.723901	1.86671	0.724684	0.649351	0.454545	0.868932
0.05	30	80	0.816544	1.80689	0.661392	0.590909	0.117117	0.956098
0.1	10	80	0.829921	1.75541	0.674051	0.542857	0.179245	0.92381
0.05	30	80	0.77069	2.19413	0.705696	0.697674	0.272727	0.936893
0.1	25	80	0.861526	2.00806	0.674051	0.568965	0.297297	0.878049
0.05	10	80	0.734872	2.02338	0.699367	0.62	0.28972	0.909091
Media		73.2000	0.8139	1.9316	0.6726	0.6116	0.3748	0.8309
Deviazione Std		21.5035	0.1149	0.1619	0.1094	0.1139	0.2598	0.2696

Tabella D.22: Risultati PTC.FR, sul loop esterno della DCV, delle migliori configurazioni CR training

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.1	25	80	0.844781	2.19557	0.673016	0.611111	0.102804	0.966346
0.1	25	80	0.871963	2.04694	0.686607	0.612245	0.272727	0.907767
0.1	10	80	0.882543	2.03297	0.66126	0.538462	0.252252	0.882927
0	30	11	0.863828	2.56767	0.664534	0.5	0.349057	0.82381
0.05	25	80	0.841068	2.39444	0.680357	0.584906	0.281818	0.893204
0.05	30	80	0.871244	1.96466	0.674107	0.564516	0.315315	0.868293
0.1	10	80	0.876322	2.26907	0.680407	0.592593	0.150943	0.947619
0.05	30	80	0.844381	2.0173	0.664831	0.555556	0.181818	0.92233
0.1	25	80	0.880345	2.1074	0.664534	0.555556	0.225225	0.902439
0.05	10	80	0.854623	2.01414	0.658135	0.488372	0.196262	0.894737
Media		73.1000	0.8631	2.1610	0.6708	0.5603	0.2328	0.9009
Varianza		21.8197	0.0158	0.1960	0.0095	0.0425	0.0762	0.0400

Tabella D.23: Risultati PTC.FR, sul loop interno della DCV, delle migliori configurazioni CR validation

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.1	25	80	1.0412	1.96834	0.416667	0.4	1	0.0454545
0.1	25	80	0.909471	2.06195	0.657143	0	0	0.958333
0.1	10	80	0.885595	1.39788	0.628571	0.285714	0.2	0.8
0	30	12	0.942484	2.32805	0.542857	0.461538	0.4	0.65
0.05	25	80	0.966336	2.40757	0.514286	0.2	0.181818	0.666667
0.05	30	80	0.883672	2.01475	0.685714	0.333333	0.1	0.92
0.1	10	80	0.893989	1.63683	0.571429	0.5	0.133333	0.9
0.05	30	80	0.801868	1.82842	0.685714	0.5	0.181818	0.916667
0.1	25	80	0.916847	2.43976	0.657143	0.375	0.3	0.8
0.05	10	80	0.908871	2.41322	0.685714	0.714286	0.357143	0.904762
Media		73.2000	0.9150	2.0497	0.6045	0.3770	0.2854	0.7562
Deviazione Std		21.5035	0.0619	0.3562	0.0908	0.1930	0.2783	0.2717

Tabella D.24: Risultati PTC.FR, sul loop esterno della DCV, delle migliori configurazioni CR test

D.4 PTC.MM

In questa sezione verranno mostrati i risultati, in forma estesa, dei test sul task MM del dataset PTC.

D.4.1 Cascade Correlation

I risultati di tutte le combinazioni dei parametri della griglia, mediati su tutte le fold di validation interne alla DCV e per la variante CC, sono visibili in Tab.D.25. Nelle Tab.D.26, D.27 e D.28 sono presentati i migliori risultati, rispettivamente, del training e test sul loop esterno e validation, sul loop interno, delle migliori configurazioni.

Weight Decay	Intervallo	λ	Acc_{tr}	Acc_{val}
0	10	0	0.807867	0.618656
0	10	0.05	0.807128	0.636104
0	10	0.1	0.805806	0.631131
0	25	0	0.809106	0.623514
0	25	0.05	0.805967	0.633202
0	25	0.1	0.80721	0.630623
0	30	0	0.808691	0.631874
0	30	0.05	0.806712	0.627667
0	30	0.1	0.805967	0.632219
0.05	10	0	0.809274	0.641836
0.05	10	0.05	0.808944	0.639563
0.05	10	0.1	0.805884	0.632519
0.05	25	0	0.809441	0.622995
0.05	25	0.05	0.808695	0.631596
0.05	25	0.1	0.807708	0.63877
0.05	30	0	0.808701	0.612678
0.05	30	0.05	0.808284	0.644874
0.05	30	0.1	0.805971	0.63124
0.1	10	0	0.808198	0.628792
0.1	10	0.05	0.80812	0.62229
0.1	10	0.1	0.807126	0.623322
0.1	25	0	0.807621	0.616683
0.1	25	0.05	0.807708	0.634557
0.1	25	0.1	0.806383	0.620328
0.1	30	0	0.810188	0.617279
0.1	30	0.05	0.806716	0.633219
0.1	30	0.1	0.807375	0.629175

Tabella D.25: Risultati **CC** del **PTC.MM**, mediati sul loop interno della **DCV**, delle configurazioni a griglia

Weight Decay	Intervallo	λ	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0	25	0.05	25	0.670139	1.70385	0.801324	0.745614	0.732759	0.844086
0	30	0.1	36	0.634167	1.58026	0.801324	0.858824	0.603306	0.933702
0	25	0.1	27	0.650429	1.87064	0.801324	0.808989	0.626087	0.909091
0	10	0	21	0.6638	1.80816	0.804636	0.785714	0.669565	0.887701
0	10	0.05	37	0.614993	1.90533	0.801324	0.887324	0.547826	0.957219
0.1	10	0	23	0.666935	2.33192	0.811258	0.763158	0.74359	0.854054
0.1	25	0.05	29	0.654783	1.63193	0.80198	0.819277	0.60177	0.921053
0.05	10	0	22	0.643932	1.94065	0.80198	0.773196	0.663717	0.884211
0.05	25	0.1	26	0.629161	1.89023	0.805281	0.762712	0.743802	0.846154
0.05	25	0.05	31	0.6386	1.94317	0.80198	0.777778	0.669565	0.882979
Media			27.7000	0.6467	1.8606	0.8032	0.7983	0.6602	0.8920
Deviazione Std			5.5588	0.0179	0.2099	0.0032	0.0455	0.0665	0.0382

Tabella D.26: Risultati PTC.MM, sul loop esterno della DCV, delle migliori configurazioni CC training

