



Università degli Studi di Milano-Bicocca
Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Magistrale in Informatica

Tesi di Laurea Magistrale

Continuous Time Bayesian Network Classifiers

Candidato:
Leonardo Di Donato
Matricola 744739

Relatore:
Prof. Fabio Stella

Correlatore:
Dott. Daniele Codecasa

Anno Accademico 2012-2013

Questa tesi è stata scritta utilizzando L^AT_EX. Il modello tipografico che si è adottato è una personalizzazione dell'autore (reperibile all'indirizzo <https://github.com/leodido/arsclassica>) del modello offerto dal pacchetto ArsClassica. Si sono utilizzati i seguenti font:

- Palatino per il testo
- Euler per le equazioni, i listati e i numeri
- Iwona sia per le unità di sezionamento (capitoli, sezioni, sotto-sezioni) sia per le etichette degli elenchi di descrizioni e degli elementi fluttuanti.

I grafici sono stati creati in linguaggio R (R Core Team, 2013) con l'ausilio del pacchetto ggplot2 (Wickham, 2009).

Leonardo Di Donato: *Continuous Time Bayesian Network Classifiers*.
Tesi di Laurea Magistrale © settembre 2013.

E-MAIL:

l.didonato@campus.unimib.it

leodidonato@gmail.com

Dedicato alla mia famiglia e a me stesso.

INDICE

INTRODUZIONE	x
1 CONTINUOUS TIME BAYESIAN NETWORK	1
1.1 Fondamenti	1
1.1.1 Bayesian Network	1
1.1.2 Processi di Markov	6
1.2 Definizioni preliminari	11
1.3 Rappresentazione	12
1.4 Apprendimento	13
1.4.1 Statistiche sufficienti	14
1.4.2 Likelihood	14
1.4.3 Stima dei parametri	16
2 CLASSIFICAZIONE	21
2.1 Modello	21
2.2 Apprendimento	24
2.3 Inferenza	28
3 APPRENDIMENTO STRUTTURALE	33
3.1 Funzione di scoring	34
3.2 Ricerca della struttura	36
3.2.1 Hill Climbing	37
4 PACCHETTO RCTBN	40
4.1 Il linguaggio R	40
4.1.1 Estendere R	41
4.2 Analisi	43
5 STRUMENTI PER LA CREAZIONE DI DATASET	48
5.1 TSIS	50
5.1.1 Componenti	51
5.1.2 Caratteristiche	53
5.2 Creazione di estensioni TSIS	56
5.2.1 Requisiti	57
5.2.2 Architettura di CORSIM	57
5.2.3 Ciclo di vita di CORSIM	59
5.2.4 Collegare una RTE a CORSIM	59
5.2.5 Utilizzo delle API	64
5.3 Estensione	66
5.3.1 Sensors DLL	66
5.3.2 Formato dell'output	69
5.4 Applicativi di supporto	71

6	ESPERIMENTI NUMERICI	72
6.1	Dataset #1	72
6.1.1	Modello TSIS	72
6.1.2	Configurazioni del dataset	77
6.2	Dataset #2	78
6.2.1	Modello TSIS	78
6.2.2	Configurazioni del dataset	86
6.3	Sperimentazione	88
6.3.1	Metriche di valutazione	89
6.3.2	Accuratezza	92
6.3.3	Curve ROC	95
	CONCLUSIONI	104
A	GUIDE ALL'USO	106
A.1	Utilizzo del pacchetto RCTBN	106
A.1.1	Gestione dei dataset	107
A.1.2	Calcolo delle statistiche sufficienti	110
A.1.3	Apprendimento dei parametri	113
A.1.4	Calcolo delle CIM	115
A.1.5	Apprendimento di un CTBNC	116
A.1.6	Classificazione di un CTNBC	119
A.1.7	Apprendimento strutturale	121
A.2	Generazione di dataset	123
A.2.1	Sensors DLL	123
A.2.2	Applicativi di supporto	124
	ACRONIMI	127
	INDICE ANALITICO	129
	BIBLIOGRAFIA	131

ELENCO DELLE FIGURE

Figura 2.1	Un esempio di CTBNC	23
Figura 2.2	Un CTNBC	23
Figura 2.3	Un CTTANBC	24
Figura 4.1	Diagramma dei componenti di RCTBN	47
Figura 5.1	Onda quadra	49
Figura 5.2	Gestione del tempo in CORSIM	56
Figura 5.3	Diagramma dei componenti di CORSIM	58
Figura 5.4	Aggiunta di una RTE a TSIS	60
Figura 5.5	Barra degli strumenti di TShell	60
Figura 5.6	Collegamento delle funzioni della RTE	61
Figura 5.7	Configurazione delle proprietà di CORSIM	62
Figura 5.8	Diagramma delle classi di Sensors DLL	69
Figura 6.1	Rete stradale relativa al dataset #1	73
Figura 6.2	Piano semaforico relativo al dataset #1	74
Figura 6.4	Intersezioni della rete stradale del dataset #2	80
Figura 6.5	Rete stradale relativa al dataset #2	82
Figura 6.6	Accuratezza dei classificatori CTBN e CTNB	94
Figura 6.7	Curve ROC relative alla classe 1	98
Figura 6.8	Curve ROC relative alla classe 2	99
Figura 6.9	Curve ROC relative alla classe 3	100
Figura 6.10	Curve ROC relative alla classe 4	101
Figura 6.11	Curve ROC relative alla classe 5	102
Figura 6.12	Curve ROC relative alla classe 6	103
Figura A.1	Esecuzione multipla guidata da file RNS	124

ELENCO DELLE TABELLE

Tabella 5.1	Durata dei passi temporali in FRESIM	55
Tabella 5.2	Ciclo di vita di CORSIM	63
Tabella 5.3	Semantica dell'output di Sensors DLL	70
Tabella 6.1	Sensori del dataset #1	75
Tabella 6.2	Periodi temporali relativi al dataset #1	75
Tabella 6.3	Flussi di ingresso relativi al dataset #1	76
Tabella 6.4	Percentuali di svolta relative al dataset #1	77
Tabella 6.5	Intersezioni relative al dataset #2	78
Tabella 6.6	Sensori relativi al dataset #2	83
Tabella 6.7	Periodi temporali del dataset #2	84
Tabella 6.8	Flussi di ingresso relativi al dataset #2	85

Tabella 6.9	Matrice di confusione multi-classe	90
Tabella 6.10	Esempio di matrice di confusione	90
Tabella 6.11	Accuratezza dei classificatori CTBN e CTNB . . .	93
Tabella A.1	Struttura d'esempio di una CTBN	112
Tabella A.2	Struttura d'esempio di classificatore CTBN . . .	116

ELENCO DEGLI ALGORITMI

Algoritmo 2.1	Apprendimento di un classificatore CTNB . . .	25
Algoritmo 2.2	Apprendimento di un classificatore CTBN . . .	27
Algoritmo 2.3	Inferenza su un classificatore CTBN	31
Algoritmo 3.1	Algoritmo hill climbing	38

ELENCO DEI SORGENTI

Sorgente 4.1	File di descrizione di un pacchetto R	42
Sorgente 5.1	Costrutto per l'esportazione delle funzioni RTE	64
Sorgente 5.2	Esempio di funzione RTE esportata	64
Sorgente 5.3	Importazione delle CORWIN API	64
Sorgente 5.4	Costrutto per l'importazione delle CORSIM API	65
Sorgente 5.5	Importazione di oggetti delle CORSIM API . . .	65
Sorgente 5.6	Rilevazione del passaggio dei veicoli	68
Sorgente 5.7	Formato di output di Sensors DLL	70
Sorgente 6.1	Configurazione RNS del dataset #2	86
Sorgente A.1	Installazione del pacchetto devtools	106
Sorgente A.2	Installazione del pacchetto RCTBN	107
Sorgente A.3	Importazione e serializzazione (dataset #2) . .	107
Sorgente A.4	Caricamento dataset (dataset #2)	109
Sorgente A.5	Calcolo delle statistiche sufficienti (dataset #2)	110
Sorgente A.6	Calcolo delle statistiche sufficienti (dataset #2)	111
Sorgente A.7	Calcolo delle statistiche sufficienti (dataset #2)	111
Sorgente A.8	Calcolo delle statistiche sufficienti (dataset #2)	112
Sorgente A.9	Stima dei parametri di una CTBN (dataset #2) .	113
Sorgente A.10	Calcolo delle CIM di una CTBN (dataset #2) . .	115
Sorgente A.11	Apprendimento di un CTBNC (dataset #2) . . .	117
Sorgente A.13	Classificazione di un'istanza (dataset #2) . . .	120
Sorgente A.14	Apprendimento della struttura (dataset #2) . .	122
Sorgente A.15	Sostituzione delle classi ai time period	125
Sorgente A.16	Generazione del dataset #2	125
Sorgente A.17	Ottimizzazione del dataset #2	126

ABSTRACT

La maggior parte dei domini e dei sistemi reali evolvono. Essi sono perciò accomunati da una caratteristica: il verificarsi di cambiamenti nel tempo continuo.

Al fine di modellare tali sistemi dinamici come insiemi di processi stocastici a tempo continuo e con spazio degli stati discreto, in questo lavoro di tesi, si presenta il framework delle Continuous time Bayesian Network (CTBN). Tale classe di modelli, sviluppando concetti propri delle Bayesian Network (BN) e dei processi di Markov, permette di rappresentare in modo esplicito l'evoluzione di sistemi dinamici nel tempo continuo senza la necessità di definire a priori una granularità temporale fissa, al contrario di altri modelli già esistenti in letteratura.

Successivamente, si descrive la classe dei Continuous time Bayesian Network Classifier (CTBNC), derivante da tale framework. I CTBNC sono modelli grafico probabilistici finalizzati alla classificazione supervisionata di traiettorie multi-variate che evolvono nel tempo continuo. Quindi, per tale modello si presentano e analizzano i processi di apprendimento e inferenza esatta nel caso di dati completi.

Si descrive, inoltre, il processo di apprendimento strutturale di un generico modello CTBN tramite un approccio basato su punteggio.

L'elaborato prosegue presentando la progettazione e implementazione dell'intero framework delle CTBN come pacchetto R.

Infine, si descrivono i dataset relativi al traffico su rete urbana, generati tramite un apposito software progettato e implementato a tale proposito. Il movente di tale operazione è consistito nell'applicazione di varie istanze di classificatori CTBN (i. e., il Continuous time Naive Bayes Classifier (CTNBC) e 3 classificatori CTBN appresi dai dati variando il numero massimo di genitori per ogni nodo) al problema della classificazione dei profili di traffico; problema per cui vengono quindi presentati i risultati della sperimentazione.

*Le previsioni sono estremamente difficili.
Specialmente sul futuro.*

— Niels Bohr

RINGRAZIAMENTI

Desidero innanzitutto ringraziare il *Prof. Fabio Stella* per avermi concesso la possibilità di svolgere il mio periodo di tesi sotto la sua guida. Ciò mi ha permesso di accedere a un supporto, sia conoscitivo sia morale, che personalmente reputo esser stato di altissimo livello.

Porgo inoltre i miei ringraziamenti anche al correlatore, il *Dott. Daniele Codecasa*. I suoi suggerimenti e, in generale, gli scambi d'opinione intercorsi tra noi, hanno sicuramente aumentato il mio livello di preparazione e di conseguenza il livello di questo lavoro di tesi.

Infine intendo ringraziare vivamente la *mia famiglia*, un dono. Con il passare degli anni e l'aumentare della distanza mi rendo sempre più conto di quanto essi siano tutto ciò che ho e che voglio avere. Grazie.

Milano, settembre 2013

ℒ.

INTRODUZIONE

Il cambiamento è un concetto ubiquo. Non è un'esagerazione pensare a tale concetto come l'unico aspetto costante della realtà. A supporto di tale tesi, si pensi a un qualsiasi insieme di azioni reali. Ad esempio si consideri l'investimento di denaro in azioni societarie, lo svolgimento di una partita di calcio, la cottura di un pasto o la cura di un paziente. Qualsiasi sia il sistema dinamico che scegliamo di considerare, per comprenderlo e manipolarlo si rende necessario ragionare circa i cambiamenti che avvengono in esso. Si ha infatti la necessità di prevedere con quale probabilità specifici eventi (che determinano il cambiamento dello stato del sistema) avvengono. La natura di ogni cambiamento è spesso determinata da molti fattori che sono, a loro volta, cangianti. Si ipotizzi, ad esempio, di voler predire il momento in cui un individuo troverà occupazione, ebbene tale evento è sicuramente influenzato dallo stato dell'economia locale in cui l'individuo vive, così come dalla sua stessa situazione finanziaria. Allo stesso modo, per predire il tempo necessario affinché dei farmaci somministrati ad un paziente sortiscano il loro effetto è necessario considerare le tempistiche e le modalità con cui il paziente si è cibato. È quindi evidente come nel concetto di cambiamento sia connaturato il concetto di tempo. Interrogarsi circa l'avvenire dei cambiamenti nei sistemi citati, sebbene essi riguardino una vasta gamma di problemi, corrisponde ad analizzare le distribuzioni temporali degli eventi.

Per rispondere ad interrogativi di questo tipo, in letteratura si è solitamente ricorso all'utilizzo di modelli basati sui processi di Markov (Lando, 1998; Duffie *et al.*, 1996). Anche se tali approcci funzionano bene, essi hanno una limitazione: non permettono la specifica di modelli con un spazio degli stati strutturato in cui alcune variabili non dipendano direttamente da altre. Ne consegue che con tali approcci non è possibile modellare la distribuzione temporale che rappresenta la velocità d'effetto di un farmaco condizionatamente alla velocità con cui esso raggiunge il sangue del paziente, variabile la cui distribuzione nel tempo può essere a sua volta dipendente dalle tempistiche con cui si verificano altri fenomeni, come per esempio l'assunzione di cibo da parte del paziente.

Le Bayesian Network (BN) (Pearl, 1988) permettono di rappresentare un dominio come uno spazio degli stati strutturato. Esse codificano esplicitamente le dipendenze tra le variabili di un sistema e sfruttano le indipendenze al fine di semplificare la complessità computazionale del modello che lo descrive. Tuttavia esse sono limitate alla rappresentazione di processi statici. Da ciò consegue che non possono essere utilizzate per rispondere direttamente a interrogativi

che riguardino i cambiamenti temporali di un sistema.

Le Dynamic Bayesian Networks (DBN) costituiscono l'estensione temporale delle Bayesian Network (Dean e Kanazawa, 1989). Esse modellano la dinamica di un sistema discretizzando il tempo, suddividendolo così in un numero prefissato di intervalli temporali. Per ognuno di tali intervalli temporali, le Dynamic Bayesian Networks rappresentano tramite una Bayesian Network la distribuzione delle transizioni di stato che le variabili del sistema effettuano. Nel concreto le Dynamic Bayesian Networks (DBN) fotografano lo stato di un sistema a differenti punti nel tempo, tutti equidistanti tra di essi. Poiché tale strumento non rappresenta il tempo in modo esplicito risulta perciò difficile poterlo utilizzare per interrogarci circa l'occorrenza, in un qualsiasi momento temporale, di specifici eventi nel sistema che esso modella. Una ulteriore limitazione di tale approccio è costituita dalla granularità temporale fissata a priori con cui le Dynamic Bayesian Networks modellano l'evoluzione dei sistemi dinamici. Nel caso in cui si intenda modellare un sistema i cui processi evolvono con differenti granularità temporali tramite le Dynamic Bayesian Networks, è necessario rappresentare e vincolare l'intero sistema alla minore di tali granularità temporali. Inoltre, nel caso in cui le osservazioni di cui si dispone avvengano a intervalli di tempo irregolari, il modello delle Dynamic Bayesian Networks non prevede l'esclusione degli intervalli di tempo in cui non è stata ottenuta alcuna osservazione.

Tali premesse costituiscono il movente di questo lavoro di tesi, il cui obiettivo principale è presentare un framework alternativo, le Continuous time Bayesian Network (CTBN) (Nodelman, 2007). La peculiarità di questo framework consiste nella sua abilità di rappresentare in modo esplicito l'evoluzione di sistemi dinamici nel tempo continuo, sorpassando così i vincoli dell'approccio utilizzato dalle Dynamic Bayesian Networks. Infatti, un modello Continuous time Bayesian Network è in grado di adattare i parametri delle distribuzioni che rappresenta e la struttura di dipendenze che codifica alle granularità temporali relative all'evoluzione delle diverse variabili che lo compongono. L'approccio che questo modello grafico probabilistico utilizza al fine di raggiungere tale scopo fonde idee provenienti dalle Bayesian Network e dalla teoria dei processi di Markov.

Il presente elaborato è quindi composto dalla descrizione dei contributi sviluppati attorno alle Continuous time Bayesian Network. Oltre alla descrizione di tale framework, finalizzato alla rappresentazione strutturata di processi di Markov che evolvono nel tempo continuo, per il quale si è progettato e implementato un pacchetto R, questo lavoro di tesi apporta principalmente i seguenti contributi:

- descrizione, a partire dalle CTBN, di una nuova *classe di classificatori*, i Continuous time Bayesian Network Classifier (CTBNC) (Stella e Amer, 2012)

- descrizione del processo di *classificazione supervisionata* di un classificatore Continuos time Bayesian Network (CTBNC) e del processo di *apprendimento strutturale* di una CTBN; processi anch'essi implementati nel succitato pacchetto R
- progettazione e implementazione di un software per il monitoraggio e *tracciamento* del passaggio di *veicoli* su reti stradali tramite sensori; sistema utile alla generazione di *dataset* per il framework delle CTBN
- applicazione del classificatore CTBN (CTBNC) al problema della *classificazione dei profili di traffico*.

Di seguito si passano in rassegna, capitolo per capitolo, gli argomenti affrontati.

NEL PRIMO CAPITOLO si introducono in primis i *fondamenti teorici* su cui il framework delle Continuos time Bayesian Network (CTBN) è basato. Successivamente si descrivono i *concetti* e gli *strumenti* da cui tale classe di modelli grafico probabilistici è costituita: *statistiche sufficienti* e *matrici di intensità condizionali*. Si affronta quindi il processo di *apprendimento dei parametri* delle CTBN da *dati completi* e, infine, il calcolo della *likelihood* di una CTBN rispetto a un insieme di *dati completi*.

NEL SECONDO CAPITOLO si affronta il problema della *classificazione supervisionata* di traiettorie multi-variate di variabili discrete a *tempo continuo*. A tal scopo si descrive una nuova classe di modelli, la classe dei classificatori Continuos time Bayesian Network (CTBNC), derivata dalle CTBN, e se ne presenta in particolare una sua specializzazione: i classificatori CTNB. Viene illustrato sia un algoritmo generale per l'*apprendimento dei parametri* dei classificatori CTBN, sia un algoritmo per l'*inferenza* della classe da associare a dei *dati completi* dato a un modello di classificatore CTBN (CTBNC) precedentemente appreso.

NEL TERZO CAPITOLO si affronta il problema dell'*apprendimento strutturale* di una CTBN da *dati completi*, per il quale si presenta un approccio risolutivo basato su punteggio. Viene presentata una funzione che associa uno *score bayesiano* ad ogni struttura rispetto a dei dati di addestramento. Infine viene descritto il funzionamento di una *procedura di ottimizzazione* (i.e., nello specifico si descrive l'algoritmo *hill climbing*) finalizzata alla ricerca di una struttura che massimizzi la funzione di punteggio (i.e., *score bayesiano*).

NEL QUARTO CAPITOLO si presentano le caratteristiche di RCTBN, il *pacchetto R* sviluppato al fine di implementare il framework delle CTBN e gli algoritmi (*apprendimento dei parametri* e *apprendimento*

strutturale di una CTBN; *classificazione supervisionata* di un CTBNC) presentati nei precedenti capitoli. Inoltre, vengono introdotti gli ulteriori moduli o pacchetti sviluppati con lo scopo di rendere maggiormente completo e snello l'intero processo di *utilizzo del framework* delle CTBN. Ci si riferisce, ad esempio, al pacchetto sviluppato per l'esecuzione di cross-validazione del succitato modello di classificazione supervisionata.

NEL QUINTO CAPITOLO si introduce il problema di *ottimizzazione del traffico urbano*. Successivamente si presentano le caratteristiche e il funzionamento di Traffic Software Integrated System (TSIS), il sistema commerciale utilizzato per creare e simulare *modelli di traffico*. Infine si descrive Sensors DLL, estensione a tempo d'esecuzione di TSIS appositamente sviluppata al fine di monitorare e tracciare il passaggio dei veicoli sulle reti stradali tramite *sensori*. Lo scopo di tale applicativo è la generazione di *dataset* sottoponibili agli algoritmi di classificazione e apprendimento strutturale delle CTBN. Tale passo è infatti propedeutico alla valutazione del processo di *classificazione dei profili di traffico* tramite classificatori CTBN.

NEL SESTO CAPITOLO si presentano le varie configurazioni dei 2 *modelli di traffico*, uno fittizio e uno che rispecchia una rete stradale reale (quella circostante *Viale C. Battisti, 20900 Monza, MB - Italia*), creati tramite TSIS e simulati tramite il relativo simulatore, CORSIM. Viene quindi fornita una descrizione dettagliata dei relativi *dataset* generati dai succitati modelli di traffico tramite Sensors DLL. Per tutti i dataset generati, vengono presentati e comparati i risultati ottenuti utilizzando i classificatori CTBN (CTBNC) per la *classificazione dei profili di traffico*.

NEL SETTIMO CAPITOLO si riportano sia le *conclusioni* tratte dalla sperimentazione attuata e riportata nel capitolo precedente, sia una serie di *considerazioni* generali sui metodi e modelli presentati e implementati.

L'APPENDICE A offre le *guide all'utilizzo* degli strumenti sviluppati a corredo di questo lavoro di tesi. In una prima parte viene presentato, tramite esempi, il funzionamento di RCTBN, il pacchetto R che implementa quanto trattato del framework CTBN. Nella seconda ed ultima parte viene invece presentata la generazione di dataset tramite l'esecuzione di Sensors DLL su modelli di traffico TSIS.

In questo capitolo si introducono i concetti fondamentali relativi alle Continuous time Bayesian Network (CTBN). Le CTBN sono un framework capace di modellare processi stocastici a tempo continuo e con spazio degli stati discreto.

Prima di affrontare tale argomento si presentano alcuni concetti propedeutici a questo lavoro di tesi: le Bayesian Network (BN) e i processi di Markov (sezione 1.1).

1.1 FONDAMENTI

Le Continuous time Bayesian Network utilizzano concetti e idee provenienti da teorie afferenti l'area statistica e del machine learning. Al fine di conferire alla discussione sulle CTBN un quadro iniziale completo ed esauriente, si presentano quindi gli aspetti di maggior rilievo di tali argomenti.

BAYESIAN NETWORK

Le Continuous time Bayesian Network utilizzano una rappresentazione strutturata dello spazio degli stati propria della teoria delle Bayesian Network. Ne ereditano perciò gli aspetti chiave (e.g., indipendenza condizionale) nonché l'insieme delle tecniche algoritmiche per l'apprendimento e l'inferenza.

PROCESSI DI MARKOV

Le Continuous time Bayesian Network descrivono la dinamica evolutiva di variabili casuali tramite un insieme di processi di Markov condizionali.

1.1.1 Bayesian Network

Una Bayesian Network è un modello grafico probabilistico costituito da un grafo aciclico orientato (DAG)¹. I nodi di tale grafo rappresentano un insieme di variabili casuali mentre gli archi evidenziano le dipendenze (e le indipendenze) condizionali fra esse (Korb e Nicholson, 2011). Una BN rappresenta la distribuzione di probabilità congiunta del suo insieme di variabili casuali tramite la distribuzione di probabilità condizionale di ognuna di essa (si veda l'equazione 1.2). Le

¹ Un grafo aciclico orientato (anche detto grafo aciclico diretto o digrafo aciclico) è un tipo di grafo che non ammette cicli ed i cui archi sono orientati: comunque si scelga un vertice non è possibile tornare ad esso percorrendo gli archi del grafo.

BN sono quindi modelli grafico probabilistici con cui è possibile modellare in modo probabilistico le relazioni causali tra variabili. Esse risultano molto utili nella rappresentazione e analisi di domini caratterizzati da incertezza. Sono infatti usate in svariate applicazioni di supporto alle decisioni, bioinformatica, biologia computazionale, data mining, information retrieval e classificazione.

Rappresentazione

Di seguito si fornisce la definizione formale delle Bayesian Network e si introducono i loro aspetti basilari.

Definizione 1.1 (Bayesian Network). Una Bayesian Network \mathcal{B} è una coppia $\mathcal{B} = (\mathcal{G}, \theta_{\mathcal{G}})$ costituita da:

- $\mathcal{G} = (\mathbf{V}(\mathcal{G}), \mathbf{A}(\mathcal{G}))$, un grafo aciclico orientato dove:
 - $\mathbf{V}(\mathcal{G}) = \{V_1, \dots, V_n\}$ è l'insieme dei nodi, ognuno dei quali è associato ad una distribuzione di probabilità condizionale (CPD)²
 - $\mathbf{A}(\mathcal{G}) \subseteq \mathbf{V}(\mathcal{G}) \times \mathbf{V}(\mathcal{G})$ è l'insieme degli archi fra i nodi $\mathbf{V}(\mathcal{G})$
- $\theta_{\mathcal{G}}$, insieme delle CPD dei nodi che specifica $\mathbf{P}_{\mathcal{B}}$, la distribuzione di probabilità congiunta delle variabili casuali $\mathbf{X}_{\mathbf{V}(\mathcal{G})}$ a cui corrispondono i nodi $\mathbf{V}(\mathcal{G})$.

Osservazione 1.1.1. Ogni nodo di una BN è condizionalmente indipendente (si veda la definizione 1.2) dai suoi non-discendenti dati i suoi nodi genitori.

La CPD di ogni variabile casuale $X_i \in \mathbf{X}_{\mathbf{V}(\mathcal{G})}$ esprime i suoi valori di probabilità in funzione dei valori assunti da $\text{Pa}(X_i)$, notazione con cui si denota l'insieme dei nodi genitori per ogni nodo o variabile casuale.

Un arco da un nodo genitore verso un nodo figlio di \mathcal{G} rappresenta una dipendenza diretta fra le corrispettive variabili casuali (si veda Russell e Norvig, 2003, sezione 14.1). I nodi non direttamente connessi rappresentano variabili casuali condizionalmente indipendenti dagli altri nodi (per quanto riguarda il concetto di *indipendenza condizionale* si rimanda alla definizione 1.2).

Prima di procedere con la discussione si introduce la Chain Rule, proprietà fondamentale delle BN.

Teorema 1.1 (Chain Rule). *Dato un insieme di variabili casuali e una distribuzione di probabilità congiunta definita su di esse, è possibile calcolare qualsiasi elemento di tale distribuzione tramite le distribuzioni di probabilità condizionale delle variabili casuali.*

² Nel caso di variabili causali discrete, le CPD sono rappresentabili come delle tabelle che contengono i valori di probabilità di un nodo in funzione di tutte le possibili configurazioni dei nodi genitori (cioè l'insieme dei nodi da cui parte un arco che punta al nodo di interesse). Tali tabelle sono spesso chiamate tabelle di probabilità condizionale (CPT).

Perciò, dato un insieme di variabili casuali A_1, \dots, A_n è possibile calcolare il valore di tale membro della distribuzione di probabilità congiunta applicando la definizione di probabilità condizionale:

$$\mathbf{P}(A_1, \dots, A_n) = \mathbf{P}(A_n | A_{n-1}, \dots, A_1) \cdot \mathbf{P}(A_{n-1}, \dots, A_1).$$

Ripetendo tale processo per ogni termine finale si ottiene:

$$\mathbf{P}\left(\bigcap_{k=1}^n A_k\right) = \prod_{k=1}^n \mathbf{P}\left(A_k | \bigcap_{j=1}^{k-1} A_j\right). \quad (1.1)$$

Applicando l'equazione 1.1 alle Bayesian Network si dice che la distribuzione di probabilità congiunta $\mathbf{P}_{\mathcal{B}}$ si *fattorizza* rispetto al grafo \mathcal{G} se è possibile scrivere:

$$\mathbf{P}_{\mathcal{B}}(X_1, \dots, X_n) = \prod_{i=1}^n \mathbf{P}(X_i | \text{Pa}(X_i)). \quad (1.2)$$

L'equazione 1.2 esprime quindi la *proprietà di fattorizzazione* della distribuzione congiunta del modello grafico, detta *distribuzione di probabilità globale*, ed è ciò che permette di descriverla efficientemente in funzione delle distribuzioni condizionali dei nodi (Russell e Norvig, 2003, sezione 14.2), dette *distribuzioni di probabilità locali*. Questa proprietà contiene in sé il concetto di *proprietà di Markov* (si veda la definizione 1.3), il quale attesta che ogni nodo di una Bayesian Network dipende solo ed esclusivamente dai suoi nodi genitori (Korb e Nicholson, 2011, sottosezione 2.2.4). Si noti inoltre, che le Bayesian Network richiedono (DAG, definizione 1.1) che la loro componente \mathcal{G} non contenga cicli affinché possano rispettare tale proprietà (Russell e Norvig, 2003, sezione 14.1).

Poiché, come detto, una Bayesian Network stabilisce che ogni nodo, dati i suoi genitori, è *condizionalmente indipendente* da ogni altro nodo che non sia un suo discendente, di seguito si introduce tale concetto formalmente.

Definizione 1.2 (Indipendenza condizionale). Un evento A è *condizionalmente indipendente* da un evento B , data l'evidenza su un evento C , qualora la conoscenza di B non apporta alcuna variazione alla probabilità di A rispetto a quella conseguente alla conoscenza di C . Formalmente, ciò significa che:

$$\mathbf{P}(A, B | C) = \mathbf{P}(A | B, C) \cdot \mathbf{P}(B | C) = \mathbf{P}(A | C) \cdot \mathbf{P}(B | C).$$

Da cui segue che:

$$A \perp B | C \iff \mathbf{P}(A | B, C) = \mathbf{P}(A | C).$$

In termini non formali, supponendo di essere nel caso della definizione, cioè di avere una variabile casuale A *condizionalmente indipendente* da B dato C , ciò significa che è possibile ignorare B poiché essa

non ha alcun riflesso sulla distribuzione condizionale di A quando sia noto l'evento C .

Si noti che il concetto appena espresso gioca un ruolo importante per i modelli probabilistici, quali sono le Bayesian Network, semplificando i calcoli richiesti per l'inferenza e l'apprendimento. Le Bayesian Network ereditano questi benefici dell'indipendenza condizionale come conseguenza della loro definizione (a tal riguardo si rimanda all'osservazione 1.1.1). Infatti, la distribuzione condizionale di ogni variabile casuale X_i dipende solo ed esclusivamente dal valore dei suoi genitori, $Pa(X_i)$, mentre ignora completamente i valori dei nodi che non discendono da essa, $Nd(X_i)$.

Grazie alla definizione 1.2 è possibile esprimere in modo formale il concetto appena espresso per ogni nodo $X_i \in \mathbf{X}_{V(G)}$:

$$P(X_i | E, Pa(X_i)) = P(X_i | Pa(X_i)) \quad \forall E \in Nd(X_i),$$

dove $Nd(X_i)$ è l'insieme dei nodi non-discendenti (ed E è una variabile casuale o un insieme di variabili casuali ad essi associati). In base a ciò si dice quindi che le Bayesian Network rispettano l'*assunzione locale di Markov*.

Apprendimento e Inferenza

In questa sezione si descrivono brevemente e a scopo introduttivo i processi di apprendimento e inferenza relativi alle Bayesian Network.

Il problema dell'apprendimento per le Bayesian Network si divide principalmente in due casi:

- apprendere le CPD, nota la struttura
- apprendere sia le CPD sia la struttura (incognita).

In entrambi i casi è di grande aiuto la rappresentazione efficiente delle Bayesian Network che, tramite la *fattorizzazione* della distribuzione di probabilità congiunta, permette di rappresentarla in modo compatto (tramite l'equazione 1.2) riducendo notevolmente il numero di parametri da calcolare.

Come detto, per specificare completamente una Bayesian Network è necessario rappresentare completamente la distribuzione di probabilità congiunta delle sue variabili tramite la distribuzione di probabilità condizionale di ognuna di esse. In generale, tali distribuzioni condizionali possono avere una qualsiasi forma anche se, al fine di semplificare i calcoli, è comune utilizzare distribuzioni discrete o Gaussiane per modellarle. Nel caso in cui i dati siano parzialmente osservabili solitamente si procede tramite l'algoritmo di Expectation Maximization (EM), il quale alterna il calcolo dei valori attesi delle variabili casuali non osservate condizionalmente ai dati osservati con la massimizzazione della likelihood. Tale approccio generalmente converge ai valori di massima probabilità a posteriori per i parametri (si veda Dempster *et al.*, 1977).

Per l'apprendimento dei parametri esistono comunque una varietà di altri approcci possibili (si veda Heckerman, 1996) (e. g., trattare i parametri come variabili casuali sconosciute addizionali) che tuttavia non sono argomento di questo lavoro di tesi.

Si noti che le Bayesian Network non sono solamente un *modello discriminativo ma anche generativo* poiché possono essere utilizzate per soddisfare query arbitrarie, cioè per effettuare *inferenza probabilistica*: calcolare la distribuzione a posteriori di un insieme di variabili casuali data l'osservazione (evidenza) di altre (sfruttando il *teorema di Bayes*). In letteratura (si veda Heckerman, 1996) sono stati esplorati molti metodi di *inferenza esatta*, quali ad esempio l'eliminazione tramite integrazione o somma delle variabili non osservate che non fanno parte della query probabilistica o il metodo *clique tree propagation*. Questi metodi, come gli altri presenti in letteratura, sono sempre esponenziali rispetto al *tree-width*³ del grafo. Per quanto riguarda invece gli algoritmi di *inferenza approssimata* si citano due tra i più comuni: *l'importance sampling* (Shachter e Peot, 1990) e i metodi Markov Chain Monte Carlo (MCMC) (*Gibbs sampling*, *Metropolis sampling*, e *Hybrid Monte Carlo sampling*), basati sul campionamento stocastico (si veda MacKay, 1998; Gilks *et al.*, 1996; S. Geman e D. Geman, 1984).

Nel caso in cui non si disponga della struttura di una BN è richiesto l'*apprendimento strutturale*. Gli algoritmi per l'apprendimento strutturale delle Bayesian Network possono essere divisi in due famiglie.

ALGORITMI BASATI SU VINCOLI

Algoritmi che apprendono la struttura del grafo analizzando le relazioni probabilistiche derivanti dalla proprietà di Markov tramite test di indipendenza condizionale e costruendo un grafo che soddisfi le proprietà di *d-separazione*⁴ corrispondenti. I modelli risultanti sono spesso interpretati come *modelli causali* (Pearl, 1988).

ALGORITMI BASATI SU PUNTEGGIO

Algoritmi che assegnano un punteggio (tramite una funzione di scoring) a tutte le strutture candidate e utilizzando tecniche di ottimizzazione cercano di raggiungere il punteggio massimo. Gli algoritmi di ricerca *greedy* sono la scelta più comune, tuttavia qualsiasi procedura di ricerca può essere usata.

³ In teoria dei grafi, il *tree-width* è un numero associato ad un grafo. Esso corrisponde alla lunghezza minima di tutti i possibili alberi di decomposizione del grafo in esame. La lunghezza di un albero di decomposizione corrisponde alla dimensione massima dei suoi nodi, ognuno dei quali è un sottoinsieme dell'insieme dei vertici del grafo, sottratto 1.

⁴ Un nodo **A** e un nodo **B** sono *d-separati* se per tutti i possibili percorsi che li congiungono esiste un nodo intermedio **V** tale che sussista almeno una delle due seguenti condizioni: **V** è istanziato (i. e., si possiede evidenza su esso) e la connessione è seriale o divergente, oppure **V** è alcuno dei suoi nodi discendenti è istanziato e la connessione è convergente (Jensen e Nielsen, 2007).

Gli *algoritmi basati su vincoli* sono basati sull'algoritmo Inductive Causation (IC) di Verma e Pearl (1991) che fornisce un contesto teorico finalizzato all'apprendimento delle strutture dei modelli causali. L'algoritmo IC può essere riassunto nei tre passi successivi.

- Apprendimento dello scheletro (i. e., grafo non diretto) della rete. Poiché la ricerca esaustiva non è, nella maggior parte dei casi, computazionalmente realizzabile, tutti gli algoritmi di apprendimento restringono la ricerca al *Markov blanket*⁵ di ogni nodo.
- Impostare la direzione degli archi che fanno parte di una *v-structure*⁶.
- Impostare la direzione degli archi fra i nodi rimanenti affinché il vincolo di aciclicità sia rispettato.

Gli *algoritmi basati su punteggio* sono invece delle applicazioni dei vari algoritmi di ricerca euristica (e. g., *hill climbing*, *tabu search*, *best first search*, *simulated annealing*) che utilizzano una funzione di *scoring*. Solitamente la funzione di *scoring* è basata sulla *likelihood*, ovvero sulla la probabilità a posteriori dell'insieme dei dati di apprendimento (i. e., *training set*), data la struttura in esame e i parametri del modello. Tale funzione è spesso *score-equivalent*, affinché reti che definiscono la stessa distribuzione di probabilità abbiano lo stesso score (Chickering, 2013). Tuttavia, per quanto questi algoritmi siano utilizzati molto frequentemente, essi sono esponenziali rispetto al numero di nodi della struttura del grafo. Inoltre, qualora si utilizzi una strategia di ricerca locale, è probabile che l'algoritmo restituisca come risultato un minimo locale. Si fa notare che è possibile ridurre il tempo necessario richiesto per l'apprendimento strutturale fissando un numero massimo di genitori candidati e cercando esaustivamente in insiemi di tale cardinalità una struttura che massimizzi l'informazione mutua fra variabili (Heckerman *et al.*, 1995).

1.1.2 Processi di Markov

Sempre al fine di preparare la discussione delle Continuous time Bayesian Network si prosegue presentando alcuni concetti propedeutici relativi ai processi di Markov, una categoria di processi stocastici con assenza di memoria (Loève, 1978).

Definizione 1.3 (Proprietà di Markov). Secondo la proprietà di Markov gli stati futuri di un processo stocastico sono indipendenti dagli stati passati, avendo evidenza sullo stato presente di tale processo.

⁵ Il *Markov blanket* di un nodo A è un insieme composto dai nodi genitori di A , dai suoi nodi figli e da tutti i nodi che condividono un figlio con A .

⁶ Una *v-structure* è una tripla di nodi $X_j \rightarrow X_i \leftarrow X_k$ incidenti su una connessione convergente.

Formalmente, un processo stocastico X gode di tale proprietà, se e solo se vale la seguente equazione (Loève, 1978):

$$\mathbf{P}(X(t + \Delta t) | X(t), X(s)) = \mathbf{P}(X(t + \Delta t) | X(t)), \quad (1.3)$$

per ogni s , e t tali che $s < t < \infty$.

I modelli che rispettano tale proprietà sono detti modelli che rispettano l'*assunzione di Markov*.

Di conseguenza la distribuzione di probabilità condizionale degli stati futuri di un processo stocastico che gode di tale proprietà è indipendente dagli stati passati dato quello attuale.

In altri termini ciò indica che lo stato futuro di una variabile casuale è *condizionalmente indipendente* (si veda la definizione 1.2) dalla sequenza dei suoi stati passati, avendo evidenza sul suo stato presente.

Dalla proprietà di Markov deriva la definizione dei processi di Markov.

Definizione 1.4 (Processo di Markov). Si definisce (Loève, 1978) come processo di Markov un processo stocastico che gode della proprietà di Markov.

Definizione 1.5 (Catena di Markov). Un processo di Markov che può assumere solo un numero finito di stati è solitamente definito come una *catena di Markov* (si veda Norris, 1998, p. 10).

Esistono due tipi di processi di Markov: omogenei e non. Si procede quindi fornendone le definizioni.

Definizione 1.6 (Processo di Markov omogeneo). Un processo di Markov è detto *omogeneo* qualora $\mathbf{P}(X(t + \Delta t) | X(t))$ non dipenda dal tempo t . Affinché ciò sia vero deve risultare che:

$$\mathbf{P}(X(t + \Delta t) | X(t)) = \mathbf{P}(X(\Delta t) | X(0)). \quad (1.4)$$

Data quindi una variabile casuale X e l'insieme delle sue istanziazioni $\text{val}(X) = \{x_1, \dots, x_J\}$, $X(t)$ è un processo di Markov *omogeneo*, a tempo continuo e stati finiti se e solo se la sua dinamica è definibile in termini di:

- una distribuzione di probabilità iniziale \mathbf{P}_X^0 su $\text{val}(X)$
- una matrice di intensità \mathbf{Q}_X .

Definizione 1.7 (Matrice di intensità). Una matrice di intensità (IM), rappresenta un *modello di transizione Markoviano*:

$$\mathbf{Q}_X = \begin{bmatrix} -\mathbf{q}_{x_1} & \mathbf{q}_{x_1 x_2} & \cdots & \mathbf{q}_{x_1 x_K} \\ \mathbf{q}_{x_2 x_1} & -\mathbf{q}_{x_2} & \cdots & \mathbf{q}_{x_2 x_K} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{q}_{x_K x_1} & \mathbf{q}_{x_K x_2} & \cdots & -\mathbf{q}_{x_K} \end{bmatrix}.$$

Lo scopo di una matrice di intensità è descrivere il comportamento transiente di X , un processo di Markov omogeneo.

Osservazione 1.7.1. L'ordine di una matrice di intensità corrisponde a $K = |\text{val}(X)|$, la cardinalità dell'insieme dei valori assunti da X .

Affinché \mathbf{Q}_X sia una matrice di intensità valida, ogni sua riga deve sommare a 0:

$$\mathbf{q}_{x_i} = \sum_{i \neq j} \mathbf{q}_{x_i x_j} \quad \text{con} \quad \mathbf{q}_{x_i}, \mathbf{q}_{x_i x_j} > 0.$$

Data quindi una matrice di intensità \mathbf{Q}_X essa descrive il comportamento transiente di $X(t)$. Se $X(0) = x_i$, allora il processo di Markov omogeneo (e indicizzato dal tempo t) $X(t)$ rimarrà nello stato x_i una quantità di tempo *esponenzialmente distribuita* rispetto al parametro \mathbf{q}_{x_i} . Di conseguenza la *funzione di densità* f e la corrispondente *funzione di ripartizione*⁷ F sono quelle della distribuzione esponenziale:

$$\begin{aligned} f(t) &= \mathbf{q}_{x_i} e^{-\mathbf{q}_{x_i} t}, \quad t > 0 \\ F(t) &= 1 - e^{-\mathbf{q}_{x_i} t}, \quad t \geq 0. \end{aligned} \tag{1.5}$$

Quando un modello di transizione è definito esclusivamente tramite una matrice di intensità \mathbf{Q}_X si dice che esso usa una *parametrizzazione pura* delle intensità. In tal caso i parametri per un processo di Markov omogeneo con K stati sono $\{\mathbf{q}_{x_i}, \mathbf{q}_{x_i x_j} : 1 \leq i, j \leq K, i \neq j\}$.

Mentre gli elementi sulla diagonale di una matrice di intensità, \mathbf{q}_{x_i} , codificano una quantità che può essere interpretata come la “*probabilità istantanea*” che X abbandoni lo stato x_i , gli elementi non sulla diagonale, $\mathbf{q}_{x_i x_j}$, esprimono l’*intensità di transizione* dallo stato x_i allo stato x_j .

Tuttavia, questa non è l’unica parametrizzazione possibile per un processo di Markov omogeneo. Si noti infatti che la distribuzione di probabilità locale sulle transizioni di X è fattorizzata in due parti.

Definizione 1.8 (Parametrizzazione mista delle intensità). La *parametrizzazione mista* delle intensità per un processo di Markov omogeneo X con K stati è composta da due insiemi di parametri:

$$\begin{aligned} \mathbf{q}_X &= \{\mathbf{q}_{x_i} : 1 \leq i \leq K\}, \\ \boldsymbol{\theta}_X &= \{\boldsymbol{\theta}_{x_i x_j} : 1 \leq i, j \leq K, i \neq j\}. \end{aligned}$$

La semantica di tali insiemi di parametri è la seguente:

- \mathbf{q}_X è un insieme di intensità \mathbf{q}_{x_i} che parametrizzano una *distribuzione di probabilità esponenziale* ed esprimono quando avvengono le transizioni

⁷ Nel calcolo delle probabilità la funzione di ripartizione di una variabile casuale X a valori reali, anche nota come funzione di distribuzione cumulativa, è la funzione che associa a ciascun valore x la probabilità che X assuma valori minori o uguali ad x (i.e., $\mathbf{P}(X \leq x)$).

- θ_X è un insieme di probabilità $\theta_{x_i x_j}$ che rappresentano la *probabilità di transitare* dallo stato x_i allo stato x_j , con $i \neq j$, sapendo che avverrà un salto ad un determinato istante di tempo.

Osservazione 1.8.1. Si osservi che, al di là del tipo di parametrizzazione con cui si sceglie di definire un modello di transizione, il numero di parametri necessari è pari a K^2 sebbene il numero di parametri liberi sia solo $K^2 - K$ (Nodelman, 2007).

Osservazione 1.8.2. Si noti, inoltre, che una parametrizzazione può essere più chiara dell'altra a seconda del processo in cui si è coinvolti, di conseguenza nel prosieguo le si utilizzerà entrambe in modo intercambiabile.

Al fine di correlare questi due tipi di parametrizzazione dei modelli di transizione si riporta il seguente teorema (Nodelman, 2007).

Teorema 1.2. *Dati X e Y , due processi di Markov omogenei con lo stesso spazio degli stati e la stessa distribuzione di probabilità iniziale, se il modello di transizione di X è definito tramite la matrice di intensità Q_X e quello di Y è definito tramite la parametrizzazione mista q_Y, θ_Y , allora X e Y sono stocasticamente equivalenti⁸ solo se:*

$$q_{y_i} = q_{x_i}$$

e

$$\theta_{y_i y_j} = \frac{q_{x_i x_j}}{q_{x_i}}.$$

Osservazione 1.8.3. Si osservi che il teorema 1.2 formalizza la relazione che sussiste fra i parametri q e θ .

Quindi, qualsiasi sia la parametrizzazione utilizzata per rappresentare il modello di transizione di un processo di Markov omogeneo X , è possibile calcolare:

- il *tempo atteso* di una transizione uscente dallo stato x_i

$$1/q_{x_i}$$

- la “*probabilità istantanea*” di transizione dallo stato x_i allo stato x_j sapendo che avverrà un salto ad un determinato istante di tempo

$$\theta_{x_i x_j} = q_{x_i x_j} / q_{x_i}.$$

Infine, si noti che la matrice Q_X fa in modo che X soddisfi la proprietà di Markov poiché il comportamento futuro di X è definito solamente in base al suo stato attuale (vale l'equazione 1.4).

⁸ Due processi di Markov sono detti *stocasticamente equivalenti* se posseggono lo stesso spazio degli stati e le stesse probabilità di transizione (Gihman e Skorohod, 1973).

Definizione 1.9 (Processo di Markov condizionale). Un processo di Markov le cui intensità di transizione variano nel tempo non in funzione del tempo ma in funzione dei valori assunti ad ogni determinato istante t da un insieme di altre variabili, che evolvono anch'esse come dei processi di Markov, è detto essere un processo di Markov condizionale (o processo di Markov non omogeneo).

Assumendo quindi che una variabile casuale X evolva come un processo di Markov $X(t)$ e che la sua dinamica sia condizionata da un insieme di altre variabili casuali $P_a(X)$, anch'esse dei processi di Markov, è possibile definire per tale variabile casuale una matrice di intensità condizionale (CIM) $\mathbf{Q}_{X|P_a(X)}$.

Specificando una distribuzione di probabilità iniziale su X si definisce quindi un processo di Markov il cui comportamento dipende dalle istanziazioni dei valori di $P_a(X)$.

Definizione 1.10 (Matrice di intensità condizionale). Dato un insieme di processi di Markov $P_a(X)$, una matrice di intensità condizionale $\mathbf{Q}_{X|P_a(X)}$ è costituita da un insieme di matrici di intensità $\mathbf{Q}_{X|p_{a_i}(x)}$, una per ogni diversa istanziazione $p_{a_i}(x)$ di $P_a(X)$ (Stella e Amer, 2012):

$$\mathbf{Q}_{X|P_a(X)} = \{ \mathbf{Q}_{X|p_{a_1}(x)}, \mathbf{Q}_{X|p_{a_2}(x)}, \dots, \mathbf{Q}_{X|p_{a_n}(x)} \}.$$

Ogni matrice di intensità di $\mathbf{Q}_{X|P_a(X)}$ è del seguente tipo:

$$\mathbf{Q}_{X|p_{a_i}(x)} = \begin{bmatrix} -q_{x_1}^{p_{a_i}(x)} & q_{x_1 x_2}^{p_{a_i}(x)} & \dots & q_{x_1 x_K}^{p_{a_i}(x)} \\ q_{x_2 x_1}^{p_{a_i}(x)} & -q_{x_2}^{p_{a_i}(x)} & \dots & q_{x_2 x_K}^{p_{a_i}(x)} \\ \vdots & \vdots & \ddots & \vdots \\ q_{x_K x_1}^{p_{a_i}(x)} & q_{x_K x_2}^{p_{a_i}(x)} & \dots & -q_{x_K}^{p_{a_i}(x)} \end{bmatrix}.$$

Di seguito si presenta un breve esempio finalizzato alla comprensione pratica delle matrici di intensità condizionali (CIM) e del loro scopo.

Esempio 1.10.1.