Weight Decay	Intervallo	λ	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0	25	0.05	28	0.859495	2.34251	0.658525	0.561905	0.508621	0.752688
0	30	0.1	32	0.880004	2.18666	0.668743	0.59633	0.53719	0.756906
0	25	0.1	31	0.867315	2.563	0.672022	0.574074	0.53913	0.754011
0	10	0	22	0.855921	2.49042	0.682022	0.594059	0.521739	0.780749
0	10	0.05	26	0.833307	2.10324	0.67541	0.589474	0.486957	0.791444
0.1	10	0	22	0.898905	2.78714	0.678306	0.619048	0.444444	0.827027
0.1	25	0.05	23	0.835132	2.41382	0.703333	0.635294	0.477876	0.836842
0.05	10	0	20	0.86661	2.66287	0.669891	0.568421	0.477876	0.784211
0.05	25	0.1	30	0.839512	2.34078	0.670382	0.612903	0.471074	0.802198
0.05	25	0.05	27	0.890965	3.16539	0.67	0.588235	0.434783	0.81383
Media			26.1000	0.8627	2.5056	0.6749	0.5940	0.4900	0.7900
Deviazione Std			4.2019	0.0228	0.3107	0.0118	0.0232	0.0362	0.0302

Tabella D.27: Risultati PTC.MM, sul loop interno della DCV, delle migliori configurazioni CC validation

Weight Decay	Intervallo	λ	Unità	MaE	$MaxAbsErr$	Acc	$Prec$	Rec	$Spec$
0	25	0.05	25	0.866124	2.74005	0.676471	0.625	0.384615	0.857143
0	30	0.1	36	0.73275	2.25222	0.823529	0.583333	0.875	0.807692
0	25	0.1	27	0.759061	1.79482	0.735294	0.727273	0.571429	0.85
0	10	0	21	0.861399	2.62205	0.705882	0.642857	0.642857	0.75
0	10	0.05	37	0.877069	2.14616	0.617647	0.555556	0.357143	0.8
0.1	10	0	23	0.82483	1.75673	0.617647	0.4	0.166667	0.863636
0.1	25	0.05	29	0.967471	2.36505	0.545455	0.545455	0.375	0.705882
0.05	10	0	22	0.848259	2.63889	0.69697	0.8	0.5	0.882353
0.05	25	0.1	26	0.892792	1.91912	0.606061	0.352941	0.75	0.56
0.05	25	0.05	31	0.960862	2.25142	0.515152	0.416667	0.357143	0.631579
Media			27.7000	0.8591	2.2487	0.6540	0.5649	0.4980	0.7708
Deviazione Std			5.5588	0.0752	0.3518	0.0921	0.1438	0.2133	0.1083

Tabella D.28: Risultati PTC.MM, sul loop esterno della DCV, delle migliori configurazioni **CC** test

D.4.2 Cascade Residual

I risultati di tutte le combinazioni dei parametri della griglia, mediati su tutte le fold di validation interne alla DCV e per la variante CR, sono visibili in Tab.D.29. Nelle Tab.D.30, D.31 e D.32 sono presentati i migliori risultati, rispettivamente, del training e test sul loop esterno e validation, sul loop interno, delle migliori configurazioni.

Weight Decay	Intervallo	Acc_{tr}	Acc_{val}
0	10	0.813403	0.613989
0	25	0.813738	0.605033
0	30	0.810681	0.615344
0.05	10	0.728676	0.630847
0.05	25	0.734371	0.64582
0.05	30	0.735779	0.631568
0.1	10	0.689753	0.63777
0.1	25	0.688755	0.636721
0.1	30	0.701719	0.631831

Tabella D.29: Risultati **CR** del **PTC.MM**, mediati sul loop interno della DCV, delle configurazioni a griglia

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0	10	13	0.65424	2.04255	0.801324	0.769231	0.689655	0.870968
0.1	25	80	0.783663	2.03855	0.695364	0.674699	0.46281	0.850829
0.05	25	80	0.757272	1.96985	0.741722	0.728395	0.513043	0.882353
0.05	25	80	0.764618	1.9753	0.705298	0.654762	0.478261	0.84492
0.05	30	80	0.705432	2.1977	0.761589	0.736264	0.582609	0.871658
0.05	25	80	0.774015	1.78598	0.711921	0.666667	0.512821	0.837838
0.05	30	80	0.735556	1.83727	0.729373	0.678161	0.522124	0.852632
0.1	25	80	0.806863	1.99982	0.650165	0.64	0.141593	0.952632
0.1	10	80	0.853789	2.26394	0.686469	0.641304	0.487603	0.818681
0	30	15	0.603516	1.89419	0.821782	0.821053	0.678261	0.909574
Media		66.8000	0.7439	2.0005	0.7305	0.7011	0.5069	0.8692
Deviazione Std		27.8320	0.0734	0.1480	0.0528	0.0604	0.1511	0.0387

Tabella D.30: Risultati PTC.MM, sul loop esterno della DCV, delle migliori configurazioni CR training

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0	10	13	0.912783	2.97573	0.62918	0.520408	0.439655	0.747312
0.1	25	80	0.871247	2.40707	0.638798	0.576923	0.371901	0.81768
0.05	25	80	0.855138	2.25462	0.678907	0.607143	0.443478	0.823529
0.05	25	80	0.88879	2.59526	0.655738	0.555556	0.478261	0.764706
0.05	30	80	0.860615	2.29011	0.662022	0.574713	0.434783	0.802139
0.05	25	80	0.887667	2.23417	0.679016	0.613636	0.461538	0.816216
0.05	30	80	0.846345	2.36315	0.669891	0.589041	0.380531	0.842105
0.1	25	80	0.885688	2.07746	0.636776	0.523077	0.300885	0.836842
0.1	10	80	0.854907	2.10211	0.663388	0.604396	0.454545	0.802198
0	30	12	0.893112	2.72702	0.679563	0.59	0.513043	0.781915
Media		66.5000	0.8756	2.4027	0.6593	0.5755	0.4279	0.8035
Deviazione Std		28.4615	0.0212	0.2851	0.0188	0.0331	0.0611	0.0307

Tabella D.31: Risultati PTC.MM, sul loop esterno della DCV, delle migliori configurazioni CR validation

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0	10	13	0.939954	2.72202	0.588235	0.461538	0.461538	0.666667
0.1	25	80	0.771057	1.83623	0.705882	0.416667	0.625	0.730769
0.05	25	80	0.943295	1.75351	0.588235	0.5	0.428571	0.7
0.05	25	80	0.832538	2.1536	0.705882	0.666667	0.571429	0.8
0.05	30	80	0.89001	1.91858	0.676471	0.615385	0.571429	0.75
0.05	25	80	0.923035	2.20297	0.558824	0.2	0.0833333	0.818182
0.05	30	80	1.07276	1.91227	0.454545	0.375	0.1875	0.705882
0.1	25	80	0.821551	2.10985	0.636364	1	0.25	1
0.1	10	80	0.925037	1.60875	0.636364	0.333333	0.5	0.68
0	30	15	1.04136	2.48594	0.575758	0.5	0.285714	0.789474
Media		66.8000	0.9161	2.0704	0.6127	0.5069	0.3965	0.7641
Deviazione Std		27.8320	0.0939	0.3399	0.0768	0.2192	0.1841	0.0977

Tabella D.32: *Risultati PTC.MM, sul loop esterno della DCV, delle migliori configurazioni CR test*

D.5 PTC.MR

In questa sezione verranno mostrati i risultati, in forma estesa, dei test sul task **MR** del dataset PTC.

D.5.1 Cascade Correlation

I risultati di tutte le combinazioni dei parametri della griglia, mediati su tutte le fold di validation interne alla DCV e per la variante CC, sono visibili in Tab.D.33. Nelle Tab.D.34, D.35 e D.36 sono presentati i migliori risultati, rispettivamente, del training e test sul loop esterno e validation, sul loop interno, delle migliori configurazioni.

Weight Decay	Intervallo	λ	Acc_{tr}	Acc_{val}
0	10	0	0.810078	0.581068
0	10	0.05	0.807091	0.557144
0	10	0.1	0.805961	0.56624
0	25	0	0.808702	0.571375
0	25	0.05	0.807817	0.565896
0	25	0.1	0.806606	0.567208
0	30	0	0.809185	0.567879
0	30	0.05	0.808219	0.559429
0	30	0.1	0.804987	0.556896
0.05	10	0	0.808543	0.568519
0.05	10	0.05	0.809186	0.562374
0.05	10	0.1	0.807089	0.559788
0.05	25	0	0.809349	0.561676
0.05	25	0.05	0.808223	0.590122
0.05	25	0.1	0.807983	0.577546
0.05	30	0	0.809916	0.567869
0.05	30	0.05	0.807411	0.558752
0.05	30	0.1	0.806928	0.554241
0.1	10	0	0.809353	0.579783
0.1	10	0.05	0.807975	0.56881
0.1	10	0.1	0.80709	0.562417
0.1	25	0	0.810239	0.56458
0.1	25	0.05	0.808222	0.563633
0.1	25	0.1	0.807735	0.56175
0.1	30	0	0.810966	0.58468
0.1	30	0.05	0.808301	0.572655
0.1	30	0.1	0.808622	0.55909

Tabella D.33: Risultati **CC** del **PTC.MR**, mediati sul loop interno della **DCV**, delle configurazioni a griglia

Weight Decay	Intervallo	λ	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.05	25	0.1	46	0.634323	1.59824	0.802589	0.772059	0.777778	0.821839
0	10	0	25	0.652532	2.06401	0.805825	0.776119	0.776119	0.828571
0.05	30	0	26	0.638057	1.8247	0.81877	0.785185	0.796992	0.835227
0.1	10	0	25	0.668653	1.96592	0.815534	0.798561	0.792857	0.83432
0	10	0	22	0.667286	1.76666	0.803226	0.772059	0.777778	0.822857
0	10	0	28	0.666548	2.21976	0.803226	0.773723	0.779412	0.821839
0.1	10	0.05	27	0.689522	1.86729	0.803226	0.777778	0.772059	0.827586
0.1	30	0	27	0.634999	1.83439	0.803226	0.784722	0.79021	0.814371
0.1	10	0	33	0.638414	1.87781	0.812903	0.813953	0.755396	0.859649
0	25	0	26	0.664405	1.56292	0.812903	0.811024	0.751825	0.861272
Media			28.5000	0.6555	1.8582	0.8081	0.7865	0.7770	0.8328
Deviazione Std			6.7536	0.0187	0.1972	0.0062	0.0159	0.0148	0.0159

Tabella D.34: *Risultati PTC.MR, sul loop esterno della DCV, delle migliori configurazioni CC training*

Weight Decay	Intervallo	λ	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0.05	25	0.1	33	0.962643	2.97528	0.618086	0.56391	0.555556	0.666667
0	10	0	25	0.908269	2.88375	0.643892	0.590909	0.58209	0.691429
0.05	30	0	25	0.958013	2.56159	0.614913	0.55	0.578947	0.642045
0.1	10	0	26	0.912672	3.11418	0.61174	0.566667	0.607143	0.615385
0	10	0	22	0.901737	2.78398	0.622581	0.569231	0.548148	0.68
0	10	0	23	0.95962	2.77214	0.596774	0.542636	0.514706	0.66092
0.1	10	0.05	31	0.942438	2.43999	0.590323	0.536	0.492647	0.666667
0.1	30	0	25	0.9912	3.25195	0.590323	0.556338	0.552448	0.622754
0.1	10	0	25	0.935555	2.99417	0.609677	0.569231	0.532374	0.672515
0	25	0	26	0.952301	2.67988	0.622581	0.578125	0.540146	0.687861
Media			26.1	0.9424	2.8456	0.6120	0.5623	0.550	0.66
Deviazione Std			3.3813	0.0282	0.24931	0.0165	0.0165	0.0335	0.026

Tabella D.35: *Risultati PTC.MR, sul loop interno della DCV, delle migliori configurazioni CC validation*

Weight Decay	Intervallo	λ	Unità	MaE	$MaxAbsErr$	Acc	$Prec$	Rec	$Spec$
0.05	25	0.1	46	0.987095	2.60379	0.6	0.615385	0.470588	0.722222
0	10	0	25	0.999089	2.75596	0.571429	0.6	0.5	0.647059
0.05	30	0	26	1.0712	3.24735	0.457143	0.5	0.473684	0.4375
0.1	10	0	25	0.873411	2.70109	0.657143	0.5	0.666667	0.652174
0	10	0	22	1.09495	2.22138	0.470588	0.473684	0.529412	0.411765
0	10	0	28	1.01554	4.19062	0.588235	0.555556	0.625	0.555556
0.1	10	0.05	27	0.97099	2.04835	0.529412	0.5	0.3125	0.722222
0.1	30	0	27	0.906079	2.79428	0.529412	0.294118	0.555556	0.52
0.1	10	0	33	1.01369	2.55045	0.5	0.357143	0.384615	0.571429
0	25	0	26	0.993449	2.86891	0.617647	0.583333	0.466667	0.736842
Media			28.5	0.9925	2.7982	0.5521	0.4979	0.4984	0.5976
Deviazione Std			6.753	0.0665	0.5916	0.0655	0.1037	0.1045	0.1176

Tabella D.36: Risultati PTC.MR, sul loop esterno della DCV, delle migliori configurazioni CC test

D.5.2 Cascade Residual

I risultati di tutte le combinazioni dei parametri della griglia, mediati su tutte le fold di validation interne alla DCV e per la variante CR, sono visibili in Tab.D.37. Nelle Tab.D.38, D.39 e D.40 sono presentati i migliori risultati, rispettivamente, del training e test sul loop esterno e validation, sul loop interno, delle migliori configurazioni.