Date due variabili causali, $E(t)$ e $H(t)$, delle quali la prima modella l'eventualità che un individuo stia mangiando o meno (se $e = 2$ allora l'individuo sta mangiando, viceversa se $e = 1$) mentre la seconda modella l'eventualità che lo stesso individuo abbia fame o meno (se $h = 2$ allora l'individuo è affamato, viceversa se $h = 1$) e la matrice di intensità condizionale $\mathbf{Q}_{E|H}$, che è un insieme composto dalle matrici di intensità $\mathbf{Q}_{E|h=1}$ e $\mathbf{Q}_{E|h=2}$, è possibile calcolare la probabilità degli eventi della variabile casuale E condizionatamente all'evidenza che si possiede sulla variabile casuale H .

$$\mathbf{Q}_{E|h=1} = \begin{matrix} & \begin{matrix} 1 & 2 \end{matrix} \\ \begin{matrix} 1 \\ 2 \end{matrix} & \begin{bmatrix} -.01 & .01 \\ 10 & -10 \end{bmatrix} \end{matrix} \quad \mathbf{Q}_{E|h=2} = \begin{matrix} & \begin{matrix} 1 & 2 \end{matrix} \\ \begin{matrix} 1 \\ 2 \end{matrix} & \begin{bmatrix} -2 & 2 \\ .01 & -.01 \end{bmatrix} \end{matrix}.$$

Ipotizzando che l'unità temporale corrisponda a un'ora:

- un individuo affamato ($h = 2$) che non sta mangiando ($e = 1$) inizierà a mangiare in 30 minuti poiché

$$\frac{1}{q_{e=1|h=2}} = \frac{1}{2},$$

- un individuo non affamato ($h = 1$) che sta mangiando ($e = 2$) smetterà di mangiare ($e = 1$) entro 6 minuti poiché

$$\frac{1}{q_{e=2|h=1}} = \frac{1}{10};$$

si osservi che la “*probabilità istantanea*” di transizione da $e = 2$ a $e = 1$ è

$$\theta_{e=2,e=1|h=1} = \frac{q_{e=2,e=1|h=1}}{q_{e=2|h=1}} = \frac{10}{10} = 1,$$

ciò poiché le matrici di intensità hanno dimensione 2×2 e, dovendo ogni loro riga sommare a 0, gli elementi sulla diagonale sono uguali al rispettivo (stessa riga) e unico elemento non sulla diagonale.

1.2 DEFINIZIONI PRELIMINARI

Nelle precedenti sezioni sono stati illustrati i concetti che si pongono a fondamento delle Continuous time Bayesian Network:

- le Bayesian Network: utili a comprendere la rappresentazione strutturata dello spazio degli stati delle CTBN, l'utilizzo della nozione di indipendenza condizionale e le conseguenti tecniche di apprendimento e inferenza
- i processi di Markov, omogenei e non, al fine di introdurre le modalità di rappresentazione (qualitativa e quantitativa) delle CTBN.

Prima di presentare le Continuous time Bayesian Network come una collezione di processi di Markov a tempo continuo non omogenei e con spazio degli stati discreto (Nodelman, 2007), si forniscono alcune definizioni utili per il prosieguo della discussione.

Definizione 1.11 (Variabile di processo). Una variabile di processo X , anche detta Process Variable (PV) (Nodelman, 2007), è un insieme di processi di Markov a tempo continuo $X(t)$.

Definizione 1.12 (Traiettoria). Istanziamento di un insieme di valori per $X(t)$ al variare di t .

Definizione 1.13 (J-time-segment). Partizionamento di un intervallo temporale $[0, T)$ in J intervalli chiusi a sinistra:

$$[0, t_1); [t_1, t_2); \dots; [t_{J-1}, T).$$

Nota 1.13.1. È possibile riferirsi a tale concetto anche tramite l'espressione "insieme dei segmenti temporali".

Definizione 1.14 (J-evidence-stream). Data una variabile di processo \mathbf{X} composta da N variabili casuali e un insieme di segmenti temporali composto da J intervalli, un *J-evidence-stream* è l'insieme delle istanziazioni comuni $\mathbf{X} = \mathbf{x}$ associate ad ogni intervallo temporale per ogni sottoinsieme delle variabili casuali (Stella e Amer, 2012). È denotato con $(\mathbf{X}^1 = \mathbf{x}^1, \mathbf{X}^2 = \mathbf{x}^2, \dots, \mathbf{X}^J = \mathbf{x}^J)$, o più concisamente con $(\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J)$.

Nota 1.14.1. È possibile riferirsi a tale concetto anche tramite l'espressione "flusso di evidenze".

Nota 1.14.2. Un flusso di evidenze $(\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J)$ è detto essere *completamente osservato* se lo stato di tutte le variabili $X_n \in \mathbf{X}$ è conosciuto in tutto l'intervallo $[0, T)$. Viceversa, un flusso di evidenze è detto *parzialmente osservato*.

1.3 RAPPRESENTAZIONE

Una Continuous time Bayesian Network è un modello grafico in cui ogni nodo rappresenta una variabile casuale i cui stati evolvono in modo continuo nel tempo. Le dinamiche evolutive degli stati dei nodi sono governate e dipendono dal valore che gli stati dei nodi padre⁹ assumono (Nodelman, 2007). Quindi ogni nodo è un processo di Markov condizionale (si veda la definizione 1.9) a tempo continuo e spazio degli stati discreto.

Una CTBN è composta principalmente da due componenti:

- una distribuzione di probabilità iniziale
- le componenti che regolano l'evoluzione nel tempo del sistema

Più formalmente si definisce:

Definizione 1.15 (Continuous time Bayesian Network). Data una variabile di processo \mathbf{X} , insieme di processi di Markov X_1, X_2, \dots, X_N a tempo continuo e con spazio degli stati finito $\text{val}(X_n) = \{x_1, \dots, x_J\}$ (dove $n = 1, \dots, N$), una CTBN \mathcal{N} su \mathbf{X} consiste di:

- una distribuzione di probabilità iniziale $\mathbf{P}_{\mathbf{X}}^0$ specificata come una Bayesian Network \mathcal{B} su \mathbf{X}

⁹ Con il termine "nodo padre", o *parent node*, si intende un nodo il cui stato condiziona quello di un altro nodo del modello grafico.

- un modello di transizione a tempo continuo, specificato da:
 - un grafo \mathcal{G} , orientato e non necessariamente aciclico, composto dai nodi X_1, X_2, \dots, X_N , ognuno dei quali possiede un insieme di genitori denotato da $\text{Pa}(X_n)$
 - una matrice di intensità condizionale $\mathbf{Q}_{X_n | \text{Pa}(X_n)}$ per ogni nodo $X_n \in \mathbf{X}$.

Per ogni variabile causale $X_n \in \mathbf{X}$ di \mathcal{N} si ha quindi un insieme di modelli di probabilità locali: $\mathbf{Q}_{X_n | \text{Pa}(X_n)}$, la CIM di X_n , è infatti un insieme di modelli di transizione Markoviani la cui cardinalità è pari a quella dell'insieme delle diverse istanziazioni di $\text{Pa}(X_n)$.

Si riscontra, quindi, quanto già affermato in precedenza (si veda la sezione 1.1 a pagina 1), cioè che una CTBN esprime la sua dinamica evolutiva globale tramite un unico processo di Markov omogeneo, costituito da un insieme di processi di Markov condizionali (un insieme di CIM e relative distribuzioni di probabilità iniziali).

Si noti che, diversamente dalle Bayesian Network, nelle Continuous time Bayesian Network gli archi fra i nodi rappresentano le dipendenze nel tempo. Per tale motivo è possibile che la componente \mathcal{G} del modello di transizione continuo contenga dei cicli. Tra l'altro, come vedremo nel prosieguo, la mancanza di tale vincolo di aciclicità porta a notevoli vantaggi computazionali relativamente all'apprendimento della struttura di una CTBN dai dati.

1.4 APPENDIMENTO

In questa sezione si argomenta sulla probabilità di un *insieme di dati completo* rispetto a una Continuous time Bayesian Network. A tal fine si mostra come una CTBN possa essere decomposta in un aggregato di modelli di probabilità locali relativi alle singole variabili casuali e espressa in termini di *statistiche sufficienti* aggregate.

Si affronta infine il processo di apprendimento dei parametri delle Continuous time Bayesian Network da *dati completi*. I processi di apprendimento relativi a dati non completi sono tralasciati poiché non facenti parte degli argomenti di questo lavoro di tesi.

Definizione 1.16 (Insieme di dati completo). Dato un insieme di variabili casuali, un insieme di dati $\mathcal{D} = \{\delta_1, \dots, \delta_h\}$ si dice *completo* se ogni δ_i (con $i = 1, \dots, h$) è un insieme di traiettorie completamente osservate delle variabili casuali (i.e., l'istanziamento di tutte le variabili casuali è osservabile per ogni istante temporale di ogni traiettoria).

1.4.1 Statistiche sufficienti

Le *statistiche sufficienti* per un singolo processo di Markov omogeneo $X(t)$ riassumono la sua dinamica evolutiva con:

- $T[x]$: la quantità di tempo trascorsa nello stato x
- $M[x, x']$: il numero di transizioni dallo stato x allo stato x' .

Il numero totale di transizioni uscenti da uno stato x è:

$$M[x] = \sum_{x'} M[x, x'].$$

Nel caso di un processo di Markov condizionale è invece necessario considerare anche l'istanziamento dell'insieme $Pa(X)$ dei nodi genitori:

- $T[x | pa_i(x)]$: la quantità di tempo trascorsa nello stato x quando $Pa(X) = pa_i(x)$
- $M[x, x' | pa_i(x)]$: il numero di transizioni dallo stato x allo stato x' quando $Pa(X) = pa_i(x)$.

Chiaramente, il numero totale di transizioni si calcola come sopra.

1.4.2 Likelihood

Al fine di presentare il calcolo della likelihood¹⁰ di una CTBN rispetto a un dataset completo $\mathcal{D} = \{\delta_1, \dots, \delta_h\}$ è bene procedere per gradi e iniziare presentando dapprima la likelihood di una singola transizione di un singolo processo di Markov omogeneo $X(t)$.

Likelihood di una singola transizione

Data una tripla $d = \langle x_d, t_d, x_{d'} \rangle \in \delta$, la quale esprime una transizione di $X(t)$ da x_d a $x_{d'}$ dopo che esso ha trascorso t_d tempo in x_d , è possibile scrivere la likelihood di questa singola transizione d in funzione dei parametri:

$$\begin{aligned} L_X(\mathbf{q}, \boldsymbol{\theta} : d) &= L_X(\mathbf{q} : d) \cdot L_X(\boldsymbol{\theta} : d) \\ &= \mathbf{q}_{x_d} e^{-\mathbf{q}_{x_d} t_d} \cdot \boldsymbol{\theta}_{x_d x_{d'}}. \end{aligned} \quad (1.6)$$

Si noti che l'equazione 1.6 è ricavata moltiplicando la *funzione di distribuzione di probabilità* di $X(t)$ (equazione 1.5) per la “*probabilità istantanea*” di transizione (si veda la definizione 1.7).

¹⁰ La likelihood di un insieme di valori dei parametri, dato un insieme di dati, corrisponde alla probabilità dell'insieme dei dati, dati tali valori dei parametri.

Likelihood di un dataset completo

Poiché tutte le transizioni sono osservabili, la *likelihood* del dataset \mathcal{D} può essere decomposta come un prodotto delle likelihood individuali di ogni singola transizione d (si veda Nodelman *et al.*, 2002, p. 3). Per tale motivo \mathcal{D} è sintetizzabile aggregando le *statistiche sufficienti* relative a ogni processo di Markov condizionale di una CTBN.

Quindi la likelihood di un dataset completo \mathcal{D} rispetto a un singolo processo di Markov omogeneo $X(t)$ è:

$$\begin{aligned} L_X(\mathbf{q}, \boldsymbol{\theta} : \mathcal{D}) &= \left[\prod_{d \in \mathcal{D}} L_X(\mathbf{q} : d) \right] \left[\prod_{d \in \mathcal{D}} L_X(\boldsymbol{\theta} : d) \right] \\ &= \left[\prod_x (\mathbf{q}_x)^{M[x]} e^{-\mathbf{q}_x^T [x]} \right] \left[\prod_x \prod_{x' \neq x} (\theta_{xx'})^{M[x, x']} \right]. \end{aligned} \quad (1.7)$$

Si supponga ora di traslare questo concetto a una Continuous time Bayesian Network \mathcal{N} con N nodi: per ogni nodo X_i , con $i = 1, \dots, N$ è necessario considerare tutte le transizioni contestualmente all'istanziamento dell'insieme $\text{Pa}(X_i)$ dei suoi nodi genitori. Poiché, nel caso di *dati completi*, si conosce sempre l'istanziamento di $\text{Pa}(X_i)$, allora, per ogni istante di tempo t , si conosce quale matrice di intensità $\mathbf{Q}_{X_i | \text{Pa}_i(x)}$, con $\text{Pa}_i(x) \in \text{Pa}(X_i)$, governi la dinamica di X_i .

Perciò la probabilità dei dati \mathcal{D} rispetto a \mathcal{N} è il prodotto delle likelihood di ogni variabile X_i :

$$\begin{aligned} L_{\mathcal{N}}(\mathbf{q}, \boldsymbol{\theta} : \mathcal{D}) &= \prod_{X_i \in \mathbf{X}} L_{X_i}(\mathbf{q}_{X_i | \text{Pa}(X_i)}, \boldsymbol{\theta}_{X_i | \text{Pa}(X_i)} : \mathcal{D}) \\ &= \prod_{X_i \in \mathbf{X}} L_{X_i}(\mathbf{q}_{X_i | \text{Pa}(X_i)} : \mathcal{D}) L_{X_i}(\boldsymbol{\theta}_{X_i | \text{Pa}(X_i)} : \mathcal{D}). \end{aligned} \quad (1.8)$$

Il termine $L_X(\boldsymbol{\theta}_{X | \text{Pa}(X)} : \mathcal{D})$ esprime la likelihood delle transizioni tra stati. Tale termine trascura il tempo che intercorre fra le transizioni poiché esse dipendono esclusivamente dal valore di nodi genitori (si veda Nodelman *et al.*, 2002, p. 3). Quindi, usando le *statistiche sufficienti* si può scrivere:

$$L_X(\boldsymbol{\theta}_{X | \text{Pa}(X)} : \mathcal{D}) = \prod_{\text{Pa}_i(x)} \prod_x \prod_{x' \neq x} (\theta_{xx' | \text{Pa}_i(x)})^{M[x, x' | \text{Pa}_i(x)]}. \quad (1.9)$$

Per quanto riguarda il calcolo di $L_X(\mathbf{q}_{X | \text{Pa}(X)} : \mathcal{D})$ va considerato il caso in cui il tempo trascorso da X in uno determinato stato x termini non a causa di una sua transizione bensì a causa di una transizione di uno o più nodi appartenenti all'insieme dei suoi nodi genitori (i.e., una nuova istanziamento per l'insieme dei genitori $\text{Pa}(X)$). È quindi necessario considerare la probabilità che il nodo X rimanga in x una quantità di tempo *almeno pari* a t mentre i suoi nodi genitori $\text{Pa}(X)$ non effettuano alcuna transizione di stato (si veda Nodelman *et al.*,

2002, p. 3). Tale quantità si ricava dalla funzione di distribuzione cumulativa di una distribuzione esponenziale (equazione 1.5):

$$1 - F(t) = e^{-\mathbf{q}_{\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})}t}.$$

Perciò la likelihood delle quantità di tempo trascorse in ogni stato è:

$$L_X(\mathbf{q}_{X|\mathbf{p}\mathbf{a}(X)} : \mathcal{D}) = \prod_{\mathbf{p}\mathbf{a}_i(\mathbf{x})} \prod_{\mathbf{x}} (\mathbf{q}_{\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})})^{M[\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})]} e^{-\mathbf{q}_{\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})} T[\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})]}.$$
(1.10)

Combinando l'equazione 1.10 e l'equazione 1.9 si ottiene la likelihood di un dataset completo \mathcal{D} rispetto a un singolo processo di Markov condizionale:

$$L_X(\mathbf{q}, \boldsymbol{\theta} : \mathcal{D}) = \prod_{\mathbf{p}\mathbf{a}_i(\mathbf{x})} \prod_{\mathbf{x}} \left[(\mathbf{q}_{\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})})^{M[\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})]} e^{-\mathbf{q}_{\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})} T[\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})]} \cdot \prod_{\mathbf{x} \neq \mathbf{x}'} (\boldsymbol{\theta}_{\mathbf{x}\mathbf{x}'|\mathbf{p}\mathbf{a}_i(\mathbf{x})})^{M[\mathbf{x}, \mathbf{x}'|\mathbf{p}\mathbf{a}_i(\mathbf{x})]} \right].$$
(1.11)

Si noti che, dal punto di vista algebrico, è conveniente riformulare l'equazione 1.11 come *log-likelihood*:

$$\ell_X(\mathbf{q}, \boldsymbol{\theta} : \mathcal{D}) = \sum_{\mathbf{p}\mathbf{a}_i(\mathbf{x})} \sum_{\mathbf{x}} \left[M[\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})] \ln(\mathbf{q}_{\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})}) - \mathbf{q}_{\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})} T[\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})] + \sum_{\mathbf{x} \neq \mathbf{x}'} M[\mathbf{x}, \mathbf{x}'|\mathbf{p}\mathbf{a}_i(\mathbf{x})] \ln(\boldsymbol{\theta}_{\mathbf{x}\mathbf{x}'|\mathbf{p}\mathbf{a}_i(\mathbf{x})}) \right].$$
(1.12)

È ora possibile asserire che la *log-likelihood* di \mathcal{N} (dall'equazione 1.8) è:

$$\ell_{\mathcal{N}}(\mathbf{q}, \boldsymbol{\theta} : \mathcal{D}) = \sum_{\mathbf{X}_i \in \mathbf{X}} \ell_{\mathbf{X}_i}(\mathbf{q}, \boldsymbol{\theta} : \mathcal{D}).$$
(1.13)

In questa sezione si è presentato come computare la likelihood di un modello di una CTBN rispetto a un dataset completo.

Tuttavia, nel caso in cui non si conoscano i parametri di una CTBN è necessario stimarli. Nella prossima sezione viene affrontato esattamente questo argomento.

1.4.3 Stima dei parametri

Si affronta ora il problema dell'apprendimento dei parametri di una Continuous time Bayesian Network (con struttura nota \mathcal{G}) da un insieme di dati completi (si veda Nodelman, 2007, sezione 5.1).

È possibile affrontare tale problema principalmente in due modi. La scelta più semplice consiste nell'effettuare una stima dei parametri del modello tramite un approccio *maximum-likelihood*. Tuttavia, è noto che tale approccio può portare a problemi con l'inferenza quando si possiedono pochi dati di input. Per evitare tale limitazione solitamente si ricorre al secondo approccio, la *regolarizzazione bayesiana* dei parametri: si sceglie una distribuzione a priori per i parametri e li si aggiorna in accordo ai dati di input.

La stima dei parametri non è un processo fine a se stesso, in quanto, da essi è possibile costruire le matrici di intensità condizionali (CIM) di ogni nodo della CTBN. Come si ricorderà, una CIM è un insieme di matrici di intensità, una per ogni istanziazione $pa_i(x)$ dei nodi genitori (si veda la definizione 1.10). Perciò, fissato $pa_i(x)$, si può computare la rispettiva matrice di intensità per un nodo qualsiasi ponendo sulla diagonale il rispettivo vettore dei parametri $q_{x|pa_i(x)}$ e ricavando i valori non sulla diagonale dalla relazione (si veda il teorema 1.2) fra i parametri q e θ :

Come costruire una CIM dai parametri.

$$q_{xx'|pa_i(x)} = \theta_{xx'|pa_i(x)} \cdot q_{x|pa_i(x)}. \quad (1.14)$$

Infine, come vedremo in seguito nel capitolo 3, i parametri sono anche un componente chiave del processo di apprendimento strutturale.

Stima maximum-likelihood

In base a quanto attestato dalla definizione stessa delle CTBN (definizione 1.15 a pagina 12), la dinamica evolutiva globale di una CTBN, cioè la dinamica di tutti i nodi di \mathcal{G} (dei processi di Markov condizionali indicizzati dal tempo), è espressa tramite un processo di Markov omogeneo. Dalla definizione 1.7, inoltre, si deduce che tale processo di Markov induce un modello di probabilità composto da una *distribuzione esponenziale* con parametro $q_{x|pa_i(x)}$, che esprime il tempo trascorso in uno stato x da un nodo X data una istanziazione $pa_i(x)$ per i nodi genitori $Pa(X)$, e una *distribuzione multinomiale* con parametro $\theta_{xx'|pa_i(x)}$, che esprime la probabilità di transizione uscenti dallo stato x verso x' (sempre fermo restando il condizionamento dato dall'istanziamento dei nodi genitori).

La media della distribuzione esponenziale in questione è pari a $1/q_{x|pa_i(x)}$. Questa quantità esprime il tempo medio delle transizioni uscenti da uno stato x , fermo restando che il genitore del nodo in questione abbia istanziazione costante e uguale a $pa_i(x)$. Poiché il tempo medio si calcola rapportando il tempo totale trascorso in x , $T[x|pa_i(x)]$, rispetto al numero totale di transizioni uscenti da x , $M[x|pa_i(x)]$, si ottiene:

$$\frac{1}{q_{x|pa_i(x)}} = \frac{T[x|pa_i(x)]}{M[x|pa_i(x)]}.$$

Invece, la probabilità di transizione da uno stato x verso x' sapendo che avverrà una transizione è data dal rapporto tra il numero totale di transizioni da x a x' diviso il numero totale di transizioni uscenti da x ; cioè:

$$\frac{M[x, x'|pa_i(x)]}{M[x|pa_i(x)]}.$$

Teorema 1.3. *Parametri maximum-likelihood (MLE). I parametri che massimizzano la likelihood (equazione 1.13) di una Continuous time Bayesian Network sono funzione delle statistiche sufficienti:*

$$\begin{aligned} \mathbf{q}_{\mathbf{x} | \mathbf{p} \mathbf{a}_i(\mathbf{x})} &= \frac{M[\mathbf{x} | \mathbf{p} \mathbf{a}_i(\mathbf{x})]}{T[\mathbf{x} | \mathbf{p} \mathbf{a}_i(\mathbf{x})]} \\ \theta_{\mathbf{x} \mathbf{x}' | \mathbf{p} \mathbf{a}_i(\mathbf{x})} &= \frac{M[\mathbf{x}, \mathbf{x}' | \mathbf{p} \mathbf{a}_i(\mathbf{x})]}{M[\mathbf{x} | \mathbf{p} \mathbf{a}_i(\mathbf{x})]}. \end{aligned} \quad (1.15)$$

Si noti che, in questo caso (*dataset completo*), $\mathbf{q}_{\mathbf{x} | \mathbf{p} \mathbf{a}_i(\mathbf{x})}$ e $\theta_{\mathbf{x} \mathbf{x}' | \mathbf{p} \mathbf{a}_i(\mathbf{x})}$ sono delle *stime esatte*. Essi massimizzano la probabilità a posteriori di un dataset, dato un modello CTBN.

Stima bayesiana

Un approccio alternativo alla stima dei parametri è la stima bayesiana (si veda Nodelman, 2007, sottosezione 5.1.1).

A tal fine è necessario definire una distribuzione a priori sui parametri di una CTBN. Come si è soliti fare in situazioni di questo tipo, per tale distribuzione si sceglie di usare una *distribuzione a priori coniugata*¹¹ poiché ciò risulta conveniente dal punto di vista algebrico (e quindi computazionale). Infatti, una distribuzione a priori coniugata fornisce un'espressione in forma chiusa per la distribuzione a posteriori (alternativamente potrebbe risultare necessario il calcolo di un integrale numerico).

Si consideri innanzitutto un singolo processo di Markov. Si ricorda (si vedano a tal riguardo le definizioni 1.7 e 1.8 a pagina 7 e a pagina 8) che un processo di Markov ha due insiemi di parametri: θ che parametrizzano una *distribuzione multinomiale* e \mathbf{q} che parametrizzano una *distribuzione esponenziale*.

Una distribuzione a priori coniugata per il parametro \mathbf{q} è la *distribuzione Gamma* $P(\mathbf{q}) = \text{Gamma}(\alpha_{\mathbf{x}}, \tau_{\mathbf{x}})$, dove (si veda Nodelman, 2007):

$$P(\mathbf{q}) = \frac{\tau_{\mathbf{x}}^{\alpha_{\mathbf{x}}+1}}{\Gamma(\alpha_{\mathbf{x}}+1)} \mathbf{q}^{\alpha_{\mathbf{x}}} e^{-\mathbf{q} \tau_{\mathbf{x}}}. \quad (1.16)$$

Invece, avendo assunto la *parameter independence* e poiché la funzione di densità della distribuzione di probabilità di θ , che è una multinomiale, è positiva, per essa si sceglie come priori coniugata la *distribuzione di Dirichlet* $P(\theta) = \text{Dir}(\alpha_{\mathbf{x}\mathbf{x}_1}, \dots, \alpha_{\mathbf{x}\mathbf{x}_K})$ (si veda Heckerman,

¹¹ Nella teoria della probabilità bayesiana, se le distribuzioni a posteriori $P(\theta|\mathbf{x})$ sono nella stessa famiglia della distribuzione a priori $P(\theta)$, le due distribuzioni sono definite coniugate, e la distribuzione a priori è chiamata *distribuzione a priori coniugata* per la verosimiglianza (*likelihood*). Una distribuzione a priori coniugata è conveniente dal punto di vista algebrico in quanto fornisce una espressione in forma chiusa per la distribuzione a posteriori e perché può fornire delle intuizioni circa il modo con cui la funzione di verosimiglianza aggiorna la distribuzione.