Weight Decay	Intervallo	Acc_{tr}	Acc_{val}
0	10	0.814035	0.564537
0	25	0.815081	0.567456
0	30	0.812902	0.567171
0.05	10	0.727325	0.561988
0.05	25	0.71584	0.556494
0.05	30	0.708407	0.546478
0.1	10	0.671991	0.554913
0.1	25	0.664651	0.559794
0.1	30	0.663999	0.554548

Tabella D.37: Risultati CR del PTC.MR, mediati sul loop interno della DCV, delle configurazioni a griglia

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0	25	17	0.655404	1.94336	0.802589	0.768116	0.785185	0.816092
0	10	17	0.605158	1.86188	0.822006	0.784173	0.813433	0.828571
0.05	25	80	0.755317	2.05191	0.721683	0.676692	0.676692	0.755682
0.1	30	80	0.864714	1.76833	0.656958	0.630769	0.585714	0.715976
0	10	13	0.65676	1.91411	0.825806	0.791367	0.814815	0.834286
0	30	14	0.66989	1.93334	0.806452	0.792308	0.757353	0.844828
0.05	25	80	0.763709	1.99956	0.712903	0.682171	0.647059	0.764368
0.05	30	80	0.753207	1.81779	0.732258	0.806122	0.552448	0.886228
0.05	10	80	0.789496	1.85622	0.712903	0.689394	0.654676	0.760234
0.05	10	80	0.759474	1.92141	0.741935	0.72093	0.678832	0.791907
Media		54.1	0.7273	1.906	0.7535	0.7342	0.6966	0.7998
Deviazione Std		33.4579	0.0781	0.084	0.0571	0.0617	0.0925	0.0512

Tabella D.38: Risultati PTC.MR, sul loop esterno della DCV, delle migliori configurazioni CR training

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0	25	13	0.974434	3.67214	0.592068	0.536585	0.488889	0.672414
0	10	12	0.933351	2.51149	0.605024	0.544118	0.552239	0.645714
0.05	25	80	0.94376	2.64812	0.608302	0.545455	0.541353	0.659091
0.1	30	80	0.985159	2.5997	0.576044	0.532847	0.521429	0.621302
0	10	12	0.93228	2.56546	0.616129	0.567797	0.496296	0.708571
0	30	13	0.932409	3.05321	0.603226	0.547445	0.551471	0.643678
0.05	25	80	0.937838	2.59201	0.577419	0.517483	0.544118	0.603448
0.05	30	80	0.940915	2.98686	0.570968	0.533784	0.552448	0.586826
0.05	10	80	0.944063	2.5352	0.6	0.553191	0.561151	0.631579
0.05	10	80	0.940495	2.46703	0.587097	0.531034	0.562044	0.606936
Media		53	0.946	2.763	0.5936	0.5409	0.5371	0.6379
Deviazione Std		34.8584	0.0182	0.3759	0.0153	0.0138	0.0261	0.0362

Tabella D.39: Risultati PTC.MR, sul loop interno della DCV, delle migliori configurazioni CR validation

Weight Decay	Intervallo	Unità	<i>MaE</i>	<i>MaxAbsErr</i>	<i>Acc</i>	<i>Prec</i>	<i>Rec</i>	<i>Spec</i>
0	25	17	0.919229	3.1991	0.542857	0.533333	0.470588	0.611111
0	10	17	1.16826	5.46062	0.485714	0.5	0.444444	0.529412
0.05	25	80	0.811766	1.51682	0.685714	0.785714	0.578947	0.8125
0.1	30	80	0.90864	1.5381	0.571429	0.333333	0.25	0.73913
0	10	13	0.992625	1.9973	0.558824	0.5625	0.529412	0.588235
0	30	14	1.17327	4.32264	0.558824	0.533333	0.5	0.611111
0.05	25	80	0.873011	1.68939	0.735294	0.733333	0.6875	0.777778
0.05	30	80	0.76962	2.10821	0.735294	0.5	0.444444	0.84
0.05	10	80	0.92595	2.32865	0.705882	0.615385	0.615385	0.761905
0.05	10	80	0.951352	1.88635	0.5	0.4	0.266667	0.684211
Media		54.1	0.949372	2.60472	0.607983	0.549693	0.478739	0.695539
Deviazione Std		33.4579	0.1333	1.3261	0.0972	0.1368	0.1394	0.1059

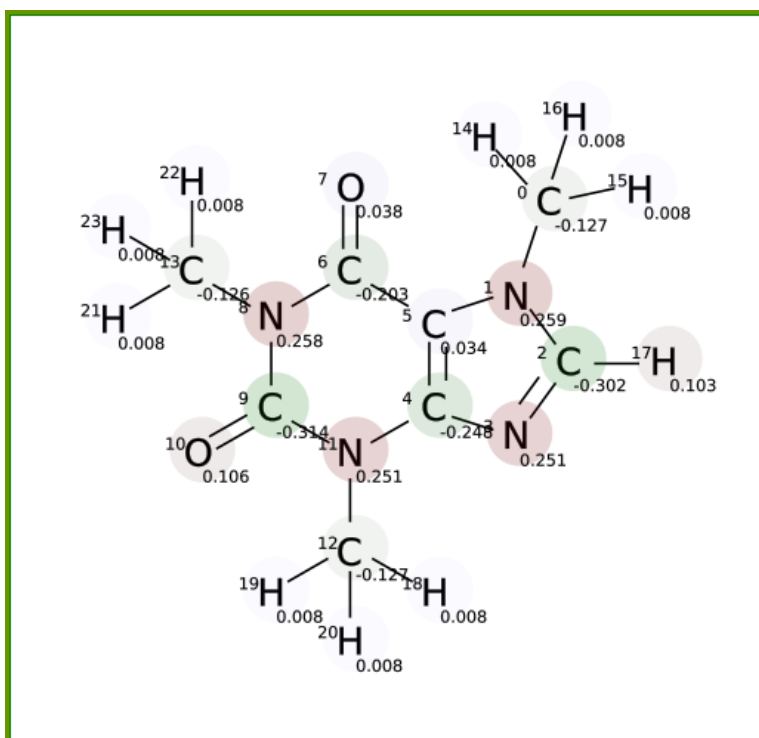
Tabella D.40: *Risultati PTC.MR, sul loop esterno della DCV, delle migliori configurazioni CR test*

Appendice E

Report Generato dal Predittore

Questo è un esempio di report generato dal predittore visto in Cap. 5. Il PDF generato contiene esattamente quanto segue, e la predizione è stata effettuata sulla molecola della caffeina per l'endpoint di mutagenicità

E.1 Predictor's Report of $C_8H_{10}N_4O_2$



Negative

(-0.171151)

E.2 Infos

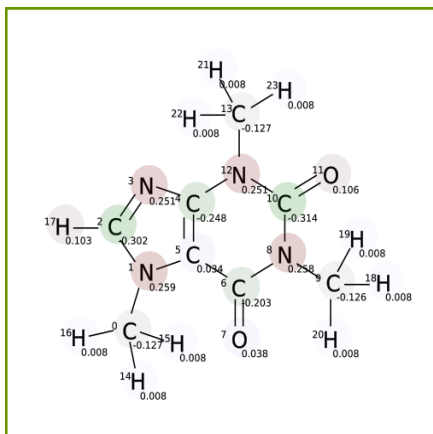
SMILES Cn1cnc2c1c(=O)n(C)c(=O)n2C

Formula $f^{(0)}C^*(-0.127) + {}^{(1)}N^*(0.259) + {}^{(2)}C^*(-0.302) + {}^{(3)}N^*(0.251) + {}^{(4)}C^*(-0.248)$
 $+ {}^{(5)}C^*(0.034) + {}^{(6)}C^*(-0.203) + {}^{(7)}O^*(0.037) + {}^{(8)}N^*(0.258) + {}^{(9)}C^*(-0.314) +$
 ${}^{(10)}O^*(0.106) + {}^{(11)}N^*(0.251) + {}^{(12)}C^*(-0.127) + {}^{(13)}C^*(-0.126) + {}^{(14)}H^*(0.008) +$
 ${}^{(15)}H^*(0.008) + {}^{(16)}H^*(0.008) + {}^{(17)}H^*(0.103) + {}^{(18)}H^*(0.008) + {}^{(19)}H^*(0.008) +$
 ${}^{(20)}H^*(0.008) + {}^{(21)}H^*(0.008) + {}^{(22)}H^*(0.008) + {}^{(23)}H^*(0.008) - 0.097 = -0.171$

E.3 Applicability Domain

- ✓ Molecule present in the training set.
- ✓ Accuracy of prediction for similar molecules found in the training set is good
- ✓ The 75% of the training set's molecules have the same atoms number (24 atoms)
- ✱ Some similar molecules found in the training set have experimental values that disagree with the predicted value