1996; Heckerman *et al.*, 1995), la cui funzione di densità (Steck, Harald and Jaakkola, 2002) è:

$$P(\theta) = \frac{\Gamma(\alpha_x)}{\Gamma(\alpha_{xx_1}) \cdot \dots \cdot \Gamma(\alpha_{xx_K})} \theta_{xx_1}^{\alpha_{xx_1}-1} \cdot \dots \cdot \theta_{xx_K}^{\alpha_{xx_K}-1}. \quad (1.17)$$

Nota 1.4.3.1. Si noti che l'iper-parametro α_x , detto *dimensione equivalente del campione*, è costituito dalla somma dei *conteggi immaginari* $\alpha_{xx_1} + \dots + \alpha_{xx_K}$, chiamati anche *pseudo-conteggi* (Steck, Harald and Jaakkola, 2002). Esso può essere pensato come un fattore che esprime la “forza” della distribuzione a priori, in quanto, più esso aumenta, più le stime dei parametri sono regolarizzate, cioè meno estreme. Chiaramente, quando α_x tende a 0 le stime dei parametri tendono alle stime maximum-likelihood. In letteratura (si veda Steck, Harald and Jaakkola, 2002) questo processo è anche chiamato “smoothing”.

Nota 1.4.3.2. L'iper-parametro τ_x , invece, rappresenta una *quantità di tempo immaginaria* che incorpora la credenza della distribuzione a priori sul parametro della distribuzione esponenziale.

Quindi, se si assume che i parametri sono *stocasticamente indipendenti*, cioè che $P(\theta, \mathbf{q}) = P(\theta) P(\mathbf{q})$, allora le distribuzioni a posteriori (i.e., condizionate sui dati) dei parametri \mathbf{q} e θ sono:

$$\begin{aligned} P(\mathbf{q} | \mathcal{D}) &= \text{Gamma}(\alpha_x + M[x], \tau_x + T[x]) \\ P(\theta | \mathcal{D}) &= \text{Dir}(\alpha_{xx_1} + M[x, x_1], \dots, \alpha_{xx_K} + M[x, x_K]). \end{aligned} \quad (1.18)$$

Al fine di generalizzare quest'idea e ottenere una distribuzione a priori coniugata per un'intera CTBN è necessario che essa soddisfi due assunzioni (comuni per le distribuzioni a priori nelle Bayesian Network, si veda Heckerman (1996)): la *global parameter independence* e la *local parameter independence*. In base alla *global parameter independence* si può scrivere:

$$P(\mathbf{q}, \theta) = \prod_{X_i \in \mathbf{X}} P(\mathbf{q}_{X_i} | \text{Pa}(X_i), \theta_{X_i} | \text{Pa}(X_i)). \quad (1.19)$$

Invece, dalla *local parameter independence* consegue che è possibile scrivere:

$$P(\mathbf{q}_X | \text{Pa}(X), \theta_X | \text{Pa}(X)) = \left[\prod_{x \in \text{Pa}_i(x)} P(\mathbf{q}_x | \text{pa}_i(x)) \right] \left[\prod_{x \in \text{Pa}_i(x)} P(\theta_x | \text{pa}_i(x)) \right]. \quad (1.20)$$

Se tale distribuzione a priori soddisfa le assunzioni di indipendenza allora anche la distribuzione a posteriori, essendovi coniugata e perciò appartenente alla stessa famiglia parametrica, le soddisferà. In tal caso è possibile mantenere la distribuzione parametrica in forma chiusa e aggiornarla usando le *statistiche sufficienti*:

- $M[x, x' | \text{pa}_i(x)]$ per il parametro $\theta_{x | \text{pa}_i(x)}$
- $M[x | \text{pa}_i(x)]$ e $T[x | \text{pa}_i(x)]$ per il parametro $\mathbf{q}_{x | \text{pa}_i(x)}$.

Data una distribuzione sui parametri è possibile usarla per predire il prossimo evento, mediando la sua probabilità sull'insieme dei possibili valori dei parametri. Questo tipo di previsione è equivalente all'utilizzo dei *valori attesi* dei parametri, i quali hanno la stessa forma dei parametri maximum-likelihood ma considerano i *conteggi immaginari* degli iper-parametri:

$$\begin{aligned}\hat{\mathbf{q}}_{\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})} &= \frac{\alpha_{\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})} + M[\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})]}{\tau_{\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})} + T[\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})]} \\ \hat{\theta}_{\mathbf{x}\mathbf{x}'|\mathbf{p}\mathbf{a}_i(\mathbf{x})} &= \frac{\alpha_{\mathbf{x}\mathbf{x}'|\mathbf{p}\mathbf{a}_i(\mathbf{x})} + M[\mathbf{x}, \mathbf{x}'|\mathbf{p}\mathbf{a}_i(\mathbf{x})]}{\alpha_{\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})} + M[\mathbf{x}|\mathbf{p}\mathbf{a}_i(\mathbf{x})]}.\end{aligned}\tag{1.21}$$

Si osservi che questi parametri sono, teoricamente, validi solo per predire una singola transizione, dopo la quale la distribuzione dei parametri andrebbe aggiornata di conseguenza. Tuttavia, è prassi comune non aggiornare la distribuzione dei parametri utilizzando i succitati *valori attesi* anche per la previsione delle transizioni successive.

2 | CLASSIFICAZIONE

La classificazione è un argomento centrale nei campi di ricerca relativi all'apprendimento automatico (anche detto *machine learning*) e l'analisi dei dati. In generale, essa consiste nel processo di assegnare una *classe* (i.e., un'etichetta) a delle istanze descritte da un insieme di attributi. Si parla di *classificazione supervisionata* quando è necessario indurre un classificatore a partire da un insieme di dati composto da istanze già etichettate e utilizzare tale classificatore per classificare nuove istanze di dati. Nel caso in cui, invece, si vogliano individuare dei raggruppamenti intrinseci (i.e., *cluster*) a un insieme di dati composto da istanze non etichettate, e creare in corrispondenza di tali raggruppamenti le classi (incognite), si parla invece di *classificazione non supervisionata*.

In questo capitolo viene quindi introdotta una classe di modelli, che prende il nome di Continuous time Bayesian Network Classifier (CTBNC), il cui scopo è la *classificazione supervisionata* di traiettorie multi-variate di variabili discrete a *tempo continuo*. Si descrivono due istanze di tale classe: i classificatori Continuous time Naive Bayes (CTNB) e i classificatori Continuous time Tree Augmented Naive Bayes (CTTANB).

Mentre nella sezione 2.2 si affronta il processo di *apprendimento* in caso di *dati completi* dei CTBNC, nella sezione 2.3 si presenta un algoritmo di *inferenza esatta* per la classe dei CTBNC.

2.1 MODELLO

Al fine di risolvere il succitato problema della classificazione sono stati proposti numerosi approcci. Ad esempio Naive Bayes Classifier, un classificatore semplice ma robusto proposto da Duda e Hart (1973); rivelatosi essere uno fra i classificatori più performanti (Langley *et al.*, 1992). Esso apprende dai dati la probabilità condizionale di ogni attributo A_i data la classe C . La classificazione di nuove istanze dei dati è effettuata applicando la *regola di Bayes* al fine di calcolare la probabilità della classe C data l'istanziatura di A_1, \dots, A_N e scegliendo quella con la maggiore probabilità a posteriori. Questo calcolo è reso possibile in modo efficiente grazie ad una forte assunzione: tutti gli attributi A_i sono *condizionalmente indipendenti* (si veda la definizione 1.2) tra di loro data evidenza sulla classe C .

Poiché tale assunzione è chiaramente irrealistica, Friedman *et al.* (1997) ha investigato come migliorare ulteriormente le prestazioni del Naive

Bayes Classifier evitando assunzioni di indipendenza non giustificate dai dati. A tal fine Friedman *et al.* (1997), generalizzando il Naive Bayes Classifier, ha proposto una classe di modelli di *classificazione supervisionata*, chiamata Bayesian Network Classifier (BNC) (di cui fa parte il classificatore Tree Augmented Naive Bayes (TAN), ad esempio) che ereditano dalla teoria delle Bayesian Network (si rimanda alla definizione 1.1 per maggiori dettagli) una rappresentazione fattorizzata delle distribuzioni di probabilità dei nodi attributo e rappresentano esplicitamente le indipendenze condizionali fra essi.

Seguendo le stesse motivazioni, in Stella e Amer (2012) viene formalizzata una classe di modelli di *classificazione supervisionata*, chiamati Continuous time Bayesian Network Classifier (CTBNC), derivata dalle CTBN (si veda definizione 1.15).

Di seguito si definiscono quindi i Continuous time Bayesian Network Classifier e due istanze di classificatori appartenenti a tale classe: il Continuous time Naive Bayes Classifier (CTNBC) e il Continuous time Tree Augmented Naive Bayes Classifier (CTTANBC).

Un Continuous time Bayesian Network Classifier estende una CTBN tramite l'aggiunta di un nodo associato alla variabile classe Y . Si ricorda, dalla definizione 1.15, che una CTBN rappresenta l'evoluzione nel tempo continuo di una variabile di processo X (i. e., insieme composto da N processi di Markov, si veda la definizione 1.11).

Di seguito si dà la definizione di questa nuova classe di modelli di *classificazione supervisionata*.

Definizione 2.1 (Continuous time Bayesian Network Classifier). Un Continuous time Bayesian Network Classifier (CTBNC) è composto da una coppia $\mathcal{C} = (N, P(Y))$ dove:

- N è una CTBN con nodi attributo X_1, X_2, \dots, X_N
- Y è il nodo classe con valori $\text{val}(Y) = \{y_1, \dots, y_K\}$ e probabilità marginale $P(Y)$.

E inoltre il grafo su N (i. e., il grafo \mathcal{G} , si veda la definizione 1.15) rispetta le seguenti condizioni:

- \mathcal{G} è un grafo connesso¹²
- $\text{Pa}(Y) = \{\}$, i. e., la variabile casuale Y è associata al nodo classe
- il nodo Y è indipendente dal tempo ed è specificato solo ed esclusivamente dalla sua probabilità marginale $P(Y)$.

A supporto della definizione 2.1, la figura 2.1 nella pagina seguente fornisce un'istanza di CTBNC composta dai nodi attributi X_1, X_2, X_3, X_4, X_5 e dal nodo classe Y (nodo radice). Si osservi come tale istanza contenga dei cicli, uno riguardante i nodi X_2, X_4, X_5, X_3 e l'altro

¹² Il grafo $\mathcal{G} = (V, E)$ è detto *connesso* se $\forall (u, v) \in V$ esiste un cammino che collega u a v .

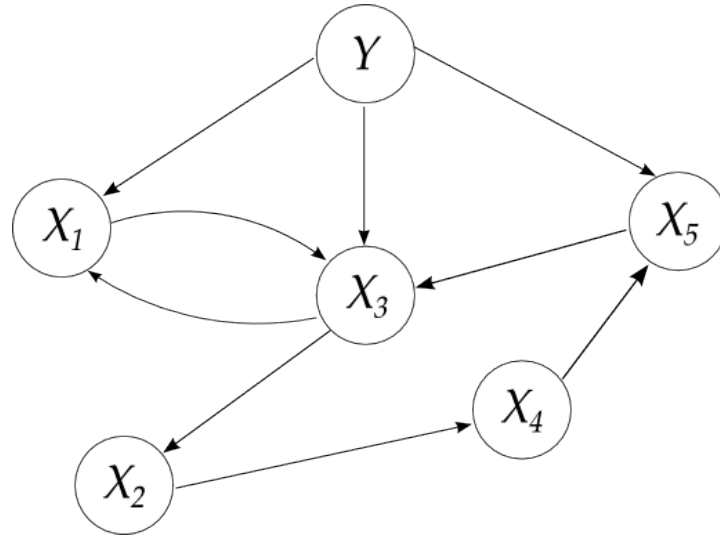


Figura 2.1: Un esempio di Continuous time Bayesian Network Classifier (CTBNC) con cinque nodi attributo, X_1, \dots, X_5 , e un nodo classe, Y .

riguardante i nodi X_1, X_3 . Si fa notare che gli archi della rete \mathcal{N} rappresentano le dipendenze causali nel tempo.

Parallelamente a quanto fatto in Langley *et al.* (1992), si presentano ora due istanze particolari di Continuous time Bayesian Network Classifier.

Definizione 2.2 (Continuous time Naive Bayes Classifier). Un Continuous time Naive Bayes Classifier (CTNBC) è un Continuous time Bayesian Network Classifier $\mathcal{C} = (\mathcal{N}, \mathbf{P}(Y))$ caratterizzato dal fatto che ogni nodo attributo ha un solo genitore, il nodo classe Y . Risulta quindi che:

$$\text{Pa}(X_i) = \{Y\} \quad \forall X_i \in \mathcal{G}.$$

Come mostrato dalla figura 2.2, un CTNBC possiede un nodo radice, associato alla variabile casuale Y , che è l'unico genitore di tutti i restanti nodi X_i (con $i = 1, 2, \dots, N$) che lo compongono. Si osservi come la rete di un CTNBC rappresenti l'assunzione di *indipendenza condizionale* di ogni nodo attributo dagli altri, data evidenza sulla variabile classe Y .

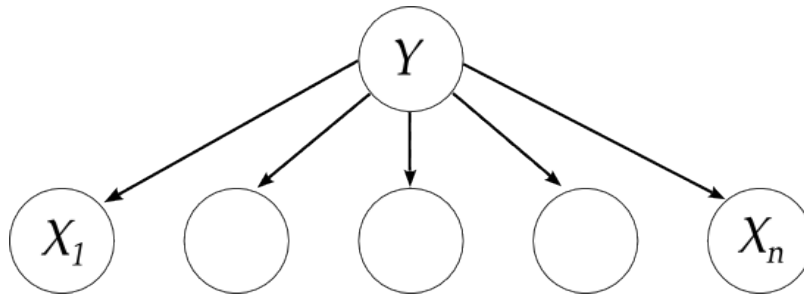


Figura 2.2: Un Continuous time Naive Bayes Classifier (CTNBC).

Definizione 2.3 (Continuous time Tree Augmented Naive Bayes Classifier). Un Continuous time Tree Augmented Naive Bayes Classifier (CTTANBC) è un Continuous time Bayesian Network Classifier $\mathcal{C} = (\mathcal{N}, \mathbf{P}(Y))$ che rispetta i seguenti vincoli:

- $Y \in \text{Pa}(X_i)$ con $i = 1, 2, \dots, N$
- i nodi attributo X_i , $i = 1, 2, \dots, N$, formano un albero:

$$\exists j : |\text{Pa}(X_j)| = 1 \quad \text{mentre per } i \neq j : |\text{Pa}(X_i)| = 2.$$

Come mostrato dalla figura 2.3, un classificatore CTTANB è un estensione del classificatore CTNB: tutti i nodi attributo della rete \mathcal{N} sono vincolati ad avere come genitore, oltre al nodo radice, al massimo un altro nodo attributo. Ciò comporta che tutti i nodi attributo facciano parte del *Markov blanket* del nodo radice associato con la variabile classe Y .

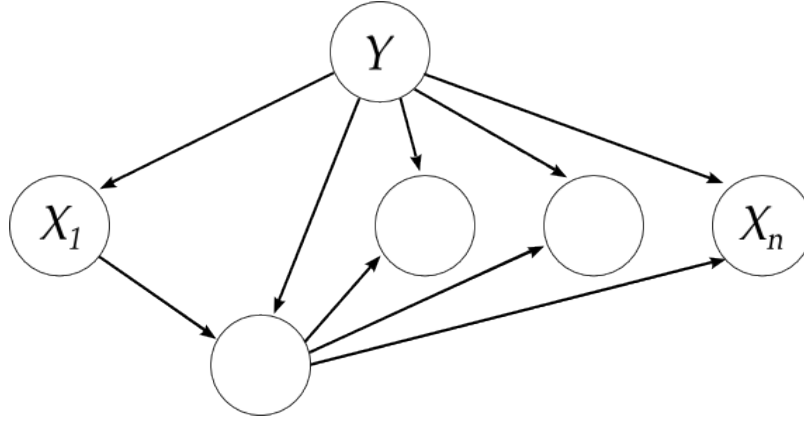


Figura 2.3: Un Continuous time Tree Augmented Naive Bayes Classifier (CTTANBC): qualora la variabile classe Y venga rimossa, le variabili rimanenti formano un albero.

2.2 APPENDIMENTO

In questa sezione si affronta il problema dell'apprendimento (da *dati completi*) dei CTBNC.

Per definizione (si veda la definizione 2.1 a pagina 22) i CTBNC sono basati sul modello delle CTBN, rappresentano perciò un insieme di modelli di probabilità locali (relativi alle variabili casuali) esprimibili in termini di statistiche sufficienti (per maggiori dettagli relativi a questo aspetto si rimanda alla sezione 1.4). Ne deriva che il problema dell'apprendimento di un classificatore CTBN si riduce alla computazione delle statistiche sufficienti dei suoi nodi attributo, da cui è successivamente possibile stimare i parametri (argomento trattato in dettaglio nella sottosezione 1.4.3) delle distribuzioni di probabilità codificate dalle matrici di intensità condizionali (CIM).

Di conseguenza, per l'apprendimento di un CTNBC è richiesto uno sforzo computazionale poco costoso.

Per l'apprendimento di un CTTANBC, poiché questo modello prevede archi anche fra i nodi attributo, è invece richiesto uno sforzo computazionale leggermente maggiore (Stella e Amer, 2012).

Si presenta di seguito l'algoritmo 2.1 (Stella e Amer, 2012) relativo all'apprendimento di un classificatore CTNB (definizione 2.2).

Esso richiede in input un *dataset completo* $\mathcal{D} = \{\delta_1, \dots, \delta_h\}$, il corrispondente insieme delle classi $\{y_1, y_2, \dots, y_h\}$, con $y_i \in \text{val}(Y)$, e il grafo \mathcal{G} di una CTBN \mathcal{N} (rispettivamente chiamati *data*, *classes* e *graph* nella firma della funzione `ctnbclearn`).

Per completezza si osservi (in base alla definizione 1.16) che ogni δ_i (con $i = 1, \dots, h$) è un flusso di evidenze $(x^1, x^2, \dots, x^{j_i})$ (a tal riguardo si veda la definizione 1.14).

Il risultato dell'applicazione dell'algoritmo 2.1 è un Continuous time Naive Bayes Classifier $\mathcal{C} = (\mathcal{N}, \mathbf{P}(Y))$.

```

1 function ctnbclearn(data, classes, graph) {
2     var h = len(data)
3     var klass = unique(classes)
4     var nk = len(klass)
5     var priors[nk]
6     for (i in index(data)) {
7         var k = index(classes[i])
8         priors[k] = priors[k] + (1 / h)
9     }
10    var m(), t()
11    for (i in index(data)) {
12        var y = classes[i]
13        var j = 1
14        while (tj ≤ Ti) {
15            for (n in index(graph.nodes)) {
16                m(xnj, xnj+1, y) = m(xnj, xnj+1, y) + 1
17                t(xnj, y) = t(xnj, y) + (tj - tj-1)
18            }
19            j = j + 1
20        }
21    }
22    var q(), thet()
23    foreach (y in klass) {
24        for (n in index(graph.nodes)) {
25            for (xn in val(Xn)) {
26                var mm(xn, y) =  $\sum_{x'_n \neq x_n} m(x_n, x'_n, y)$ 
27                q(xn, y) = mm(xn, y) / t(xn, y)
28                thet(xn, y) = m(xn, x'_n, y) / mm(xn, y)
29            }

```

```

30     }
31 }
32 var ctbn = new ctbn(graph, q, thet)
33 return (priors, ctbn)
34 }

```

Algoritmo 2.1: Apprendimento di un classificatore CTNB (CTNBC).

L'algoritmo di apprendimento appena presentato consiste nella stima delle matrici di intensità condizionali (CIM) di ogni variabile casuale di \mathcal{N} per ogni classe $y_i \in Y$. Più in dettaglio, esso è composto da tre fasi consecutive:

1. da linea 2 a linea 9 viene calcolata la *probabilità a priori* della variabile classe Y in base alla frequenza di ogni sua istanziazione $y_i \in Y$ in \mathcal{D}
2. da linea 10 a linea 21 vengono calcolate le statistiche sufficienti di ogni nodo attributo X_i , con $i = 1, 2, \dots, N$, sull'insieme di dati di apprendimento (anche detto *training set*) \mathcal{D}
3. da linea 22 a linea 31 vengono infine stimati, a partire dalle statistiche sufficienti, i *parametri maximum-likelihood* (MLE).

Si osservi che, poiché il processo di apprendimento è eseguito su un classificatore CTNB, l'algoritmo condiziona sia il calcolo delle statistiche sufficienti (i.e., variabili t e m) che la stima dei parametri (i.e., variabili q e thet) di ogni nodo attributo X_i solo ed esclusivamente al valore della variabile classe Y (i.e., variabile y). Questa semplificazione è dovuta al vincolo che caratterizza i classificatori CTNB: ogni nodo attributo X_i ha un solo genitore, il nodo associato alla variabile classe Y (si veda la definizione 2.2).

Come anticipato, al costo di un leggero incremento di complessità computazionale, è possibile estendere l'algoritmo 2.1 al fine di creare un algoritmo di apprendimento generale che apprenda un qualsiasi classificatore CTBN. Affinché tale obiettivo sia raggiunto è necessario rimuovere il succitato vincolo sull'insieme dei genitori di ogni nodo attributo. Mentre il calcolo della probabilità a priori della variabile classe Y non varia, il calcolo delle statistiche sufficienti e la stima dei parametri, invece, necessitano di tale generalizzazione.

Nello specifico:

1. il calcolo delle statistiche sufficienti di ogni nodo attributo X_i va condizionato all'istanziatura attuale (i.e., al tempo j) del suo insieme di nodi genitori $\text{Pa}(X_i)$; di conseguenza, a linea 15 dell'algoritmo 2.2, si prende in considerazione tale valore (i.e., variabile p)
2. la stima dei parametri maximum-likelihood (MLE) di ogni nodo attributo X_i va eseguita in base a ogni istanziazione del suo

insieme di nodi genitori $\text{Pa}(X_i)$; perciò a linea 25 si itera in base a ogni valore (i.e., variabile p) assunto da $\text{Pa}(X_i)$ (i.e., $\text{val}(\text{Pa}(X_i))$) mentre l'iterazione per classe non è più coerente e di conseguenza rimossa.

Si riporta di seguito l'algoritmo 2.2, il quale include le succitate modifiche finalizzate alla creazione di un algoritmo di apprendimento generale per i CTBNC.

```

1  function learn(data, classes, graph) {
2      var h = len(data)
3      var klass = unique(classes)
4      var nk = len(klass)
5      var priors[nk]
6      for (i in index(data)) {
7          var k = index(klass[i])
8          priors[k] = priors[k] + (1 / h)
9      }
10     var m(), t()
11     foreach (i in index(data)) {
12         var j = 1
13         while ( $t_j \leq T_i$ ) {
14             foreach (n in index(graph.nodes)) {
15                 var p =  $\text{val}^j(\text{Pa}(X_n))$ 
16                  $m(x_n^j, x_n^{j+1}, p) = m(x_n^j, x_n^{j+1}, p) + 1$ 
17                  $t(x_n^j, p) = t(x_n^j, p) + (t_j - t_{j-1})$ 
18             }
19             j = j + 1
20         }
21     }
22     var q(), thet()
23     foreach (n in index(graph.nodes)) {
24         for ( $x_n$  in  $\text{val}(X_n)$ ) {
25             foreach (p in  $\text{val}(\text{Pa}(X_n))$ ) {
26                  $\text{mm}(x_n, p) = \sum_{x'_n \neq x_n} m(x_n, x'_n, p)$ 
27                  $q(x_n, p) = \text{mm}(x_n, p) / t(x_n, p)$ 
28                  $\text{thet}(x_n, p) = m(x_n, x'_n, p) / \text{mm}(x_n, p)$ 
29             }
30         }
31     }
32     var ctbn = new ctbn(graph, q, thet)
33     return (priors, ctbn)
34 }

```

Algoritmo 2.2: Apprendimento di un classificatore CTBN (CTBNC).

2.3 INFERENZA

In questa sezione si affronta il problema della *classificazione* di un *flusso di evidenze completamente osservato*, indicato con $(\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J)$ (si veda la definizione 1.14), rispetto a un classificatore CTBN (CTBNC). Si osservi che, in tale situazione (i. e., *dati completi*), l'unica variabile casuale non osservata è la variabile classe, perciò è possibile sfruttare le relazioni di indipendenza fra variabili casuali così come si fa per le Bayesian Network. L'argomento di questa sezione è quindi il processo di *classificazione supervisionata*, di cui si presentano in primis le basi teoriche e successivamente l'implementazione algoritmica che ne consegue.

Il processo di classificazione di un flusso di evidenze è effettuato in base alla regola *maximum a posteriori*¹³ (MAP) (si veda Stella e Amer, 2012): un flusso di evidenze completamente osservato viene classificato assegnandogli la classe la cui probabilità a posteriori (rispetto al flusso di evidenze stesso) è massima. A tale scopo è necessario calcolare la probabilità a posteriori della variabile classe Y del CTBNC, rispetto al flusso di evidenze in input, per tutti i suoi possibili stati (i. e., classi, o etichette).

Il classificatore CTBN classifica quindi il flusso di evidenze massimizzando la seguente probabilità a posteriori, ricavata applicando la *regola di Bayes*:

$$P(Y | (\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J)) = \frac{P((\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J) | Y) P(Y)}{P((\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J))}. \quad (2.1)$$

Si specifica di seguito la semantica dei componenti dell'equazione 2.1:

- la probabilità marginale associata alla variabile classe Y

$$P(Y)$$

- la probabilità del flusso di evidenze

$$P((\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J))$$

- la likelihood del flusso di evidenze dato il valore della variabile classe, a cui ci si riferisce nel prosieguo usando l'espressione "*likelihood temporale*"

$$P((\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J) | Y).$$

¹³ La stima della probabilità maximum a posteriori (MAP) è una moda della distribuzione a posteriori che può essere usata per ottenere una stima puntuale di una quantità inosservata sulla base di dati empirici. Può essere vista come una regolarizzazione della stima maximum-likelihood (MLE) poiché è strettamente correlata ad essa; vi differisce perché impiega un obiettivo di massimizzazione incrementato che incorpora una distribuzione a priori sopra la quantità che si vuole stimare.

La probabilità del flusso di evidenze (i. e., denominatore dell'equazione 2.1), similmente a quanto accade per i BNC (Friedman *et al.*, 1997), può essere omessa poiché sussiste la relazione di proporzionalità tra il numeratore e la probabilità che si intende calcolare:

$$P(Y | (\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J)) \propto P((\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J) | Y) P(Y). \quad (2.2)$$

Il primo termine dell'equazione 2.2, cioè la *likelihood temporale*, è invece fondamentale per la classificazione tramite regola MAP ed è possibile riformularlo nel seguente modo:

$$P((\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J) | Y) = \prod_{j=1}^J P(\mathbf{x}^j | Y) P(\mathbf{x}^{j+1} | \mathbf{x}^j, Y), \quad (2.3)$$

dove:

- $P(\mathbf{x}^j | Y)$ rappresenta la probabilità che il *vettore aleatorio*¹⁴ \mathbf{X} resti nello stato \mathbf{x}^j durante l'intervallo temporale $[t_{j-1}, t_j)$ data evidenza sulla variabile classe Y
- $P(\mathbf{x}^{j+1} | \mathbf{x}^j, Y)$ rappresenta la probabilità che in \mathbf{X} si verifichi una transizione da \mathbf{x}^j a \mathbf{x}^{j+1} all'istante di tempo t_j data evidenza sulla variabile classe Y .

Inoltre, al fine di assicurare la consistenza dell'equazione 2.3 si assume che $P(\mathbf{x}^{J+1} | \mathbf{x}^J, Y) = 1$. Si osservi che benché non sia possibile che alcuna transizione di stato avvenga nell'ultimo (i. e., J -esimo) segmento temporale, semplicemente poiché non è previsto un segmento temporale successivo, si assume che la probabilità che se ne verifichino è sempre pari a 1 affinché il valore nullo di tale quantità non vanifichi il contributo apportato dal primo termine dell'equazione 2.3 nell'ultimo segmento temporale.

Il passo successivo consiste nel calcolo dei due termini da cui è composta l'equazione 2.3. A tal fine si utilizzano le distribuzioni di probabilità locali associate ad ogni nodo del CTBN su cui è costruito il classificatore. Come già descritto nella sottosezione 1.1.2, tali modelli di probabilità sono espressi tramite le matrici di intensità condizionali e quindi tramite i parametri \mathbf{q} e $\boldsymbol{\theta}$.

In tale contesto è quindi possibile calcolare il termine $P(\mathbf{x}^j | Y)$ come segue:

$$P(\mathbf{x}^j | Y) = \prod_{n=1}^N \exp\left(-\mathbf{q}_{\mathbf{x}_n^j}^{\mathbf{p} \mathbf{a}_j(\mathbf{x}_n)} (t_j - t_{j-1})\right), \quad (2.4)$$

dove $\mathbf{q}_{\mathbf{x}_n^j}^{\mathbf{p} \mathbf{a}_j(\mathbf{x}_n)}$ è il valore del parametro della *distribuzione esponenziale* quando la variabile casuale X_n è nello stato x_n durante il j -esimo

¹⁴ Un *vettore aleatorio* $\mathbf{X} = (X_1, X_2, \dots, X_n)$ è una *n-upla* (composta da n variabili casuali) i cui elementi sono dati da numeri aleatori.

intervallo temporale $[t_{j-1}, t_j]$ e contemporaneamente l'istanziamento dei genitori di X_n è $pa_j(x_n)$.

Uguualmente, il termine $P(x^{j+1} | x^j, Y)$ è così calcolabile:

$$P(x^{j+1} | x^j, Y) = \prod_{n=1}^N P(x_n^{j+1} | x_n^j, Y), \quad (2.5)$$

dove

$$P(x_n^{j+1} | x_n^j, Y) = \begin{cases} q_{x_n^j x_n^{j+1}}^{pa_j(x)} & \text{se } x_n^j \neq x_n^{j+1} \\ 1, & \text{altrimenti} \end{cases}. \quad (2.6)$$

Il termine $P(x_n^{j+1} | x_n^j, Y)$ rappresenta la probabilità che nel vettore aleatorio \mathbf{X} si verifichi una transizione dallo stato x^j allo x^{j+1} , dato il valore della variabile classe Y .

Poiché il modello CTBN implica che, ad ogni istante t_j , solo un componente X_n del vettore aleatorio \mathbf{X} può essere soggetto a transizione, allora $P(x_n^{j+1} | x_n^j, Y)$ rappresenta la probabilità che la variabile casuale X_n effettui una transizione da x_n^j a x_n^{j+1} mentre tutti gli altri componenti del vettore aleatorio \mathbf{X} (i.e., X_i con $i \neq n$) non cambiano il proprio stato.

Perciò, come specificato dall'equazione 2.6, nel caso in cui avvenga un cambio di stato in X_n , il termine $P(x_n^{j+1} | x_n^j, Y)$ equivale alla quantità $q_{x_n^j x_n^{j+1}}^{pa_j(x)}$, ricavata dalla relazione fra i parametri (si veda il teorema 1.2). Tale quantità rappresenta il parametro associato alla transizione da x_n^j , stato in cui la variabile casuale X_n si trovava durante il j -esimo intervallo temporale $[t_{j-1}, t_j]$, a x_n^{j+1} , stato in cui X_n si troverà durante il successivo intervallo temporale $[t_j, t_{j+1}]$; data l'istanziamento $pa_j(x_n)$ dei genitori di X_n durante il j -esimo intervallo temporale.

Combinando l'equazione 2.4 e l'equazione 2.5 si ottiene:

$$P(x^j | Y) P(x^{j+1} | x^j, Y) = \prod_{n=1}^N \exp\left(-q_{x_n^j}^{pa_j(x_n)} (t_j - t_{j-1})\right) P(x_n^{j+1} | x_n^j, Y). \quad (2.7)$$

Utilizzando l'equazione 2.7 appena ricavata è possibile riformulare la *likelihood temporale* (equazione 2.3) nel seguente modo:

$$P((x^1, x^2, \dots, x^J) | Y) = \prod_{j=1}^J \prod_{n=1}^N \exp\left(-q_{x_n^j}^{pa_j(x_n)} (t_j - t_{j-1})\right) P(x_n^{j+1} | x_n^j, Y). \quad (2.8)$$

Sostituendo infine l'equazione equazione 2.8 nell'equazione 2.2 si formula definitivamente la probabilità a posteriori della variabile classe Y dato un flusso di evidenze:

$$P(Y | (x^1, x^2, \dots, x^J)) \propto P(Y) \cdot \prod_{j=1}^J \prod_{n=1}^N \left[\exp\left(-q_{x_n^j}^{pa_j(x_n)} (t_j - t_{j-1})\right) \cdot P(x_n^{j+1} | x_n^j, Y) \right]. \quad (2.9)$$

Di conseguenza, dato un classificatore CTBN $\mathcal{C} = \{\mathcal{N}, P(Y)\}$ e un flusso di evidenze completamente osservato $(\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^J)$, la regola MAP seleziona la classe $y^* \in \text{val}(Y)$ massimizzando l'equazione 2.9:

$$y^* = \arg \max_{y \in \text{val}(Y)} P(Y) \prod_{j=1}^J \prod_{n=1}^N \exp\left(-\mathbf{q}_{x_n^j}^{p_{a_j}(x_n)} (t_j - t_{j-1})\right) P(x_n^{j+1} | x_n^j, Y) \quad (2.10)$$

Si presenta di seguito l'algoritmo per l'*inferenza esatta* (Stella e Amer, 2012) di un flusso di evidenze completamente osservato rispetto a un classificatore CTBN.

```

1 function infer(ctbnc, timeseg, stream) {
2   var priors = ctbnc.priors
3   var logp[len(priors)]
4   for (k in index(priors)) logp[k] = log(priors[k])
5   for (k in index(priors)) {
6     for (j in index(timeseg)) {
7       for (n in index(ctbnc.graph.nodes)) {
8         logp[k] = logp[k] -  $\mathbf{q}_{x_n^j}^{p_{a_j}(x_n)}$  * timeseg[j]
9         if ( $x_{n_j} \neq x_{n_{j+1}}$ ) {
10          logp[k] = logp[k] + log( $\mathbf{q}_{x_n^j x_n^{j+1}}^{p_{a_j}(x)}$ )
11        }
12      }
13    }
14  }
15  return which(max(logp))
16 }
```

Algoritmo 2.3: Inferenza su un classificatore CTBN (CTBNC).

Si osservi che tale algoritmo rappresenta le probabilità come *log-probabilità*¹⁵, a causa della succitata convenienza algebrica che ne deriva. Ciò significa che l'algoritmo 2.3 implementa la probabilità a posteriori della classe dato un flusso di evidenze (equazione 2.9) come segue:

$$\ell_{P(Y|\dots)} = \log(P(Y)) + \sum_{j=1}^J \sum_{n=1}^N -\mathbf{q}_{x_n^j}^{p_{a_j}(x_n)} (t_j - t_{j-1}) + \log(P(x_n^{j+1} | x_n^j, Y)). \quad (2.11)$$

A tal proposito si noti che, nel caso in cui non avvenga alcun cambio di stato durante un determinato istante di tempo t_j , la quantità $P(x_n^{j+1} | x_n^j, Y)$ dell'equazione 2.11 sarà pari a 1 (come specificato

¹⁵ La *log-probabilità* è un modalità di rappresentazione della probabilità che porta con sé alcuni vantaggi algebrici e computazionali. Ad esempio, l'utilizzo della *log-probabilità* generalmente comporta una maggior velocità dovuta alla trasformazione delle moltiplicazioni, più computazionalmente costose, in addizioni. In informatica è molto comune l'utilizzo della sua variante negativa, la quale codifica un valore di probabilità $x \in [0, 1]$ come $x' = -\log(x) \in \mathbb{R}$.

dall'equazione 2.6), il cui logaritmo è pari a 0. Ciò si riflette nella struttura di controllo condizionale alla linea 9.

L'algoritmo 2.3 di inferenza esatta appena presentato è composto principalmente da due fasi corrispondenti ai due termini principali dell'equazione 2.11: da linea 2 a linea 4 converte la distribuzione della variabile classe Y del classificatore CTBN in forma logaritmica; da linea 5 a linea 14 aggiorna, incrementandola o decrementandola, il valore di *log-probabilità* relativo a ogni classe (si veda il ciclo **for** alla linea 5) del classificatore CTBN iterando il flusso di evidenze per ogni segmento temporale cui sono associate le sue istanze (i.e., variabile *timeseg*, che si assume venga fornita in input; si veda il ciclo **for** alla linea 6), infine iterando sui nodi del grafo N del CTBNC (ciclo **for** a linea 7).

Il passo finale dell'algoritmo, corrispondente alla linea 15, consiste nella restituzione della classe la cui *log-probabilità* è maggiore di quella delle restanti classi. Tale passo completa perciò l'implementazione dell'equazione 2.10.

L'algoritmo di inferenza esatta presentato in questa sezione è polinomiale.

Nota 2.3.1. Si osservi che è possibile implementare l'algoritmo di inferenza anche utilizzando la *parametrizzazione mista* (si veda la definizione 1.8). In tal caso si evita la computazione delle matrici di intensità condizionali (CIM) ma, in contrasto, ogni qual volta una variabile casuale del flusso di evidenze di input effettua una transizione è necessario calcolare il termine $q_{x_n^j x_n^{j+1}}^{p a_j(x)}$ (a tal proposito si veda l'equazione 1.14).

3 | APPRENDIMENTO STRUTTURALE

Uno dei casi principali che costituisce il problema dell'*apprendimento* di modelli grafico probabilistici è l'apprendimento della struttura incognita sottostante un modello, cioè la selezione di un modello probabilistico che rappresenti un dato *training set*.

Il problema dell'*apprendimento strutturale* da *dati completi* di una Continuous time Bayesian Network (CTBN) è l'argomento trattato in questo capitolo.

Questo problema può essere informalmente descritto nel seguente modo: dato un *training set* composto da istanze di un insieme di variabili casuali si trovi un grafo che rappresenti le relazioni fra le variabili casuali evidenziate nei dati.

L'obiettivo è quindi indurre una struttura (i.e., grafo) che descriva nel miglior modo possibile la distribuzione di probabilità sui dati (i.e., *training set*). Si osservi, inoltre, che questo problema di ottimizzazione è NP-completo per le Bayesian Network (Chickering *et al.*, 2004; Chickering, 1994). Per questa ragione viene spesso trattato con algoritmi approssimati.

Per quanto riguarda invece il caso delle CTBN, Nodelman *et al.* (2002) hanno dimostrato che, grazie alla mancanza del vincolo di aciclicità, come già accennato nella sezione 1.3, il problema dell'apprendimento strutturale di una CTBN è significativamente più facile rispetto all'apprendimento strutturale di una Bayesian Network, o di modelli da esse derivanti (e.g., le Dynamic Bayesian Networks (DBN)). Inoltre, nel caso si vincoli la procedura di ricerca a strutture con un numero massimo di genitori per nodo, questo problema può essere risolto in tempo polinomiale rispetto al numero di nodi nella rete¹⁶

L'approccio che si presenta in questo capitolo è quindi un approccio basato sul punteggio: si definisce una funzione che computa uno *score bayesiano* finalizzato alla valutazione di ogni struttura rispetto ai dati di addestramento (i.e., *training set*) e si usa una tecnica di ricerca euristica (e.g., la ricerca *hill climbing*) per cercare nello spazio delle strutture candidate quella che esibisce il maggior punteggio.

Si osservi che l'apprendimento dei parametri (si veda la sottosezione 1.4.3 a pagina 16) è propedeutico per tale obiettivo poiché essi costituiscono la base dello *score bayesiano*.

¹⁶ Si noti comunque che il problema rimane esponenziale rispetto al numero massimo di genitori nella rete. Per tale motivo Stella e Amer (2012) definiscono un classificatore bayesiano a tempo continuo.

3.1 FUNZIONE DI SCORING

Qualsiasi processo di apprendimento strutturale basato su punteggio è costituito da due componenti: una *funzione di scoring* e una procedura di ottimizzazione.

L'obiettivo di questa sezione è quindi presentare una funzione di scoring per l'apprendimento strutturale delle Continuous time Bayesian Network (CTBN). Lo scopo di tale funzione è calcolare il punteggio (i.e., lo score bayesiano) di una struttura relativamente al *training set* \mathcal{D} fornito.

Si definisce lo *score bayesiano* sul grafo \mathcal{G} di una CTBN nel seguente modo:

$$\text{score}_B(\mathcal{G} : \mathcal{D}) = \ln P(\mathcal{D} | \mathcal{G}) + \ln P(\mathcal{G}) \quad (3.1)$$

Come mostra l'equazione 3.1 la funzione di scoring utilizza la probabilità a posteriori dell'insieme dei dati di apprendimento (i.e., il training set \mathcal{D}) data la struttura candidata (i.e., \mathcal{G}), oltre alla probabilità a priori della struttura stessa.

È possibile aumentare in modo significativo l'efficienza dell'algoritmo di ricerca che si affronta nella prossima sezione qualora si facciano determinate assunzioni. Nello specifico, se si assume che la probabilità a priori della struttura, $P(\mathcal{G})$, soddisfi la *structure modularity*, ne consegue:

$$P(\mathcal{G}) = \prod_{X_i} P(\text{Pa}(X_i) = \text{Pa}_{\mathcal{G}}(X_i)). \quad (3.2)$$

Se si assume, inoltre, che la probabilità a priori dei parametri soddisfi la *parameter modularity*, allora per ogni due strutture \mathcal{G} e \mathcal{G}' tali che $\text{Pa}_{\mathcal{G}}(X) = \text{Pa}_{\mathcal{G}'}(X)$ risulta:

$$P(\mathbf{q}_X, \boldsymbol{\theta}_X | \mathcal{G}) = P(\mathbf{q}_X, \boldsymbol{\theta}_X | \mathcal{G}'). \quad (3.3)$$

Combinando l'assunzione di *parameter independence* con l'equazione 3.3 derivante dalla *parameter modularity*, si ottiene:

$$P(\mathbf{q}_{\mathcal{G}}, \boldsymbol{\theta}_{\mathcal{G}} | \mathcal{G}) = \prod_{X_i} \left[P(\mathbf{q}_{X_i} | \text{Pa}(X_i) | \text{Pa}(X_i) = \text{Pa}_{\mathcal{G}}(X_i)) \cdot P(\boldsymbol{\theta}_{X_i} | \text{Pa}(X_i) | \text{Pa}(X_i) = \text{Pa}_{\mathcal{G}}(X_i)) \right]. \quad (3.4)$$

Si osservi che, poiché la *penalità del grafo*, corrispondente al termine $P(\text{Pa}(X_i) = \text{Pa}_{\mathcal{G}}(X_i))$ dell'equazione 3.2, è legata alla dimensione del grafo ma indipendente dalla quantità dei dati, è possibile ignorare il termine $P(\mathcal{G})$ della funzione di scoring (equazione 3.1).

Di conseguenza il termine significativo dell'equazione 3.1 è la *likelihood marginale*, $P(\mathcal{D} | \mathcal{G})$. Tale termine, infatti, codifica l'incertezza sui parametri integrando su tutti i possibili valori che essi possono assumere:

$$P(\mathcal{D} | \mathcal{G}) = \int_{\mathbf{q}_{\mathcal{G}}, \boldsymbol{\theta}_{\mathcal{G}}} P(\mathcal{D} | \mathbf{q}_{\mathcal{G}}, \boldsymbol{\theta}_{\mathcal{G}}) P(\mathbf{q}_{\mathcal{G}}, \boldsymbol{\theta}_{\mathcal{G}} | \mathcal{G}) d\mathbf{q}_{\mathcal{G}} d\boldsymbol{\theta}_{\mathcal{G}}. \quad (3.5)$$

Come per l'equazione 1.8, la likelihood marginale può essere decomposta come un prodotto di likelihood:

$$\begin{aligned} P(\mathcal{D} | \mathbf{q}_g, \boldsymbol{\theta}_g) &= \prod_{X_i} L_{X_i}(\mathbf{q}_{X_i} | \text{pa}(X_i) : \mathcal{D}) L_{X_i}(\boldsymbol{\theta}_{X_i} | \text{pa}(X_i) : \mathcal{D}) \\ &= \underbrace{\left[\prod_{X_i} L_{X_i}(\mathbf{q}_{X_i} | \text{pa}(X_i) : \mathcal{D}) \right]}_{L(\mathbf{q} : \mathcal{D})} \underbrace{\left[\prod_{X_i} L_{X_i}(\boldsymbol{\theta}_{X_i} | \text{pa}(X_i) : \mathcal{D}) \right]}_{L(\boldsymbol{\theta} : \mathcal{D})}. \end{aligned} \quad (3.6)$$

Combinando tale decomposizione con l'*parameter independence* si può riformulare la likelihood marginale (equazione 3.5) nel seguente modo:

$$\begin{aligned} P(\mathcal{D} | \mathcal{G}) &= \int_{\mathbf{q}_g, \boldsymbol{\theta}_g} L(\mathbf{q}_g : \mathcal{D}) L(\boldsymbol{\theta}_g : \mathcal{D}) P(\mathbf{q}_g) P(\boldsymbol{\theta}_g) d\mathbf{q}_g d\boldsymbol{\theta}_g \\ &= \underbrace{\left[\int_{\mathbf{q}_g} L(\mathbf{q}_g : \mathcal{D}) P(\mathbf{q}_g) d\mathbf{q}_g \right]}_{(a)} \cdot \underbrace{\left[\int_{\boldsymbol{\theta}_g} L(\boldsymbol{\theta}_g : \mathcal{D}) P(\boldsymbol{\theta}_g) d\boldsymbol{\theta}_g \right]}_{(b)}. \end{aligned} \quad (3.7)$$

Ottenuta tale equazione, si affronta di seguito l'analisi e la decomposizione dei due termini che la compongono.

Utilizzando l'assunzione di *local parameter independence*, il termine (a) dell'equazione 3.7 è decomponibile nel seguente modo. Si noti che per brevità si pone $u = \text{pa}_i(x)$.

$$\prod_{X_i} \prod_u \prod_x \int_0^\infty P(\mathbf{q}_{x|u}) \cdot L_{X_i}(\mathbf{q}_{x|u} : \mathcal{D}) d\mathbf{q}_{x|u}. \quad (a)$$

Sostituendo a tale termine la distribuzione a priori coniugata su \mathbf{q} (si veda l'equazione 1.16) e la likelihood delle quantità di tempo trascorse in ogni stato (si veda l'equazione 1.10) si ottiene:

$$\begin{aligned} \prod_{X_i} \prod_u \prod_x \int_0^\infty &\left(\frac{(\tau_{x|u})^{\alpha_{x|u}+1}}{\Gamma(\alpha_{x|u}+1)} (\mathbf{q}_{x|u})^{\alpha_{x|u}} e^{-\mathbf{q}_{x|u} \tau_{x|u}} \cdot \right. \\ &\left. \cdot (\mathbf{q}_{x|u})^{M[x|u]} e^{-\mathbf{q}_{x|u} T[x|u]} \right) d\mathbf{q}_{x|u}. \end{aligned} \quad (a)$$

Si procede semplificando:

$$\prod_{X_i} \prod_u \prod_x \int_0^\infty \frac{(\tau_{x|u})^{\alpha_{x|u}+1} \cdot (\mathbf{q}_{x|u})^{\alpha_{x|u}+M[x|u]}}{\Gamma(\alpha_{x|u}+1) \cdot e^{\mathbf{q}_{x|u}(\tau_{x|u}+T[x|u])}} d\mathbf{q}_{x|u}. \quad (a)$$

E infine, risolvendo l'integrale, si ottiene:

$$\prod_{X_i} \prod_u \prod_x \underbrace{\frac{\Gamma(\alpha_{x|u}+M[x|u]+1)(\tau_{x|u})^{\alpha_{x|u}+1}}{\Gamma(\alpha_{x|u}+1)(\tau_{x|u}+T[x|u])^{\alpha_{x|u}+M[x|u]+1}}}_{\text{MargL}^q(X_i, \text{pa}_g(X_i) : \mathcal{D})}. \quad (a)$$

Relativamente all'analisi del termine (b) dell'equazione 3.7 si osservi che, poiché le distribuzioni sui parametri θ sono di *Dirichlet*, tale operazione è analoga a quella comune per le Bayesian Network.

Ne consegue che il termine (b) si semplifica:

$$\underbrace{\prod_{X_i} \prod_u \prod_x \frac{\Gamma(\alpha_{x|u})}{\Gamma(\alpha_{x|u} + M[x|u])} \cdot \prod_{x \neq x'} \frac{\Gamma(\alpha_{xx'|u} + M[x, x'|u])}{\Gamma(\alpha_{xx'|u})}}_{\text{MargL}^\theta(X_i, \text{Pa}_g(X_i) : \mathcal{D})} \cdot (b)$$

Quindi si può riformulare la likelihood marginale:

$$P(\mathcal{D} | \mathcal{G}) = \prod_{X_i} \text{MargL}^q(X_i, \text{Pa}_g(X_i) : \mathcal{D}) \cdot \text{MargL}^\theta(X_i, \text{Pa}_g(X_i) : \mathcal{D}). \quad (3.8)$$

Al fine di derivare la probabilità a priori della struttura (equazione 3.2) si è già assunto in precedenza che l'ipotesi di *structure modularity* sussista. Perciò, sfruttando tale assunzione e combinando l'equazione 3.2 della probabilità a priori della struttura con la likelihood marginale (equazione 3.8), si ottiene:

$$\begin{aligned} \text{score}_B(\mathcal{G} : \mathcal{D}) &= \sum_{X_i} \left[\ln P(\text{Pa}(X_i) = \text{Pa}_g(X_i)) + \right. \\ &\quad \left. + \ln \text{MargL}^q(X_i, \text{Pa}_g(X_i) : \mathcal{D}) + \right. \\ &\quad \left. + \ln \text{MargL}^\theta(X_i, \text{Pa}_g(X_i) : \mathcal{D}) \right] = \\ &= \sum_{X_i} \text{famscore}_B(X_i, \text{Pa}_g(X_i) : \mathcal{D}). \end{aligned} \quad (3.9)$$

Si è quindi definita la funzione di scoring come una somma di score bayesiani, $\text{famscore}_B(X_i, \text{Pa}_g(X_i) : \mathcal{D})$, relativi ai nodi del grafo \mathcal{G} . Ognuno di tali score bayesiani misura la qualità di $\text{Pa}_g(X_i)$ come insieme dei nodi genitori di X_i , dato l'insieme dei dati di apprendimento \mathcal{D} .

3.2 RICERCA DELLA STRUTTURA

In questa sezione si affronta il secondo passo del processo di apprendimento strutturale: l'utilizzo di una *procedura di ottimizzazione* finalizzata alla ricerca di una struttura che massimizzi lo *score bayesiano*.