E.4 Similar Molecules



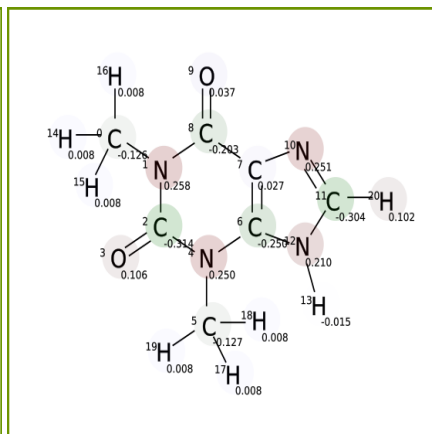
Name 58-08-258-08-2

Similarity 100%

Target -1

Out -0.0594651

Predicton Correct



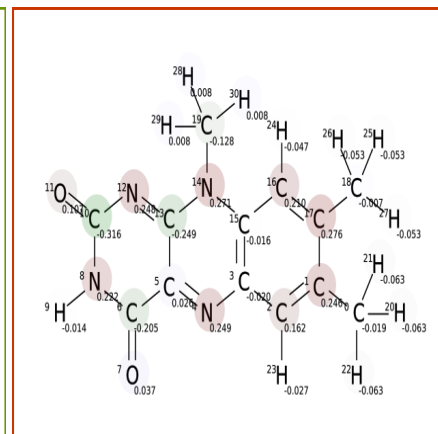
Name 58-55-958-08-2

Similarity 97%

Target -1

Out -0.0642717

Predicton Correct



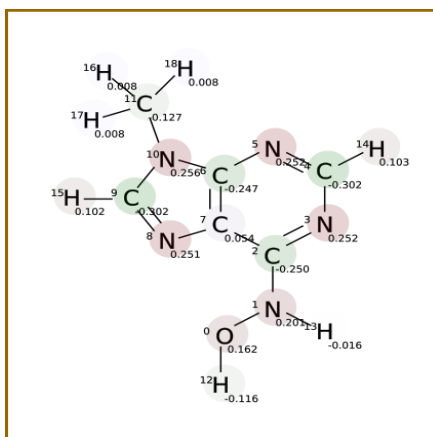
Name 1088-56-858-08-2

Similarity 57%

Target 1

Out 1.11836

Predicton Correct



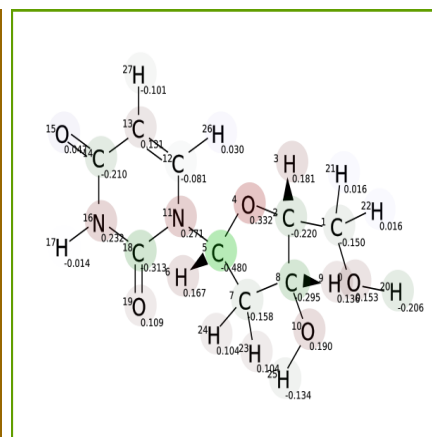
Name 7269-60-558-08-2

Similarity 43%

Target 1

Out 0.246307

Predicton Correct



Name 951-78-058-08-2

Similarity 42%

Target -1

Out -0.8209

Predicton Correct

Appendice F

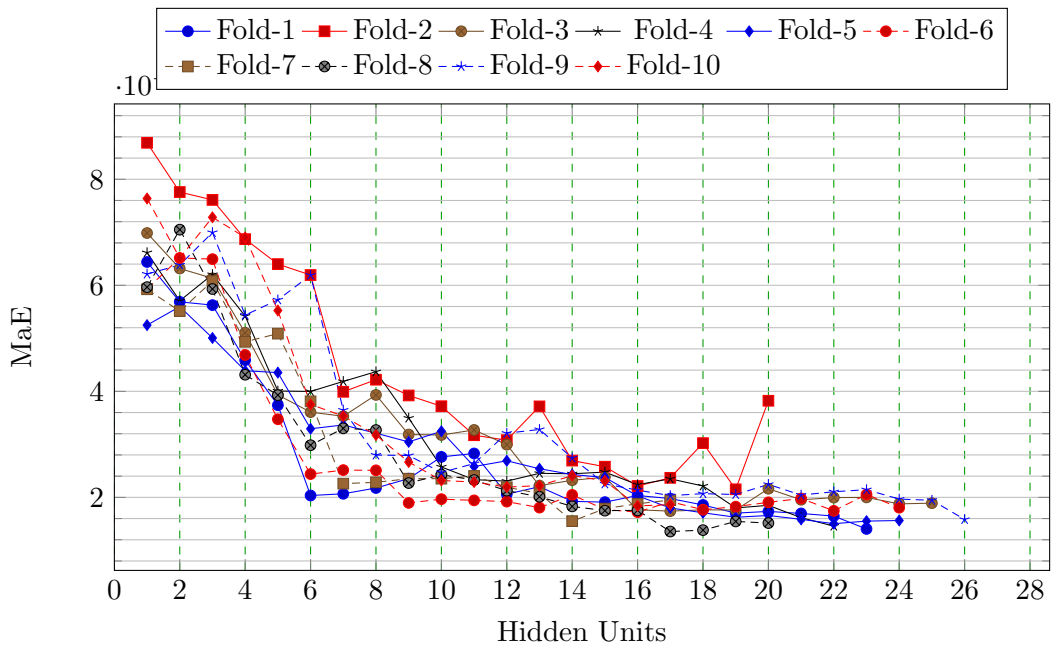
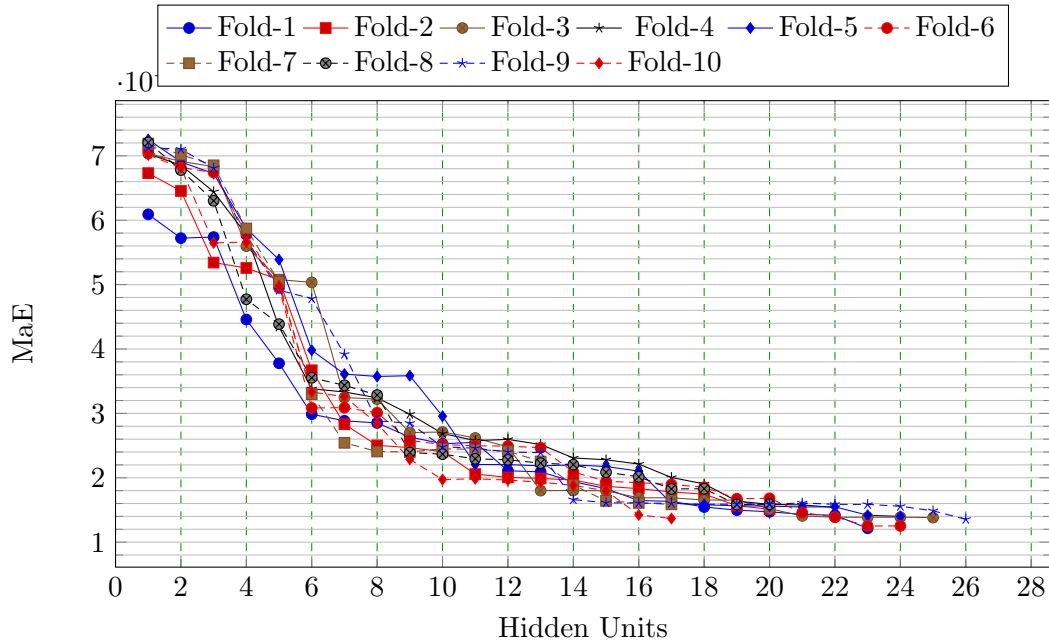
Report Generato da *felix-reader*

Questo è un esempio di report generato dal tool *felix-reader* visto in Sez. 4.3.2. L'esempio mostra il report generato da una CV interna del test visto in Sez. 6.1.1. Per motivi di brevità è riportato il dettaglio di due sole fold, ma il PDF originale conteneva le informazioni di training riguardo tutte le 10 fold su cui è avvenuto l'addestramento e validazione.

F.1 Results Info

- Name of the dataset: `/home/barontil/tesi/data/alkanes/alkanes_cv.gph`
- Number of trials: **1**
- Kind of separation: **10-Folds CV**
- Kind of task: **Regression**
- Target Range [min,max]: **[-1.64, 1.7412]**
- Null Model Error (on Training): **0.31732**
- seed: **1394906342**
- Time Elapsed: **0:00:13**

F.2 Training Results



Averaged Fold's Results

	MeanAbsError	MaxAbsError	Iterations	R	S
Training	0.0141971	0.0476104	21	0.99932	0.0178817
Validation	0.0188522	0.0724958	21	0.997665	0.027145

- Averaged number of Hidden Units: 21

F.2.1 Fold-1 results

Averaged Final Results

	MeanAbsError	MaxAbsError	Iterations	R	S
Training	0.0121569	0.0414067	23	0.999362	0.0155182
Validation	0.0140434	0.0350527	23	0.997977	0.01907

- Averaged number of Hidden Units: **23**

Final stop criteria: *Current MaxAbsError is equals or lesser than the minimum one ($0.0414067 \leq 0.05$)*

Here are the best/worst patterns on the **Training** dataset

Best				Worst			
<i>Pattern</i>	<i>Target</i>	<i>Output</i>	$ o - t $	<i>Pattern</i>	<i>Target</i>	<i>Output</i>	$ o - t $
45	1.405	1.405	3.18338e-06	150	1.7412	1.78261	0.0414067
123	1.57	1.56974	0.000263326	126	1.66	1.62049	0.0395107
110	1.575	1.57449	0.000508807	133	1.448	1.48718	0.0391805
88	1.65	1.64939	0.000613912	97	1.533	1.49507	0.0379306
102	1.53	1.53061	0.00061476	120	1.63	1.5929	0.0370964

Here are the best/worst patterns on the **Validation** dataset

Best				Worst			
<i>Pattern</i>	<i>Target</i>	<i>Output</i>	$ o - t $	<i>Pattern</i>	<i>Target</i>	<i>Output</i>	$ o - t $
71	1.433	1.433	3.52581e-06	41	1.4027	1.36765	0.0350527
111	1.638	1.63632	0.00168052	131	1.67	1.63628	0.0337207
61	1.405	1.40675	0.00174816	51	1.265	1.29862	0.0336215
141	1.5884	1.5865	0.00190037	81	1.549	1.57902	0.0300209
31	1.156	1.15348	0.00251882	11	0.603	0.589042	0.0139576

F.2.2 Fold-2 results

Averaged Final Results

	MeanAbsError	MaxAbsError	Iterations	R	S
Training	0.0151057	0.0463075	20	0.999151	0.0192508
Validation	0.0382228	0.307433	20	0.991424	0.0824929

- Averaged number of Hidden Units: **20**

Final stop criteria: *Current MaxAbsError is equals or lesser than the minimum one ($0.0463075 \leq 0.05$)*

Here are the best/worst patterns on the **Training** dataset

Best				Worst			
<i>Pattern</i>	<i>Target</i>	<i>Output</i>	$ o - t $	<i>Pattern</i>	<i>Target</i>	<i>Output</i>	$ o - t $
1	-1.64	-1.63997	2.69022e-05	100	1.47	1.51631	0.0463075
118	1.64	1.63977	0.000232994	150	1.7412	1.78576	0.0445612
71	1.433	1.43271	0.000287825	51	1.265	1.30709	0.0420879
34	1.094	1.09367	0.000327736	96	1.74	1.69827	0.0417261
111	1.638	1.63747	0.000527006	81	1.549	1.589	0.0399954