Chickering (1994) ha mostrato come il problema di apprendere la struttura ottimale di una Bayesian Network, detto problema *k-learn*, dove k è il numero massimo di genitori per ogni variabile casuale, sia un problema NP-completo anche qualora si imponga $k = 2$. La ragione di tale complessità è dovuta al vincolo di aciclicità delle BN (i.e., il grafo di una BN, come da definizione 1.1, deve essere un DAG):

non è perciò possibile determinare l'insieme ottimale dei genitori di ogni nodo di una BN individualmente; poiché la scelta di un insieme di genitori per un nodo restringe la possibilità di scelta relativa ai nodi restanti.

Come già accennato ed intuibile dalla composizione dello score bayesiano (si veda la funzione $\text{famscore}_{\mathcal{B}}$, equazione 3.9), la ricerca della struttura ottimale di un modello Continuous time Bayesian Network (CTBN) è invece notevolmente più semplice rispetto a quello relativo alle BN (o alle DBN). La motivazione di tale vantaggio risiede nel fatto che, poiché gli archi fra i nodi del grado \mathcal{G} di una CTBN rappresentano l'effetto nel tempo del valore attuale della variabile casuale padre sul valore futuro della variabile casuale figlia, non esiste un vincolo di aciclicità ed è di conseguenza possibile ottimizzare l'insieme dei nodi genitori di un qualsiasi nodo separatamente dagli altri.

Inoltre, qualora si restringa il massimo numero di genitori a un valore k , per ogni variabile casuale X_i di una CTBN, con $i = 1, \dots, N$, si può semplicemente enumerare ogni suo possibile insieme di nodi genitori $\text{Pa}(X_i)$ tale che $|\text{Pa}(X_i)| \leq k$ e calcolarne il rispettivo punteggio $\text{famscore}_{\mathcal{B}}(X_i, \text{Pa}(X_i) : \mathcal{D})$. Infine scegliere come insieme dei nodi genitori di X_i quello con punteggio massimo.

Si definisce perciò il seguente teorema (Nodelman *et al.*, 2002).

Teorema 3.1 (Problema k -learn). *Il problema k -learn per le Continuous time Bayesian Network, fissato k , può essere risolto in tempo polinomiale rispetto al numero di variabili casuali N e alla dimensione dell'insieme di dati \mathcal{D} .*

Si osservi che, fissando k a priori, non è necessario enumerare esaurientemente tutti i possibili insiemi di nodi genitori di ogni nodo di una CTBN. È quindi possibile utilizzare un algoritmo di ricerca euristica di tipo *greedy* per esplorare lo spazio di ricerca. Nella sottosezione 3.2.1 si presenta l'algoritmo scelto per l'apprendimento strutturale della struttura ottimale di un modello CTBN.

3.2.1 Hill Climbing

L'algoritmo *hill climbing* è una tecnica di ricerca euristica finalizzata alla risoluzione di problemi di ottimizzazione i cui stati (i.e., elementi dello spazio di ricerca) contengono tutte le informazioni necessarie a costituire una soluzione (Russell e Norvig, 2003).

L'idea su cui si basa tale algoritmo consiste nell'iniziare la ricerca con una soluzione sub-ottimale e, nei passi successivi, migliorare iterativamente la soluzione, osservando gli stati vicini, finché una qualche condizione di fermata venga raggiunta (e.g., l'algoritmo non può migliorare ulteriormente la soluzione corrente). Il processo di miglioramento è, come già accennato, basato sulla valutazione dello stato corrente tramite una funzione di punteggio.

A differenza di altri algoritmi di ricerca basati sul miglioramento iterativo della soluzione (e.g., *simulated annealing*, *tabu search*), l'algoritmo *hill climbing* si sposta sempre in uno stato che fornisce una soluzione con maggior punteggio (Russell e Norvig, 2003). L'utilizzo di tale algoritmo garantisce il raggiungimento di soluzioni localmente ottime (i.e., soluzioni che non possono essere migliorate considerando solamente la configurazione degli stati vicini) in una quantità di tempo relativamente bassa. Tuttavia esso non garantisce il raggiungimento di una soluzione che costituisca l'ottimo globale, a meno che la funzione rappresentante lo spazio di ricerca non sia convessa. Ciò avviene poiché l'algoritmo smette di effettuare progressi verso la soluzione globalmente ottima nel momento in cui il vicinato della soluzione corrente non permette alcun miglioramento immediato.

Per sorpassare tale limitazione è possibile attuare varie strategie (si veda Russell e Norvig, 2003), quali, ad esempio: l'esplorazione iterativa, a partire da diverse configurazioni iniziali, dello stesso spazio di ricerca (i.e., *random restart hill climbing*); la selezione stocastica del vicinato da esaminare ad ogni passo della ricerca locale, sulla base della probabilità che un dato vicinato porti a un progresso maggiore rispetto ad altri vicini; oppure la scelta di soluzioni non migliorative finalizzata ad una maggiore esplorazione dello spazio di ricerca (i.e., *simulated annealing*).

Nota 3.2.1.1. Questo algoritmo opera una gestione efficiente della memoria poiché non necessita il mantenimento di alcun albero di ricerca: esso conserva solamente l'informazione (i.e., punteggio della soluzione) sullo stato corrente e quello successivo.

L'algoritmo 3.1 illustra la tecnica di ricerca *hill climbing* dato uno spazio degli stati discreto.

```

1 function hillclimbing(problem) {
2   var current_state = start_state(problem)
3   while (true) {
4     var nb = neighbors(current_state)
5     var next_eval = -inf
6     var next_state = null
7     for (x in nb) {
8       var x_score = score(x)
9       if (x_score > next_eval) {
10        next_state = x
11        next_eval = x_score
12      }
13    }
14    if (next_eval ≤ score(current_state)) {
15      break
16    }
17    current_state = next_state

```

```

18     }
19     return current_state
20 }

```

Algoritmo 3.1: Algoritmo *hill climbing* per uno spazio degli stati discreto.

Di seguito si discute l'applicazione di tale algoritmo all'apprendimento strutturale delle CTBN.

Come detto, a causa della mancanza del vincolo di aciclicità, è possibile eseguire la succitata procedura di ottimizzazione in modo indipendente per ogni singolo nodo del modello CTBN in esame. Ciò permette di scomporre il problema dell'apprendimento strutturale in un insieme di problemi della stessa entità, di minore complessità e completamente indipendenti fra di essi; contesto ottimo per un approccio parallelizzato alla risoluzione del problema padre.

Dato uno qualsiasi dei succitati sotto-problemi, lo spazio degli stati che l'algoritmo *hill climbing* può esplorare è composto da tutti i possibili insiemi di genitori con cardinalità minore o uguale a k , il numero massimo di genitori di ogni singolo nodo della CTBN. L'algoritmo *hill climbing* individua l'insieme di genitori ottimale per il nodo in esame, valutando, iterativamente, i possibili insiemi di nodi genitori con cardinalità minore e maggiore di 1 rispetto alla cardinalità dell'insieme di genitori corrente. Si osservi che tale configurazione prevede che l'insieme dei genitori con cardinalità pari a 0 (i.e., insieme vuoto \emptyset) venga anch'esso valutato.

La funzione di *scoring* con cui tali insiemi di genitori vengono valutati (funzione **score**, algoritmo 3.1) corrisponde alla funzione $\text{famscore}_{\mathcal{B}}$, presentata nella sezione 3.1 a pagina 34.

Uno degli obiettivi di questo lavoro di tesi è consistito nella creazione di un framework in linguaggio R per le CTBN.

Perciò, in questo capitolo si descrivono i principali aspetti del pacchetto RCTBN, preposto a tale scopo.

Tuttavia, prima di presentare e chiarire le funzionalità offerte da tale pacchetto, è necessario introdurre il contesto per cui esso è stato pensato, progettato e implementato: il linguaggio R.

La discussione del presente capitolo segue quindi tale logica.

4.1 IL LINGUAGGIO R

R è un linguaggio di programmazione *interpretato* (e al contempo un ambiente di sviluppo) pensato per applicazioni di tipo *statistico*. Esso incorpora di default una vastissima gamma di *funzionalità statistiche* (e.g., modelli di regressione, test statistici, analisi delle serie temporali, classificazione, clustering) e di tecniche inerenti la *manipolazione dei dati* (e.g., lazy loading¹⁷, strutture dati di tipo tabulare, procedure di calcolo matriciale efficaci) utili allo sviluppo di applicazioni e modelli per l'*analisi dei dati* (R Core Team, 2013).

R supporta principalmente il paradigma di *programmazione funzionale con scoping lessicale*¹⁸. Tuttavia esso può essere esteso al fine di utilizzare il paradigma di *programmazione orientata agli oggetti*.

Lo scoping lessicale, insieme ad altre peculiarità di R, quali ad esempio il supporto per le *closure* o le funzioni di prima classe¹⁹, fanno sì che esso costituisca un ottimo strumento per la creazione di *software scientifico*, così come di applicazioni che richiedono tempi d'esecuzione elevati. In tali casi, infatti, è spesso auspicabile che sia possibile verificare a priori l'eseguibilità o meno del codice sorgente almeno dal punto di vista sintattico (Oliveira e Stewart, 2006).

¹⁷ Il lazy loading è un design pattern comunemente utilizzato al fine di deferire l'inizializzazione, o il caricamento, di un qualsiasi oggetto finché ciò non sia strettamente necessario.

¹⁸ Con il termine *scoping lessicale*, o *scoping statico*, ci si riferisce a una determinata modalità di valutazione delle variabili in un dato linguaggio di programmazione. Nello specifico, in un linguaggio con scoping lessicale, il riferimento a un nome di variabile viene risolto basandosi ricorsivamente sulla struttura sintattica che incorpora la definizione di tale nome di variabile. Ne consegue che l'associazione di un valore a una variabile non avviene al momento dell'applicazione e dell'esecuzione bensì durante l'analisi sintattica del sorgente.

¹⁹ Una funzione di prima classe è una funzione che restituisce un'altra funzione. Essa costituisce la più diffusa modalità d'applicazione delle *closure*.

Una ulteriore caratteristica di R è costituita dalla sua natura altamente modulare. Esso è infatti pensato per invogliare l'utilizzatore alla creazione di pacchetti, cioè di porzioni di codice R riutilizzabili. A tale scopo R fornisce una serie di funzionalità finalizzate alla creazione, pubblicazione e condivisione con la comunità di pacchetti R. Infatti, tali moduli, una volta sviluppati, possono essere distribuiti in un apposito archivio online di pacchetti R, chiamato CRAN²⁰. Questa caratteristica, parallelamente alle precedenti, costituisce una delle principali cause dell'ampia adozione di tale linguaggio di programmazione per l'implementazione di software scientifico.

4.1.1 Estendere R

Poiché lo scopo di questo capitolo è presentare il succitato pacchetto RCTBN, in questa sottosezione si descrive brevemente il processo di estensione di R, attuabile, come detto, tramite la creazione di moduli chiamati pacchetti.

Un pacchetto R consiste in un insieme di cartelle e file che rispettano determinate convenzioni. Nello specifico, data una cartella radice che si intende utilizzare come contenitore del sorgente di un qualsiasi pacchetto R, è necessario che essa contenga almeno i seguenti elementi:

- una sotto cartella R/, in cui è necessario inserire i sorgenti in linguaggio R del pacchetto
- un file DESCRIPTION, che descriva il pacchetto, il suo scopo, le sue dipendenze da altri pacchetti, la licenza con cui viene distribuito e l'autore.

La struttura di un pacchetto R prevede ulteriori elementi opzionali:

- una sotto cartella man/, preposta alla documentazione del pacchetto
- un file NEWS per la descrizione dei cambiamenti in ogni versione del pacchetto tramite il formato standard che R fornisce a tale scopo
- un file README che contenga una panoramica generale del pacchetto
- un file inst/CITATION finalizzato alla descrizione delle modalità con cui citare il corrente pacchetto in lavori scientifici
- una sotto cartella demo/ contenente delle applicazioni d'esempio che utilizzano il pacchetto stesso

²⁰ Il The Comprehensive R Archive Network (CRAN) costituisce la principale fonte di moduli aggiuntivi (i.e., pacchetti) per R. Esso è raggiungibile all'indirizzo: <http://cran.r-project.org>.

- una sotto cartella `inst/doc` contenente la documentazione approfondita e eventuali manuali d'utilizzo
- un file `NAMESPACE`, che descriva quali funzioni devono far parte della API del pacchetto cosicché esse siano utilizzabili dagli utenti
- le sotto cartelle `test/` e `inst/test/`, contenenti i test per il pacchetto, allo scopo di assicurarsi che esso operi come progettato
- una sotto cartella `data/`, preposta all'incorporazione nel pacchetto di eventuali dataset d'esempio
- una sotto cartella `src/` per il codice sorgente non in linguaggio R (e. g., C++, C o FORTRAN)
- una sotto cartella `exec/` preposta alla distribuzione di ulteriori script eseguibili
- una sotto cartella `po/` che contenga i file di traduzione per il pacchetto.

È possibile creare la struttura di un pacchetto R manualmente o usufruendo della funzione base `package.skeleton`. Inoltre, a titolo informativo, si evidenzia l'esistenza di un pacchetto esterno finalizzato alla facilitazione di tale processo, il pacchetto `devtools` (Wickham e Chang, 2013) (si veda l'appendice A a pagina 106 per maggiori dettagli a riguardo).

Infine, si fornisce una panoramica delle pratiche consigliate (o necessarie) riguardanti gli elementi obbligatori per la creazione di un pacchetto R.

La sotto cartella `R/` non impone alcun vincolo sull'organizzazione del codice sorgente che essa deve contenere, al di là del fatto che esso deve essere codice in linguaggio R. Tuttavia, quando si intende sviluppare utilizzando l'approccio funzionale, venendo meno, almeno esplicitamente, il concetto di classe, è pratica consolidata e consigliata organizzare il codice sorgente in diversi file in base all'area tematica cui appartengono le funzioni che essi contengono.

Invece, il file `DESCRIPTION`, al fine di definire i metadati del pacchetto R, deve contenere determinati campi.

Il sorgente 4.1 illustra, tramite un esempio di file `DESCRIPTION` per `RCTBN`, tale pratica.

```
Package: ctbn
Title: The CTBN framework
Description:
Version: 0.1
Author: Leonardo Di Donato <leodidonato@gmail.com>
Maintainer: Leonardo Di Donato <leodidonato@gmail.com>
```

```

Depends:
    R (>= 3.0),
    ...
License: GPL (>= 2)
Collate:
    ...
LinkingTo: ...

```

Sorgente 4.1: Esempio di possibile file di descrizione del pacchetto RCTBN.

Di seguito si elencano i sei campi che tale file di testo deve obbligatoriamente contenere:

- il campo `Package`, nel quale si definisce il nome del pacchetto (e dovrebbe, in teoria, corrispondere al nome della cartella radice)
- il campo `Title`, contenente a una descrizione sintetica del pacchetto (massimo una linea di testo)
- il campo `Description`, preposto a una descrizione maggiormente dettagliata del pacchetto e delle sue funzionalità
- il campo `Version`, contenente il numero di versione, auspicabilmente nel formato `major.minor.patchlevel`
- il campo `Maintainer`, contenente un singolo nome e indirizzo e-mail per la persona responsabile della manutenzione del pacchetto
- il campo `License`, contenente una abbreviazione standard di una licenza open source.

4.2 ANALISI

In questa sezione si descrive per grandi linee il pacchetto RCTBN.

Lo scopo di tale pacchetto, come detto, consiste nell'implementazione del framework delle CTBN: esso espone perciò una collezione di funzioni (i. e., API) correlate ai modelli CTBN e CTBNC.

Al fine di presentare le funzionalità di tale pacchetto, di seguito, si riportano i requisiti cui esso aderisce.

R1 – GESTIONE DEI DATASET

Il pacchetto deve permettere di caricare nel minore tempo possibile insiemi di dati completi (eventualmente di dimensioni considerevoli) presenti sul disco rigido dell'utente. Il sistema deve fornire all'utente un'interfaccia che semplifichi questo processo nascondendo e automatizzando la scelta dei percorsi di ricerca e memorizzazione dei dataset.

R1.1 – IMPORTAZIONE

Il pacchetto deve consentire di importare insiemi di *dati completi*. Gli insiemi di dati corrispondono a insiemi di file posti nella medesima cartella. Tali file possono essere in qualsiasi formato testuale (e.g., CSV, TSV, TXT) purché essi contengano una struttura tabulare, specificabile dall'utente. Il sistema in esame deve controllare che determinati vincoli, propri del modello in questione, siano rispettati cosicché ogni dataset importato possa realmente rispecchiare un modello CTBN. Si elencano i controlli necessari:

- controllare che ogni file contenga i dati relativi all'evoluzione temporale (i.e., colonna temporale)
- controllare che ogni file contenga una colonna classe e che il suo valore sia univoco per ogni file, nel caso l'utente lo richieda
- controllare che ogni file contenga delle traiettorie di dati completi.

R1.2 – SERIALIZZAZIONE

Il sistema deve consentire all'utente di comprimere e serializzare il dataset importato, in modo automatico o manuale, su disco rigido.

R1.2 – CARICAMENTO VELOCE

Il sistema deve permettere, in qualsiasi momento, il caricamento veloce di un qualsiasi dataset precedentemente importato e serializzato.

Si osservi che tali processi devono poter essere eseguiti anche in sessioni di lavoro diverse.

R2 – APPRENDIMENTO DI UN MODELLO CTBN

Il pacchetto deve consentire l'apprendimento dei parametri di un modello CTBN rispetto a un insieme di dati completi precedentemente importato e processato. La natura del modello (i.e., la struttura del grafo) che l'utente intende apprendere deve essere specificata ad ogni esecuzione.

R2.1 – COMPUTAZIONE STATISTICHE SUFFICIENTI

Il sistema deve permettere il calcolo delle *statistiche sufficienti* consentendo all'utente di selezionare l'insieme di nodi (e i rispettivi insiemi di genitori) da sottoporre a tale processo, come mostrato nella sottosezione 1.4.1 a pagina 14.

R2.2 – COMPUTAZIONE IPER-PARAMETRI

Il sistema deve consentire il calcolo degli *iper-parametri* di un modello CTBN a partire da delle statistiche sufficienti precedentemente calcolate, come mostrato nella sottosezione 1.4.3 a pagina 16.

R2.3 – COMPUTAZIONE MATRICI DI INTENSITÀ CONDIZIONALI

Il pacchetto deve permettere il calcolo delle *matrici di intensità* a partire dagli iper-parametri.

R3 – APPRENDIMENTO STRUTTURALE DI UN CTBN

Il pacchetto deve consentire l'apprendimento della struttura di un modello CTBN dato un insieme di dati completo. L'utente deve poter specificare, qualora lo desideri, l'approccio da utilizzare (attualmente si fornisce solo un approccio basato su punteggio, quello descritto nel capitolo 3 a pagina 33).

R3.1 – RICERCA EURISTICA

Il sistema deve fornire un modulo preposto alla ricerca euristica della struttura CTBN ottimale. Attualmente è prevista solamente la procedura di ricerca *hill climbing* con memoizzazione²¹ (a tal riguardo si rimanda alla sottosezione 3.2.1 a pagina 37).

R3.2 – FUNZIONE DI PUNTEGGIO

Il sistema deve fornire una funzione di punteggio delle strutture CTBN (attualmente è prevista solamente la computazione dello *score bayesiano*, presentata nella sezione 3.1 a pagina 34).

R4 – CLASSIFICAZIONE TRAMITE MODELLI CTBNC

Il pacchetto deve permettere, tramite l'utilizzo di un classificatore CTBNC, la *classificazione supervisionata* di traiettorie multivariate che evolvono nel tempo continuo. L'utente deve immettere nel sistema l'istanza per cui inferire la classe di appartenenza e selezionare il classificatore da utilizzare per classificarla.

R4.1 – APPRENDIMENTO DI UN CLASSIFICATORE CTBN

Tale processo incorpora quello relativo al punto R2. In aggiunta, esso deve computare la probabilità a priori della variabile classe sull'insieme di dati completi di input. Il pacchetto deve fornire sia una versione generale del processo di apprendimento, funzionante per qualsiasi classificatore CTBN (i.e., algoritmo 2.2 presentato a a pagina 27), sia una versione dedicata esclusivamente ai classificatori CTNB (i.e., algoritmo 2.1 presentato a a pagina 25).

²¹ La *memoizzazione* è una tecnica di ottimizzazione delle chiamate delle funzioni finalizzata al miglioramento dei tempi d'esecuzione di un sistema software e al risparmio di risorse computazionali. Essa consiste nel salvare in memoria il risultato ottenuto dalla chiamata di una funzione, comprensiva di parametri attuali, in modo da non dover effettuare nuovamente la computazione in caso la funzione in questione venga nuovamente chiamata con gli stessi parametri attuali. Nel contesto dell'apprendimento strutturale di una CTBN questa tecnica può essere usata con efficacia evitando di computare nuovamente il punteggio di un modello CTBN già processato precedentemente.

R4.2 – INFERENZA TRAMITE CLASSIFICATORE CTBN

Il sistema deve consentire all'utente, dato un qualsiasi classificatore CTBN, di classificare una qualsiasi istanza completa dei dati assegnando ad essa la classe che ottiene il valore maggiore di probabilità secondo l'algoritmo 2.3 presentato a a pagina 31.

Si osservi che è stata tralasciata la discussione di una serie di ulteriori requisiti e relative funzionalità, quali ad esempio quelle relative all'esecuzione della *cross-validazione* e al calcolo delle *metriche di valutazione* dei classificatori, poiché esse riguardano direttamente l'argomento primario di questo lavoro di tesi e, inoltre, sono state implementate separatamente come pacchetti R indipendenti da RCTBN.

La principale entità che tale pacchetto rappresenta corrisponde al modello CTBN. Tale entità è composta dal grafo ad essa sottostante e dall'insieme dei parametri e eventualmente delle matrici di intensità relative ad ogni nodo di tale grafo, secondo la definizione 1.15 a pagina 12.

Parimenti, RCTBN può rappresentare anche i modelli CTBNC, che costituiscono, come specificato dalla definizione 2.1 a pagina 22, una specializzazione del modello CTBN. Il vincolo che tale specializzazione apporta corrisponde all'esistenza di un particolare nodo radice: il nodo classe.

Si noti infine che ognuna delle succitate macro funzionalità corrisponde a un componente del pacchetto RCTBN e viene implementata separatamente dalle altre.

Quindi, al fine di chiarire ulteriormente l'architettura del pacchetto RCTBN e le relazioni che intercorrono tra i componenti da cui esso è costituito, la figura 4.1 nella pagina successiva ne illustra il relativo diagramma dei componenti. Tale diagramma evidenzia le interfacce di comunicazione fra i vari componenti e la loro corrispondenza rispetto alle funzionalità descritte precedentemente.

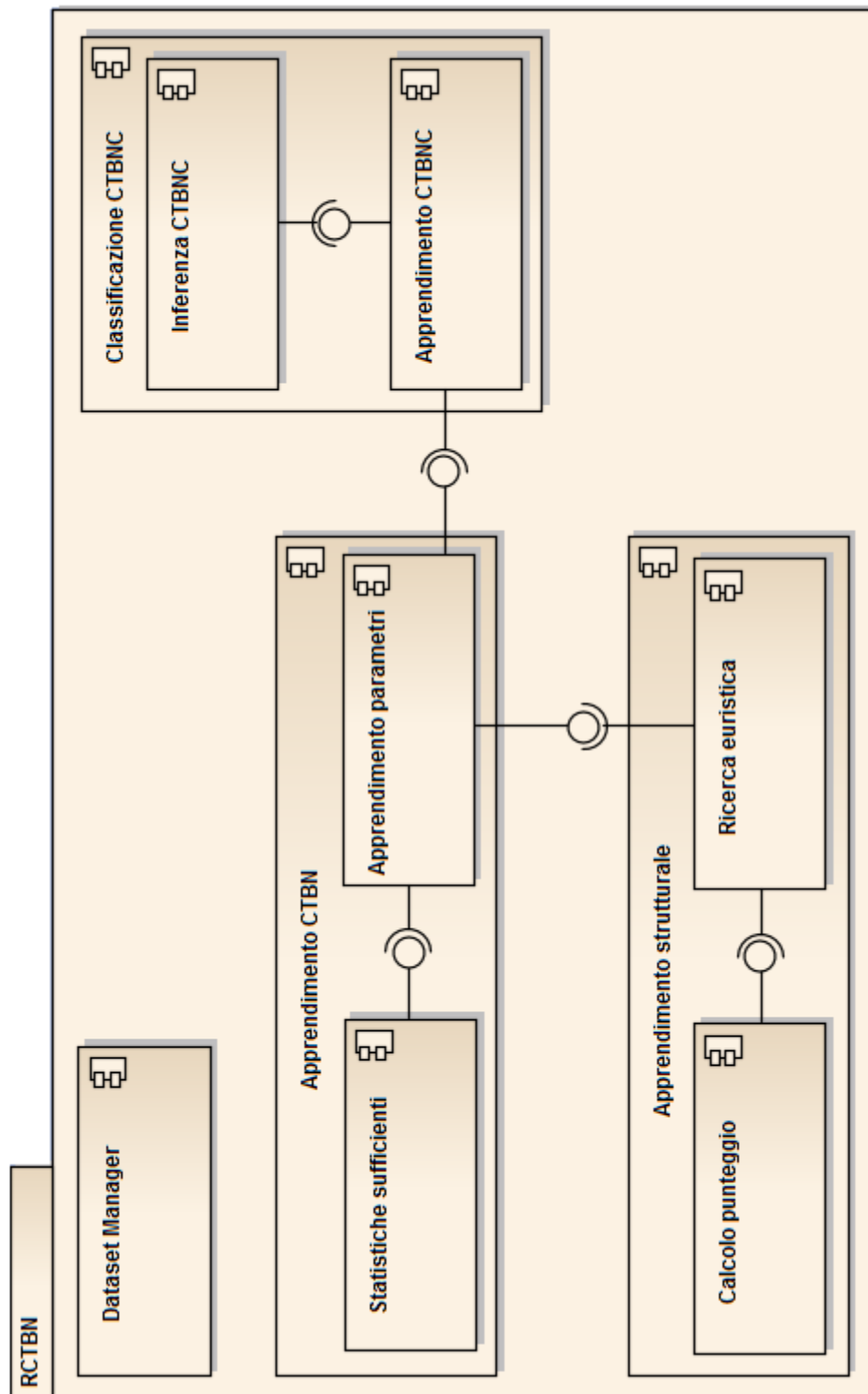


Figura 4.1: Diagramma dei componenti di RCTBN.

5 | STRUMENTI PER LA CREAZIONE DI DATASET

Al fine di valutare le prestazioni degli algoritmi di apprendimento e classificazione delle Continuous time Bayesian Network è emersa la necessità di generare dei dataset adeguati a tale scopo.

Come già specificato in precedenza, le CTBN sono un modello stocastico dedito alla rappresentazione dell'evoluzione di sistemi dinamici, cioè di fenomeni che evolvono nel tempo, rappresentati come insiemi di traiettorie multi-variate. Si è quindi scelto di generare dei dataset che rappresentassero un tipico sistema dinamico complesso: il traffico automobilistico su rete urbana.

Tale scelta non è finalizzata al mero esercizio delle tecniche di apprendimento e classificazione descritte e sviluppate, bensì è motivata dalla necessità di individuare delle possibili strategie di ottimizzazione del problema del traffico.

Infatti, il traffico intenso è un problema che affligge tutte le grandi città quotidianamente. La congestione delle reti urbane incide negativamente sulla qualità della vita. Il tempo perso in coda dagli automobilisti ha ripercussioni in primis sulle attività economiche. Alcuni studi a riguardo, risalenti al 2004, stimano le perdite del sistema economico italiano da esso derivanti in 6,4 MLD di € annui (Certet, 2004). A ciò si aggiunga l'impatto negativo del traffico sull'ambiente e quindi sulla salute: l'aumento dei tassi di diossina nell'aria è correlato all'aumento delle malattie respiratorie (e.g., asma) e immunitarie (e.g., allergie), o peggio all'insorgenza di tumori (Mantecca *et al.*, 2007; Gualtieri *et al.*, 2005). Inoltre, elevati livelli di traffico sono fonte di stress da cui possono derivare ulteriori complicazioni. Le problematiche citate evidenziano i costi ingenti che il problema del traffico impone alla collettività. Ne deriva la necessità di affrontare tale problema per cercare di ridurre l'impatto negativo. Tra gli approcci (elencati esaustivamente in Papageorgiou *et al.*, 2005) finalizzati al miglioramento delle condizioni del traffico urbano emerge, per il grande impatto atteso in termini di benefici, l'ottimizzazione dei piani semaforici in base alle condizioni di traffico.

L'ottimizzazione semaforica è un problema complesso: pochi degli approcci proposti in letteratura (Gershenson, 2008; Felici *et al.*, 2006; Park e Messer, 1998; Thorpe e Anderson, 1996) possono essere effettivamente applicati nella realtà, seppur in seguito a grosse semplificazioni dovute alla difficoltà computazionale che li caratterizza. Un modo per risolvere questo problema, non nuovo alla letteratura (si veda Angulo *et al.*, 2011), è semplificare il problema dividendolo in due passi:

- classificazione del profilo di traffico
- selezione del piano semaforico ad esso associato.

In questa tesi affrontiamo il primo passo: la classificazione dei profili di traffico, propedeutico all'ottimizzazione del problema descritto. Inoltre, l'algoritmo di apprendimento strutturale, descritto nel capitolo 3 a pagina 33, fornisce la mappa delle relazioni nel tempo tra i vari sensori della rete stradale; ciò si traduce in uno strumento di osservazione e controllo della stessa.

Con l'ausilio di un software commerciale, Traffic Software Integrated System (TSIS) (versione ≥ 6.2), e di una sua estensione a tempo d'esecuzione appositamente sviluppata al fine di monitorare e tracciare il passaggio dei veicoli, sono stati generati dei dataset contenenti un insieme di documenti rappresentanti la presenza (durante l'evolvere del tempo) di veicoli sui sensori di una rete stradale.

Si precisa che i sensori cui ci si riferisce sono le spire, cioè dei sensori ad induttanza magnetica, posti sotto l'asfalto, che generano un'informazione, detta onda quadra, così composta: stato pari a 1 se un veicolo è presente sul sensore, stato pari a 0 viceversa. La figura 5.1 illustra visivamente quanto appena descritto.

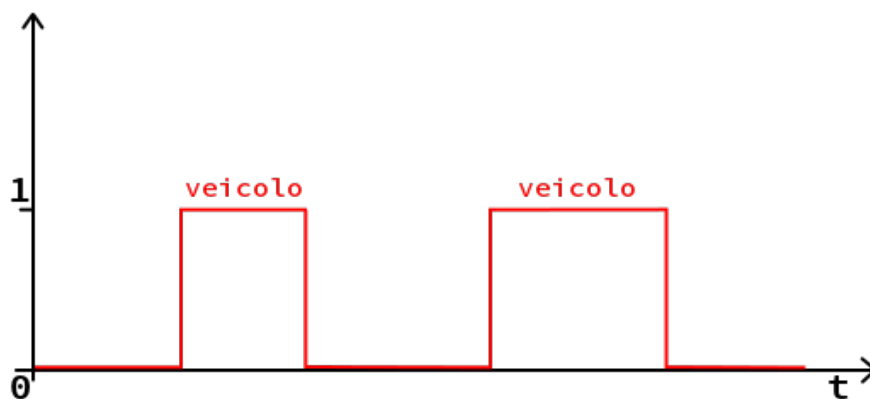


Figura 5.1: Rappresentazione dell'onda quadra generata da un sensore a spira magnetica.

Poiché TSIS supporta esclusivamente la generazione di output aggregati rispetto al tempo (e. g., densità dei veicoli per strada, velocità media, conteggio dei veicoli per sensore), si è reso necessario lo sviluppo di una estensione a tempo d'esecuzione dedicata che fosse in grado di memorizzare il dato atomico dei sensori al fine di ottenere la succitata onda quadra a tempo continuo.

In questo capitolo si presentano sia i succitati strumenti utilizzati per la creazione di reti stradali e relativi modelli di simulazione, sia l'estensione ideata per la creazione di dataset relativi al traffico.

Relativamente, invece, al processo pratico di creazione dei dataset si rimanda alla sezione A.2.

5.1 TSIS

Traffic Software Integrated System (TSIS) è un ambiente di sviluppo integrato²², distribuito commercialmente da McTrans²³ e supportato dalla Federal Highway Administration (FHWA)²⁴, il cui scopo ultimo è permettere la simulazione e l'analisi di modelli di reti stradali.

TSIS è costituito da insieme di strumenti dedicati alla creazione di reti stradali e relativi modelli di simulazione, all'esecuzione, e eventualmente alla visualizzazione, di tali modelli, così come all'interpretazione dei risultati ottenuti. Tale insieme di strumenti è reso accessibile tramite delle interfacce grafiche²⁵.

L'architettura modulare con cui TSIS è realizzato permette, in caso di necessità, di estendere tale ambiente creando degli ulteriori strumenti.

Di seguito si introducono brevemente i concetti relativi a TSIS utilizzati nel prosieguo di questo lavoro di tesi.

Tuttavia si osservi che, poiché lo scopo di questa sezione non consiste nel documentare TSIS, la sua trattazione esaustiva (e.g., semantica e lista completa dei tipi di dati rappresentabili) è omessa. A tale fine si rimanda invece alla documentazione ufficiale del software in questione.

Definizione 5.1 (Progetto TSIS). Un progetto TSIS è un insieme di modelli di simulazione per una specifica rete stradale.

Definizione 5.2 (Modello di simulazione TSIS). Un modello di simulazione è costituito da un input (e.g., variazioni dei flussi di ingresso nella rete, variazioni delle percentuali di svolta dei veicoli nelle intersezioni) per la simulazione di una determinata rete stradale e tutti i dati generati dalla sua esecuzione (i.e., simulazione).

Osservazione 5.2.1. Un modello di simulazione può anche prevedere che la sua esecuzione sia eseguita più volte. Finché il seme dei numeri casuali non è modificato, esso è sempre considerato un singolo modello di simulazione.

Definizione 5.3 (Formato TRF). TRF²⁶ è il formato dei file accettati dal simulatore di TSIS. Esso codifica e rappresenta una rete stradale e

²² Un ambiente di sviluppo integrato, comunemente chiamato anche Integrated Development Environment (IDE), è un insieme di programmi finalizzati a supportare il processo di sviluppo dei software. Generalmente, un IDE è costituito da uno strumento per la creazione e modifica del codice sorgente, un compilatore o un interprete, strumenti per l'automazione dello sviluppo e la qualità del codice sorgente.

²³ McTrans Moving Technology: <http://mctrans.ce.ufl.edu>.

²⁴ Agenzia del Dipartimento dei Trasporti degli Stati Uniti d'America: www.fhwa.dot.gov.

²⁵ Un'interfaccia grafica, nota anche come Graphical User Interface (GUI), è un tipo di interfaccia utente il cui fine è permettere all'utente di interagire con il software manipolando oggetti grafici convenzionali.

²⁶ Traffic File (TRF).

il relativo modello di simulazione specificandone i vari componenti tramite l'utilizzo dei rispettivi RT (si veda la definizione 5.5).

Osservazione 5.3.1. Il formato TRF è equivalente al formato TRAF²⁷.

Definizione 5.4 (Formato TNO). TNO²⁸ è il formato nativo con cui vengono rappresentate in memoria le reti stradali create visivamente tramite l'interfaccia grafica TRAFED.

Definizione 5.5 (Record Type). Un Record Type (RT) rappresenta il blocco informativo minimo su cui è costruita una rete stradale e il suo modello di simulazione. Nel concreto esso consiste in una singola riga di testo nei file TRF contenente un codice identificativo numerico e dei valori per i rispettivi campi accettati nell'ordine prestabilito. Allo stesso modo, tale formato descrive anche il modello di simulazione della rete stradale.

Definizione 5.6 (RTE). Una Run-Time Extension (RTE), estensione a tempo d'esecuzione, è un'applicazione in grado di comunicare a tempo d'esecuzione con un'altra applicazione esterna. Una RTE, in ambiente *Microsoft Windows*, è solitamente compilata separatamente sotto forma di DLL²⁹ e deve rispettare una determinata interfaccia: deve perciò implementare e esportare determinate funzioni che l'applicazione oggetto della comunicazione chiama a tempo d'esecuzione. Per la comunicazione in ingresso, invece, una RTE deve essere collegata alle librerie dell'applicazione con cui intende interfacciarsi, avendo così accesso alle strutture dati e alle funzioni che questa esporta.

5.1.1 Componenti

In questa sezione si elencano gli strumenti che costituiscono l'ambiente di sviluppo TSIS.

CORSIM

CORSIM³⁰ costituisce il componente principale dell'insieme di strumenti denominato TSIS. È un simulatore il cui obiettivo è permettere la creazione e l'esecuzione di modelli di simulazione TSIS. È composto da due simulatori integrati che rappresentano l'intero sistema di traffico come funzione del tempo: NETSIM³¹

²⁷ Traffic File (TRAF).

²⁸ TRAFED Native Object (TNO).

²⁹ Una libreria a collegamento dinamico (DLL) è una libreria software che viene caricata dinamicamente in fase di esecuzione, invece di essere collegata staticamente a un eseguibile in fase di compilazione. Queste librerie sono anche chiamate Dynamic-link library (DLL). L'acronimo DLL corrisponde all'estensione che tali oggetti hanno in ambiente *Microsoft Windows*. Tuttavia esse sono spesso chiamate più genericamente con il termine librerie condivise (da *shared library*). Nei sistemi *Linux*, esse sono anche note come oggetti *shared object*.

³⁰ Corridor microscopic simulation program (CORSIM).

³¹ Network Simulator (NETSIM).

e FRESIM³². Tali simulatori integrati rappresentano, rispettivamente, il traffico sulle strade urbane e non. La simulazione effettuata da tali strumenti è di tipo microscopico: essi modellano individualmente il comportamento di ogni singolo veicolo, prendendo in considerazione per ognuno di essi una serie di variabili, anche di tipo stocastico (e.g., tipologia di guidatore). Per tale motivo CORSIM è dotato di molte possibili opzioni di configurazione e permette lo studio di modelli molto complessi e dettagliati.

TRAFED

TRAFED³³ è una GUI il cui scopo è permettere la creazione e la modifica di reti stradali e di modelli di simulazione per CORSIM.

TShell

TShell³⁴ è la GUI di TSIS. Funge da contenitore degli strumenti (preconfigurati, o creati dall'utente) di questo ambiente di sviluppo integrato e permette la gestione dei progetti TSIS.

TRAFVU

TRAFVU³⁵ è una GUI finalizzata alla visualizzazione dei modelli di simulazione simulati con CORSIM. Essa permette sia di visualizzare in modo animato l'evoluzione del traffico nella rete stradale con una qualsiasi granularità temporale, sia di visualizzare una serie di misure di interesse relative alla simulazione.

TSIS TEXT EDITOR

TSIS Text Editor è uno strumento il cui scopo è facilitare la modifica manuale dei file TRF. A tale scopo esso visualizza per ogni RT che si intende modificare sia la sua descrizione sia l'insieme dei campi supportati.

TSIS SCRIPT TOOL

TSIS Script Tool è uno strumento per la creazione, la modifica e l'esecuzione di codice VBScript³⁶. Questo strumento fornisce un meccanismo utile ad automatizzare le funzionalità di simulazione di TSIS (e.g., esecuzioni multiple dello stesso modello di simulazione variando il seme dei numeri casuali che governa la distribuzione di ingresso dei veicoli nella rete stradale).

TSIS TRANSLATOR

TSIS Translator è uno strumento utile alla conversione dei file dal

³² Freeway Simulator (FRESIM).

³³ TRAF Editor (TRAFED).

³⁴ TSIS Shell (TShell).

³⁵ TRAF Visualization Utility (TRAFVU).

³⁶ Microsoft's Visual Basic Scripting Edition (VBScript) è un linguaggio interpretato, sottoinsieme del linguaggio *Visual Basic*, utilizzato come sostituto o integrazione della linea di comando o per il controllo di applicazioni esterne in ambiente *Microsoft Windows*.

formato TRF al formato TNO e viceversa. Tale operazione risulta utile al fine di rendere i file TRF utilizzabili tramite lo strumento TRAFED così come per rendere i file TNO utilizzabili con CORSIM.

TSIS OUTPUT PROCESSOR

TSIS Output Processor è uno strumento finalizzato alla raccolta e l'aggregazione dei dati da CORSIM durante l'esecuzione multipla di modelli di simulazione. La sua caratteristica principale consiste nella computazione automatica di un insieme di statistiche predefinite. Esso permette di scegliere sia le statistiche di interesse sia la granularità temporale della loro computazione.

5.1.2 Caratteristiche

Segue una panoramica il cui scopo è presentare sia le principali capacità, sia i vincoli di modellazione, simulazione e analisi di TSIS.

TSIS, tramite CORSIM, permette la modellazione di reti stradali con le seguenti caratteristiche:

- reti stradali urbane, non urbane o miste
- controllo quantitativo (i. e., quantità di veicoli per intervallo temporale, distribuita secondo una distribuzione statistica) e qualitativo (i. e., tipo di veicoli) dei flussi di veicoli in ingresso nella rete stradale
- controllo completo del flusso di traffico (i. e., percentuali di svolta dei veicoli variabile nel tempo)
- supporto per strade con più corsie (massimo 9)
- supporto dei segnali stradali
- supporto per gli attraversamenti pedonali
- intersezioni non controllate
- intersezioni controllate da semafori
- canalizzazione delle corsie
- piani semaforici predefiniti, dinamici (e.g., priorità ai veicoli di emergenza) o guidati da algoritmi (e.g., attivazione del semaforo in base alle rilevazioni effettuate dai sensori)
- sensori per la rilevazione del passaggio o della presenza di veicoli
- supporto per i mezzi di trasporto pubblico (i. e., autobus, taxi)
- rilevamento di incidenti

- supporto per i sistemi di guida anglosassoni
- simulazione guidata all'analisi delle code
- simulazione guidata allo studio del grado di occupazione della rete stradale
- output di dati di interesse aggregati
- output di dati statistici collezionati

Poiché un modello di simulazione del traffico è caratterizzato dal cambiamento di un insieme di condizioni (e.g., volumi di traffico, canalizzazione delle corsie, percentuali di svolta dei veicoli) della rete stradale, esso deve specificare, oltre alla natura stessa dei cambiamenti, anche i fattori in base a cui essi avvengono. Il fattore primario che viene preso in considerazione è il tempo. Ne consegue perciò che un modello di simulazione del traffico deve specificare l'intervallo temporale in cui specifici cambiamenti di determinate condizioni avvengono.

CORSIM affronta questo problema permettendo di partizionare il tempo totale di simulazione in una serie di periodi temporali (i.e., *time period*) di durata variabile. Ogni periodo temporale possiede perciò un insieme di dati di input che non variano per tutta la sua durata. Inoltre, i periodi temporali sono a loro volta suddivisi in intervalli temporali (i.e., *time interval*), anch'essi suddivisi in passi temporali (i.e., *time step*).

Le principali limitazioni di TSIS derivano dalle modalità con cui CORSIM implementa la gestione del tempo: la lista dei periodi temporali è rappresentata internamente tramite un array statico di dimensione prefissata. Nello specifico, un modello CORSIM può essere costituito da un massimo di 19 periodi temporali, ognuno dei quali viene specificato tramite il RT 03, e può avere una durata compresa tra i 10 e i 9999 secondi. Ognuno di tali periodi temporali è suddiviso in intervalli (specificati tramite dei RT 04) la cui durata, compresa tra 1 e 200 secondi, deve essere un sottomultiplo della durata del periodo temporale cui appartiene. Infine, ogni intervallo temporale è partizionato in passi temporali: NETSIM utilizza un passo temporale fisso di 1 secondo mentre FRESIM opera in base al passo temporale specificato dall'utente. Poiché i due modelli di simulazione microscopica che costituiscono CORSIM operano in modo sincronizzato, la durata del passo temporale di FRESIM, anche se compresa nell'intervallo $[0.1, 1]$ secondi, non può essere specificata liberamente dall'utente. All'utente è permesso specificare solo il numero di passi temporali di FRESIM (tramite il campo 1, RT 04) che devono essere eseguiti per ogni secondo di simulazione. Tale approccio permette perciò di utilizzare dei passi temporali di durata minore per il simulatore FRESIM mantenendo la sincronizzazione con il simulatore NETSIM.

La tabella 5.1 mostra la corrispondenza tra il numero di passi temporali di FRESIM e la reale durata che verrà loro assegnata.

Valore del campo 1	Durata del passo temporale in FRESIM (sec)
1	1.0
2	0.5
3	0.333333
4	0.25
5	0.2
6	0.166667
7	0.142857
8	0.125
9	0.111111
10	0.1

Tabella 5.1: Relazione tra numero di passi temporali per ogni secondo di simulazione e durata effettiva del passo temporale in FRESIM.

La figura 5.2 nella pagina seguente mostra, tramite un esempio, quanto appena descritto relativamente alla rappresentazione del tempo in CORSIM.

Si osservi, inoltre, che la granularità temporale massima con cui è possibile ottenere statistiche cumulative, e, in generale, dati relativi alla simulazione, corrisponde alla durata scelta per gli intervalli temporali (si veda a tal riguardo la documentazione relativa al tipo RT 05). Si consideri ad esempio la figura 5.2 nella pagina successiva: in tal caso non sarà possibile ottenere dati cumulati relativi a un intervallo di tempo minore di 60 secondi.

La gestione del tempo che CORSIM attua implica quindi alcune limitazioni:

- il tempo massimo di simulazione, benché sufficiente nella maggior parte dei casi, è di circa 52 ore
- la massima granularità con cui è possibile raccogliere informazioni è di 1 secondo, impostando a tale valore la durata degli intervalli temporali
- i dati raccolti dalla simulazione sono sempre aggregati in base alla durata dell'intervallo temporale cui si riferiscono.

Tali vincoli rendono impossibile recuperare l'output dei sensori a tempo continuo in modo non aggregato o in generale con una granularità temporale inferiore a 1 secondo. Al fine di sorpassare tali limitazioni si è proceduto sviluppando una estensione a tempo d'esecuzione apposita, *Sensors DLL*, descritta dettagliatamente nella sezione 5.3 a

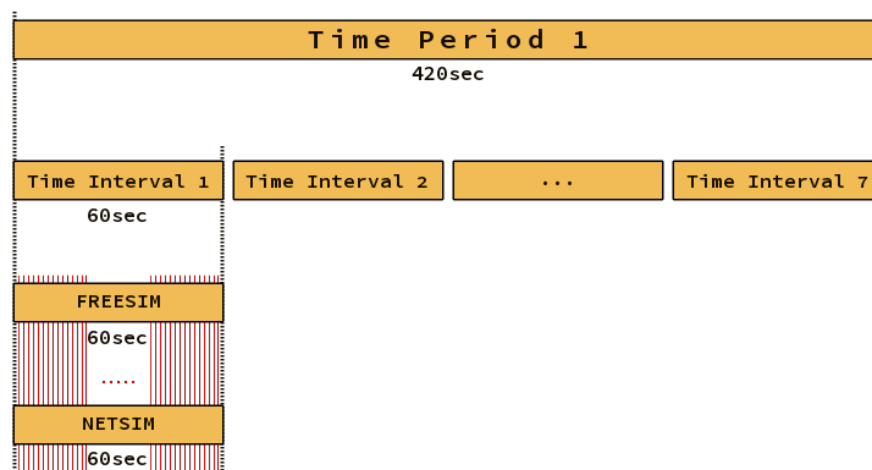


Figura 5.2: Esempio di suddivisione gerarchica delle unità temporali in CORSIM. Un *time period* della durata di 420 secondi è suddiviso in 7 *time interval*, ciascuno della durata di 60 secondi. Ogni *time interval* verrà poi automaticamente diviso in 60 *time step* da 1 secondo ciascuno per NETSIM (i.e., linee rosse). La durata di tali *time step* di NETSIM può o meno coincidere, come accade in questo esempio, con quella assegnata ai *time step* di FRESIM a seconda del valore del campo 1 del RT 04 nel modello CORSIM.

pagina 66. Al fine di rendere la descrizione di tale software maggiormente chiara, è necessario presentare il meccanismo di estensione di CORSIM. La sezione che segue affronta perciò tale argomento.

5.2 CREAZIONE DI ESTENSIONI TSIS

TSIS espone un meccanismo finalizzato all'estensione delle sue funzionalità tramite la creazione, da parte dell'utente, di altri strumenti da integrare nell'ambiente di sviluppo. Tali strumenti, interfacciandosi direttamente con CORSIM, possono modificarne o aumentarne la logica di simulazione, collezionare dati o monitorare eventi speciali (e.g., incidenti).

In questa sottosezione si presenta il funzionamento dei meccanismi di interfacciamento (i.e., più brevemente detti API³⁷) tra CORSIM e strumenti esterni, a cui ci si riferirà da questo momento in poi con il termine CORSIM RTE.

³⁷ Con il termine Application Programming Interface (API) si indica un insieme di procedure rese disponibili all'esterno, di solito raggruppate a formare un insieme di strumenti specifici per l'espletamento di un determinato compito all'interno di un programma.

5.2.1 Requisiti

Al fine di sviluppare e compilare con successo una CORSIM RTE in C++ è necessario disporre dei seguenti strumenti:

- il compilatore della piattaforma *Microsoft Visual C++*
- il pacchetto software di TSIS, il quale include di default tutti i componenti necessari mostrati dalla figura 5.3 nella pagina successiva (ad eccezione, chiaramente, del componente RTE).

Si osservi, inoltre, che è possibile sviluppare una CORSIM RTE anche in linguaggio C o FORTRAN.

5.2.2 Architettura di CORSIM

La figura 5.3 nella pagina seguente illustra l'architettura modulare di CORSIM e il suo funzionamento all'interno dell'ambiente di sviluppo TSIS. Si osservi che i componenti che costituiscono CORSIM sono di due tipi: librerie DLL e moduli COM³⁸. Il CORSIM Driver Component, ad esempio, è il modulo COM di TSIS preposto a interfacciare CORSIM e TShell, permettendo così il controllo e l'esecuzione di CORSIM, delle RTE create dall'utente e degli altri strumenti (e. g., TSIS Output Processor) di TSIS tramite GUI.