Here are the best/worst patterns on the **Validation** dataset

Best				Worst			
<i>Pattern</i>	<i>Target</i>	<i>Output</i>	$ o - t $	<i>Pattern</i>	<i>Target</i>	<i>Output</i>	$ o - t $
102	1.53	1.52959	0.000412603	2	-0.886	-1.19343	0.307433
142	1.5987	1.60506	0.00636168	132	1.62	1.67074	0.0507424
12	0.633	0.640194	0.00719414	82	1.553	1.59535	0.0423462
42	1.33	1.33977	0.00977225	22	0.984	0.95284	0.0311604
92	1.6944	1.70686	0.01246	62	1.38	1.40084	0.0208396

F.3 Parameters

F.3.1 Model Parameters

Model **NN4G**

- weights (minWeightValue=**-0.5**) (maxWeightValue=**0.5**)
- outputActivationFunction **Linear**
- hiddenActivationFunction **Sigmoidal** (max=**1**) (min=**-1**) (slope=**0.4**)
- normalization (type=**Normalized**)

F.3.2 Learning Parameters

Learning **CascadeCorrelation** (verbose=**true**) (saveExtendedLogging=**false**)

- pool (size=**4**)
- candidateLearning **QuickPropagation** (learningRate=**0.004**) (weightDecay=**0**) (normalizeDerivate=**true**) (saveExtendedLogging=**false**)
 - stopCriteria **maxIterationReached** (maxIteration=**300**)

- stopCriteria **minIterations** (minIteration=**20**) (tick=**10**)
- stopCriteria **resultsStale** (staleTreshold=**0.1**) (nResultsToCheck=**2**) (checkTrainingResults=**true**)
- outputLearning **Pseudoinverse** (lambda=**0**)
- stopCriteria **maxIterationReached** (maxIteration=**200**)
- stopCriteria **minimumMaxAbsError** (minMae=**0.05**)

Bibliografia

- [1] T. Mitchell, *Machine Learning*. McGraw Hill, 1997.
- [2] A. Micheli, “Neural Network for Graphs: A Contextual Constructive Approach,” *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.
- [3] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.
- [4] M. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: The rprop algorithm. In IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS,” 1993.
- [5] S. E. Fahlman, “An Empirical Study of Learning Speed in Back-Propagation Networks,” Carnegie-Mellon Univ., Tech. Rep. CMU-CS-88-162, 1988.
- [6] E. H. Moore, “On the reciprocal of the general algebraic matrix,” *Bulletin of the American Mathematical Society*, vol. 26, pp. 349–395, 1920.
- [7] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, 1996.
- [8] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning*. Springer Verlag, August 2001.
- [9] S. E. Fahlman and C. Lebiere, “The Cascade-Correlation Learning Architecture.” in *NIPS*, D. S. Touretzky, Ed. Morgan Kaufmann, 1989, pp. 524–532.
- [10] L. Prechelt, “Investigation of the CasCor Family of Learning Algorithms.” *Neural Networks*, vol. 10, no. 5, pp. 885–896, 1997.
- [11] N. Qian and T. Sejnowski, “Predicting the secondary structure of globular proteins using neural network models,” *Journal of molecular biology*, vol. 202, no. 4, pp. 865–884, 1988.

- [12] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang, “Phoneme recognition using time-delay neural networks,” *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 3, pp. 328–339, May 1989.
- [13] V. N. Vapnik, “An Overview of Statistical Learning Theory,” *IEEE Transaction on Neural Networks*, 1999.
- [14] —, “Pattern Recognition Using Generalized Portrait Method,” *Automation and Remote Control*, vol. 24, pp. 774–780, 1963.
- [15] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*. New York, NY, USA: ACM Press, 1992, pp. 144–152.
- [16] C. Cortes and V. Vapnik, “Support Vector Networks,” *Machine Learning*, vol. 20, pp. 273–297, 1995.
- [17] Q. Liao, J. Yao, and S. Yuan, “Prediction of Mutagenic Toxicity by Combination of Recursive Partitioning and Support Vector Machines,” *Molecular Diversity*, vol. 11, no. 2, pp. 59–72, 2007.
- [18] D. Haussler, “Convolution Kernels on Discrete Structures,” 1999.
- [19] H. Kashima, K. Tsuda, and A. Inokuchi, “Marginalized kernels between labeled graphs,” in *Proceedings of the 20th International Conference on Machine Learning*, 2003.
- [20] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell, “Optimal assignment kernels for attributed molecular graphs,” in *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005)*, L. de Raedt and S. Wrobel, Eds. Bonn, Germany: ACM Press, August 2005, pp. 225–232.
- [21] J. Ramon and T. Gärtner, “Expressivity versus efficiency of graph kernels,” in *First International Workshop on Mining Graphs, Trees and Sequences*, 2003, pp. 65–74.
- [22] J. F. Kolen and S. C. Kremer, *A field guide to dynamical recurrent networks*. John Wiley & Sons, 2001.

- [23] M. Lukosevicius and H. Jaeger, “Reservoir Computing Approaches to Recurrent Neural Network Training.” *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.
- [24] R. J. Williams and D. Zipser, “A Learning Algorithm for Continually Running Fully Recurrent Neural Networks,” *Neural Computation*, vol. 1, no. 2, pp. 270–280, 1989.
- [25] P. Werbos, “Backpropagation through time: what does it do and how to do it,” in *Proceedings of IEEE*, vol. 78, no. 10, 1990, pp. 1550–1560.
- [26] H. Jaeger, “The echo state approach to analysing and training recurrent neural networks,” GMD - German National Research Institute for Computer Science, GMD Report 148, 2001.
- [27] —, “Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication,” *Science*, vol. 304, pp. 78–80, 2004.
- [28] —, “Adaptive Nonlinear System Identification with Echo State Networks.” in *NIPS*, S. Becker, S. Thrun, and K. Obermayer, Eds. MIT Press, 2002, pp. 593–600.
- [29] D. Verstraeten, B. Schrauwen, M. D’Haene, and D. Stroobandt, “An Experimental Unification of Reservoir Computing Methods.” *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.
- [30] B. Schrauwen, D. Verstraeten, and J. M. V. Campenhout, “An overview of reservoir computing: theory, applications and implementations.” in *ESANN*, 2007, pp. 471–482.
- [31] C. Gallicchio and A. Micheli, “Architectural and Markovian factors of echo state networks.” *Neural Networks*, vol. 24, no. 5, pp. 440–456, 2011.
- [32] —, “Tree Echo State Networks.” *Neurocomputing*, vol. 101, pp. 319–337, 2013.
- [33] A. Sperduti, D. Majidi, and A. Starita, “Extended cascade-correlation for syntactic and structural pattern recognition,” *Advances in Structural and Syntactical Pattern Recognition*, vol. 1121, pp. 90–99, 1996.

- [34] S. Fahlman and C. Lebiere, "The recurrent cascade-correlation learning architecture," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-CS-91-100, May 1991.
- [35] A. Micheli, D. Sona, and A. Sperduti, "Contextual Processing of Structured Data by Recursive Cascade Correlation," *Transactions on Neural Networks*, vol. 15, no. 6, p. 13961410, Nov. 2004.
- [36] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," *Transactions on Neural Networks*, vol. 20, no. 1, p. 6180, Jan. 2009.
- [37] M. A. Khamsi, "An Introduction to Metric Spaces and Fixed Point Theory," Wiley, Tech. Rep., 2001.
- [38] M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *Computer Journal*, vol. 7, pp. 155–162, 1964.
- [39] C. Gallicchio and A. Micheli, "Graph Echo State Networks." in *IJCNN*. IEEE, 2010, pp. 1–8.
- [40] C. Helma, R. D. King, S. Kramer, and A. Srinivasan, "The Predictive Toxicology Challenge 2000-2001." *Bioinformatics*, vol. 17, no. 1, pp. 107–108, 2001.
- [41] R. J. Kavlock, G. Ankley, J. Blancato, M. Breen, R. Conolly, D. Dix, K. Houck, E. Hubal, R. Judson, J. Rabinowitz *et al.*, "Computational toxicology—a state of the science mini review," *Toxicological sciences*, vol. 103, no. 1, pp. 14–27, 2008.
- [42] H. Kubinyi, "From narcosis to hyperspace: the history of QSAR," *Quantitative Structure-Activity Relationships*, vol. 21, no. 4, pp. 348–356, 2002.
- [43] R. Benigni and C. Bossa, "Predictivity of QSAR," *Journal of chemical information and modeling*, vol. 48, no. 5, pp. 971–980, 2008.
- [44] OECD, "Guidance Document on the Validation of (Quantitative) Structure-Activity Relationship [(Q)SAR] Models," OECD, Tech. Rep., 2007.
- [45] R. Todeschini and V. Consonni, *Molecular Descriptors for Chemoinformatics, Volume 41 (2 Volume Set)*. John Wiley & Sons, 2009, vol. 41.

- [46] S. Thompson, C. Hattotuwigama, J. Holliday, and D. Flower, "On the hydrophobicity of peptides: Comparing empirical predictions of peptide log P values," *Bioinformatics*, vol. 1, no. 7, pp. 237–241, 2006.
- [47] T. T. Tanimoto and D. J. Rogers, "A Computer Program for Classifying Plants," *Science*, vol. 132, pp. 1115–1118, 1960.
- [48] R. Benigni and C. Bossa, "Structural Alerts of Mutagens and Carcinogens," *Current Computer-Aided Drug Design*, vol. 2, no. 2, pp. 169–176, 2006.
- [49] —, "Structure Alerts for Carcinogenicity, and the Salmonella Assay System: A Novel Insight Through the Chemical Relational Databases Technology," *Mutation Research/Reviews in Mutation Research*, vol. 659, no. 3, pp. 248–261, 2008.
- [50] J. Ashby, "Fundamental Structural Alerts to Potential Carcinogenicity or Non-carcinogenicity," *Environmental Mutagenesis*, vol. 7, no. 6, pp. 919–921, 1985.
- [51] R. Benigni, C. Bossa, N. Jeliaskova, T. I. Netzeva, and A. P. Worth, "The Benigni/Bossa Rulebase for Mutagenicity and Carcinogenicity - A Module of Toxtree," EUR, Tech. Rep., 2008.
- [52] A. Dalby, J. G. Nourse, W. D. Hounshell, A. K. Gushurst, D. L. Grier, B. A. Leland, and J. Laufer, "Description of Several Chemical Structure File Formats Used by Computer Programs Developed at Molecular Design Limited," *Journal of chemical information and computer sciences*, vol. 32, no. 3, pp. 244–255, 1992.
- [53] D. Weininger, "SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules," *Journal of chemical information and computer sciences*, vol. 28, no. 1, pp. 31–36, 1988.
- [54] "Estimation Programs Interface Suite for Microsoft Windows, v 4.11," Washington, DC, USA. [Online]. Available: <http://www.epa.gov/oppt/exposure/pubs/episuite.htm>
- [55] "T.E.S.T." [Online]. Available: <http://www.epa.gov/nrmrl/std/qsar/qsar.html#TEST>
- [56] J. H. Ward, "Hierarchical Grouping to Optimize an Objective Function," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 236–244, 1963.

- [57] “Vega.” [Online]. Available: <http://www.vega-qsar.eu>
- [58] N. M. O’Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, and G. R. Hutchison, “Open Babel: An open chemical toolbox,” *Journal of Cheminformatics*, vol. 3, p. 33, October 2011, health Research Board.
- [59] C. Sanderson, “Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments,” NICTA, PO Box 6020, St Lucia, QLD 4067, Tech. Rep., Sep. 2010. [Online]. Available: <http://arma.sourceforge.net>
- [60] Trolltech, “QT.” [Online]. Available: <http://doc.trolltech.com>
- [61] C. Helma, R. D. King, S. Kramer, and A. Srinivasan, “The predictive toxicology challenge 2000–2001,” *Bioinformatics*, vol. 17, no. 1, pp. 107–108, 2001.
- [62] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi, “Graph Kernels for Chemical Informatics,” *Neural Networks*, vol. 18, no. 8, pp. 1093–1110, 2005.
- [63] C. Gallicchio and A. Micheli, “Supervised State Mapping of Clustered GraphESN States.” in *WIRN*, 2011, pp. 28–35.
- [64] J. Kazius, R. McGuire, and R. Bursi, “Derivation and Validation of Toxicophores for Mutagenicity Prediction,” *Journal of Medicinal Chemistry*, vol. 48, no. 1, pp. 312–320, 2005.
- [65] T. Ferrari and G. Gini, “An open source multistep model to predict mutagenicity from statistical analysis and relevant structural alerts,” *Chemistry Central Journal*, vol. 4, no. Suppl 1, p. S2, 2010.
- [66] W. W. Piegorsch and E. Zeiger, “Measuring intra-assay agreement for the Ames Salmonella assay,” in *Statistical methods in toxicology*. Springer, 1991, pp. 35–41.
- [67] C. Helma, “Lazy structure-activity relationships (lazar) for the prediction of rodent carcinogenicity and Salmonella mutagenicity,” *Molecular diversity*, vol. 10, no. 2, pp. 147–158, 2006.
- [68] T. Ferrari, G. Gini, and E. Benfenati, “Support Vector Machines in the Prediction of Mutagenicity of Chemical Compounds,” *Proc NAFIPS 2009, June 14-17, Cincinnati, USA*, pp. 1–6, 2009.