Anche se la figura 5.3 nella pagina successiva mostra per completezza l'intera architettura di CORSIM, si procede con la descrizione delle interfacce di CORSIM preposte alla comunicazione con RTE sviluppate dall'utente, identificate dalle frecce tratteggiate e numerate.

Per ogni passo temporale di simulazione, il CORSIM Server chiama una serie di funzioni di CORSIM finalizzate a guidare l'andamento della simulazione. Quando una RTE viene inserita nell'ambiente di sviluppo integrato, il CORSIM Server chiama anche le funzioni che la RTE esporta in base ai messaggi che riceve da CORSIM durante la sua esecuzione. Questa interfaccia è rappresentata dalla freccia 1 nella figura 5.3 nella pagina seguente; la sottosezione 5.2.3 a pagina 59 riporta maggiori dettagli sui punti di chiamata che CORSIM espone.

TSIS fornisce inoltre una API, identificata dalla freccia 2 nella figura 5.3 nella pagina seguente, chiamata CORWIN, che permette alle RTE di inviare messaggi al modulo CORSIM Server affinché essi siano visualizzati in TShell dal CORSIM Driver Component.

Infine, una RTE può accedere direttamente a una serie di funzioni e strutture dati esportate da CORSIM nella memoria condivisa. Anche se non è possibile riferirsi a questo meccanismo di comunicazione come una API vera e propria, nel prosieguo, ci riferiremo ad essa con il

³⁸ Il Component Object Model (COM) è uno standard per componenti software ideato da *Microsoft*. Il suo fine consiste nel permettere la comunicazione fra processi e la creazione dinamica di oggetti. Una interfaccia COM è una collezione di funzioni, incapsulata in un componente software binario e neutrale rispetto al linguaggio.

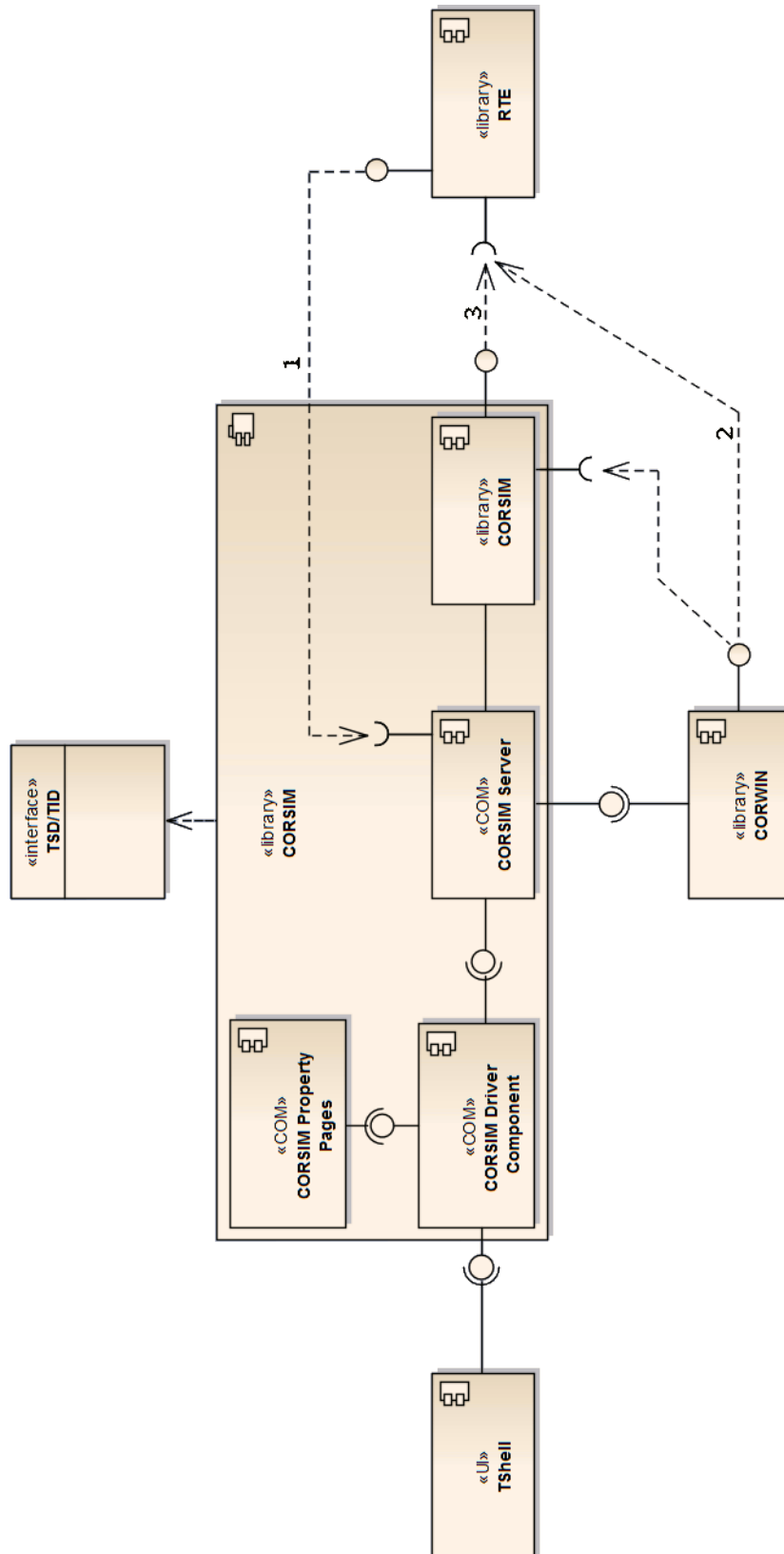


Figura 5.3: Porzione del diagramma dei componenti di TSIS: mostra l'architettura modulare e il funzionamento di CORSIM all'interno di TSIS.

termine CORSIM API al fine di semplificare la discussione. Perciò, la CORSIM API, rappresentata dalla freccia 3 della figura 5.3 nella pagina precedente, oltre a permettere l'estrazione di informazioni relative alla simulazione, permette alla RTE di controllare, eventualmente, molti aspetti della simulazione operata da CORSIM (e. g., aborto della simulazione).

Si osservi che, anche se la figura 5.3 nella pagina precedente non evidenzia tale possibilità, l'architettura di CORSIM supporta l'utilizzo di più RTE contemporaneamente.

Nota 5.2.1. Un attento osservatore noterà come CORSIM, la libreria finalizzata al processo di simulazione, sia a sua volta una RTE automaticamente collegata ai moduli CORSIM Server e CORWIN. Inoltre, la figura 5.4 nella pagina successiva fa notare come tutti i componenti (elencati e descritti nella sottosezione 5.1.1 a pagina 51) dell'ambiente di sviluppo TSIS siano anch'essi dei moduli architetturalmente uguali alle RTE.

5.2.3 Ciclo di vita di CORSIM

Di seguito si presenta il ciclo di vita di CORSIM descrivendo i punti di chiamata che esso esporta tramite apposite funzioni affinché una RTE, implementando ed esportando una funzione per almeno uno di essi, possa interfacciarsi con il processo di simulazione. Tali informazioni sono quindi relative alla API rappresentata dalla freccia 1 nella figura 5.3 nella pagina precedente. Le modalità di implementazione e utilizzo in C++ di tale API sono illustrate nella sottosezione 5.2.5 a pagina 64.

La tabella 5.2 a pagina 63 descrive tutti i punti di chiamata relativi alla linea di esecuzione temporale di CORSIM. Una volta compilata la RTE e ottenuto il relativo file DLL, ognuno dei punti di chiamata di CORSIM deve essere associato alla rispettiva funzione implementata dalla RTE. La sezione che segue descrive in maggior dettaglio tale processo.

5.2.4 Collegare una RTE a CORSIM

Questa sottosezione illustra i passi necessari a espletare il processo di collegamento (i. e., *linking*) di una RTE a CORSIM. Tale processo è eseguibile direttamente tramite TShell.

A scopo esemplificativo si descrive come aggiungere la RTE per il rilevamento e tracciamento del passaggio dei veicoli sui sensori (descritta nella sezione 5.3 a pagina 66). Tale operazione viene svolta tramite il menù *Tools* di TShell scegliendo la voce *Tool Configuration* e cliccando sul pulsante per l'aggiunta (i. e., *Add*) di una RTE. La figura 5.4 nella pagina successiva mostra lo strumento di configurazione degli strumenti appartenenti all'ambiente di TSIS.

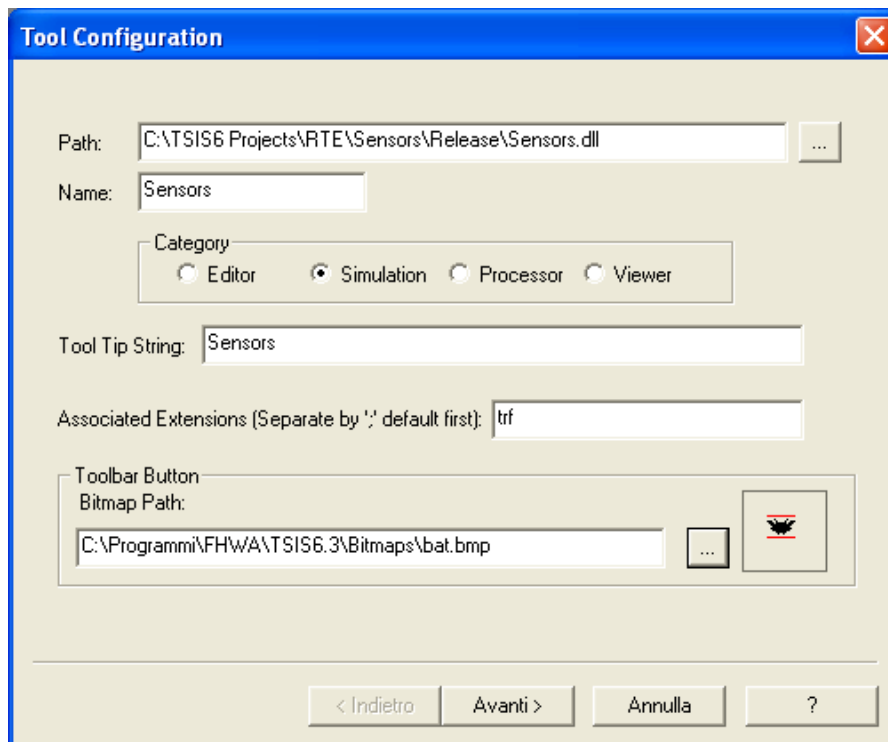


Figura 5.4: Strumento finalizzato all'aggiunta di una RTE a TSIS.

Specificato il percorso a cui risiede il file DLL della RTE, il tipo di RTE, il nome e l'icona che si intende assegnare a tale strumento e il tipo di file a cui va associato (e. g., TRF), la RTE è aggiunta a TSIS. La figura 5.5 mostra la barra degli strumenti di TShell quando un file TRF viene aperto: essa contiene il pulsante per l'avvio della RTE appena aggiunta all'ambiente di sviluppo TSIS.



Figura 5.5: Barra degli strumenti di TShell contenente il pulsante per l'invocazione della RTE aggiunta all'ambiente di sviluppo TSIS.

Tuttavia, come detto, le funzioni della RTE devono essere collegate ai punti di chiamata di CORSIM affinché la RTE risulti completamente funzionante. Per adempiere tale operazione è necessario utilizzare nuovamente lo strumento di configurazione cliccando sul pulsante per la modifica (i. e., *Edit*) di una RTE. A questo punto, selezionando la scheda relativa alle RTE, è possibile invocare lo strumento per effettuare il succitato collegamento. La figura 5.6 nella pagina successiva mostra il processo di collegamento tra le funzioni della RTE e i punti di chiamata di CORSIM.

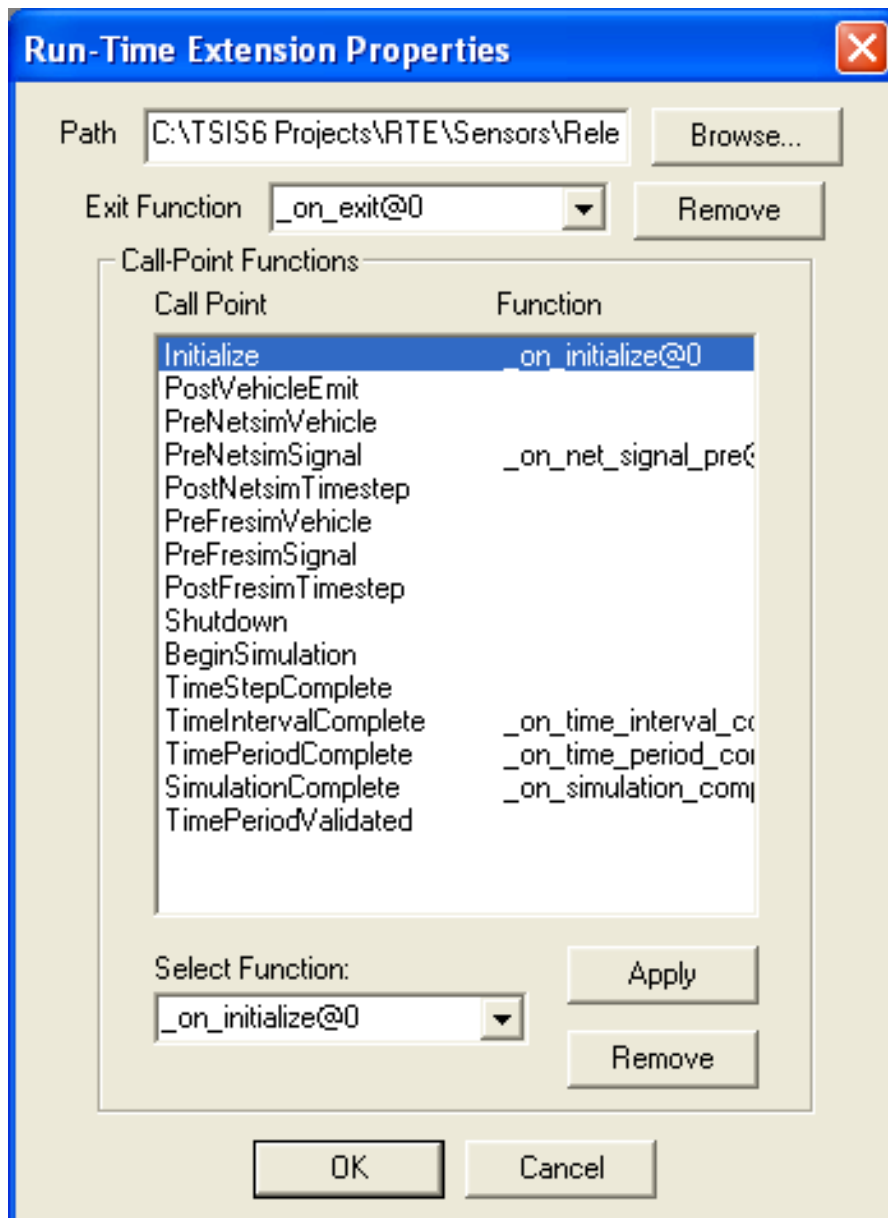
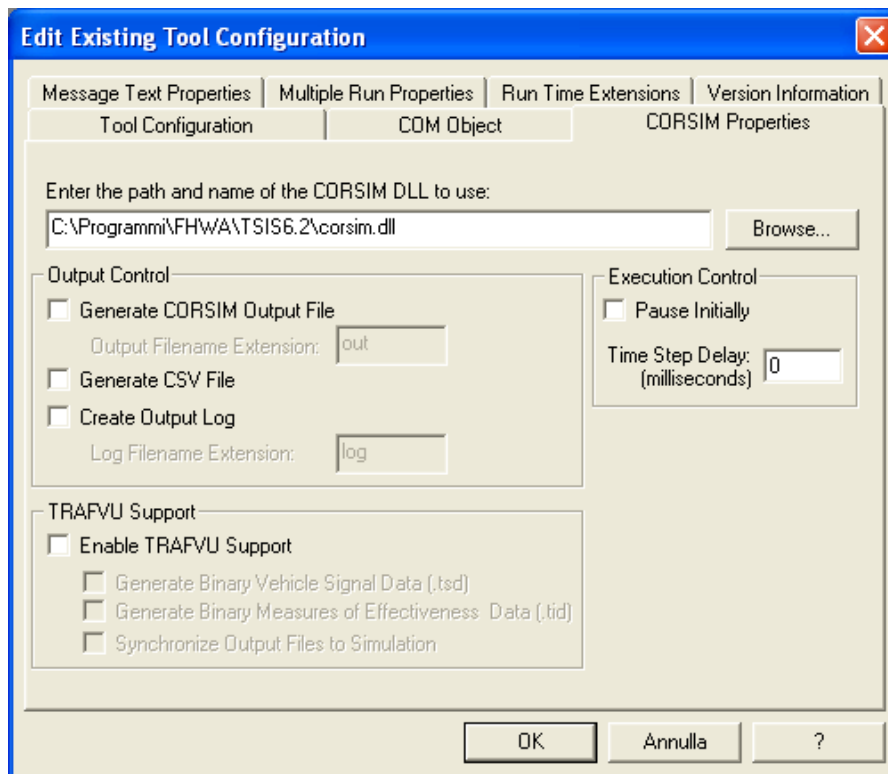
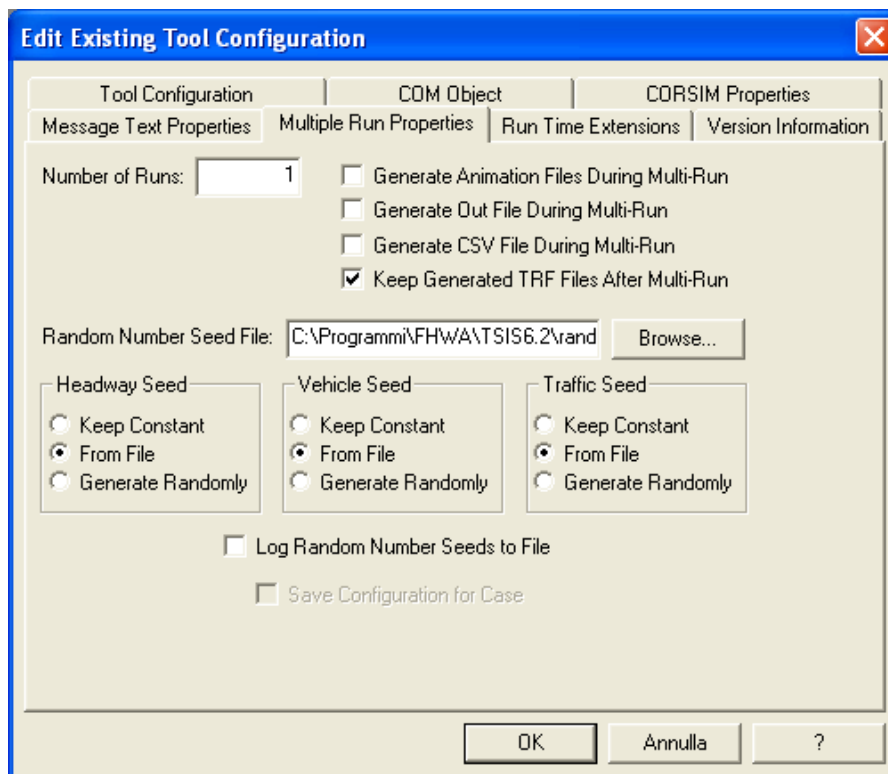


Figura 5.6: Collegamento delle funzioni della RTE ai rispettivi punti di chiamata di CORSIM.

Inoltre, può essere necessario dover configurare la RTE aggiunta in base alle sue esigenze, così come modificare alcune funzionalità di CORSIM relativamente ad essa. Ad esempio, la RTE per il monitoraggio e il tracciamento del passaggio dei veicoli sui sensori non necessita che i file di output di CORSIM vengano generati, né che vengano generati i file per la visualizzazione animata della simulazione in TRAFED. Inoltre, tale RTE, non necessita che la simulazione CORSIM sia eseguita più volte. La figura 5.7 nella pagina seguente mostra la configurazione di tali opzioni.



(a) Proprietà di CORSIM.



(b) Impostazioni proprietà d'esecuzione di CORSIM.

Figura 5.7: Configurazione delle proprietà di CORSIM per la RTE: disattivazione dell'output di CORSIM, della generazione dell'input per TRAFED e delle esecuzioni multiple della simulazione.

Punto di chiamata	Descrizione
Initialize	Chiamato all'inizio della simulazione prima della fase di inizializzazione CORSIM ma dopo la lettura del file TRF di input.
PostVehicleEmit	Chiamato ad ogni <i>time step</i> dopo che i veicoli sono stati immessi nella rete stradale.
PreNetsimVehicle	Chiamato ad ogni <i>time step</i> appena prima che i veicoli inizino a muoversi nel sotto-modello NETSIM.
PreNetsimSignal	Chiamato ad ogni <i>time step</i> appena prima che i segnali (e.g., semafori) del sotto-modello NETSIM vengano impostati.
PostNetsimTimestep	Chiamato in corrispondenza della fine del processo di simulazione di ogni <i>time step</i> di NETSIM e prima che FRESIM inizi a simulare il suo relativo sotto-modello.
PreFresimVehicle	Chiamato ad ogni <i>time step</i> appena prima che i veicoli inizino a muoversi nel sotto-modello FRESIM.
PreFresimSignal	Chiamato ad ogni <i>time step</i> appena prima che i segnali (e.g., semafori) del sotto-modello FRESIM vengano impostati.
PostFresimTimestep	Chiamato in corrispondenza della fine del processo di simulazione di ogni <i>time step</i> di FRESIM.
BeginSimulation	Chiamato dopo l'inizializzazione di CORSIM (i.e., rete stradale piena) in corrispondenza dell'inizio della simulazione.
TimeStepComplete	Chiamato in corrispondenza della fine del processo di simulazione di ogni <i>time step</i> .
TimeIntervalComplete	Chiamato in corrispondenza della fine del processo di simulazione di ogni <i>time interval</i> .
TimePeriodComplete	Chiamato in corrispondenza della fine del processo di simulazione di ogni <i>time period</i> .
TimePeriodValidated	Chiamato in corrispondenza della fine del processo di lettura e validazione del file di input relativo a ogni <i>time period</i> , prima dell'inizializzazione e dell'effettivo inizio del processo di simulazione di ogni <i>time period</i> .
SimulationComplete	Chiamato in corrispondenza della fine della simulazione e prima della completa terminazione del processo.
Shutdown	Chiamato appena prima che l'esecuzione di CORSIM termini.
Exit	Chiamato in corrispondenza della fine dell'intero processo di simulazione.

Tabella 5.2: Descrizione di punti di chiamata che CORSIM espone all'esterno.

5.2.5 Utilizzo delle API

Lungi dal volere fornire una documentazione esaustiva delle API dell'ambiente di sviluppo TSIS, in questa sezione, si intende presentare, tramite esempi, le modalità di utilizzo di tali API in linguaggio C++.

Ad esempio, il listato 5.1 mostra il codice di intestazione necessario a definire ed esportare la funzione `on_initialize` (listato 5.2), che, come mostrato dalla figura 5.6 a pagina 61 viene collegata al punto di chiamata `Initialize` di `CORSIM`. Si osservi che la scelta del nome di tale funzione non è vincolata ad alcun criterio.

```

1 #ifdef __cplusplus
2 #define DLL_EXPORT extern "C" __declspec(dllexport)
3 #endif // __cplusplus

```

Sorgente 5.1: Costrutto per l'esportazione delle funzioni RTE

```

1 DLL_EXPORT void __stdcall on_initialize() {
2     // implementation
3 }

```

Sorgente 5.2: Esempio di funzione RTE esportata

Il listato 5.3 illustra invece come importare una delle funzioni esposte dalla `CORWIN` API in una RTE. Nello specifico, l'esempio in questione, è relativo all'importazione della funzione `OutputString`, finalizzata all'invio di messaggi di output a `TShell` durante la simulazione `CORSIM`.

```

1 #ifdef __cplusplus
2 #define CORWINAPI extern "C" __declspec(dllimport)
3 CORWINAPI void __stdcall OutputString(const char *str,
4     unsigned int size, int msgCode, unsigned long color);
5 #endif // __cplusplus

```

Sorgente 5.3: Importazione delle `CORWIN` API

Infine, come già detto, `CORSIM` permette l'accesso alla maggior parte dei dati che esso manipola durante la simulazione, esportandoli nella memoria condivisa. Tali dati sono dei seguenti tipi:

1. variabili scalari (non array)
2. array allocati staticamente
3. array allocati dinamicamente
4. funzioni esposte tramite API.

Il listato 5.4 mostra il costrutto utilizzabile per l'importazione di dati e funzioni dalla cosiddetta CORSIM API.

```

1 #ifdef __cplusplus
2 #define DLL_IMPORT extern "C" __declspec(dllimport)
3 #endif // __cplusplus

```

Sorgente 5.4: Costrutto per l'importazione delle CORSIM API

Tale costrutto viene poi utilizzato per l'effettiva importazione di dati e funzioni da CORSIM. Il listato 5.5 riporta un esempio di importazione per ogni tipo di dati che le CORSIM API rendono disponibile:

1. a linea 2 si importa il numero di sensori presenti sulla rete stradale urbana, esportato da CORSIM tramite la variabile scalare NETSIM_DETECTORS_mp_NUMDET e rinominato, a linea 3, in net_det_num
2. a linea 5 si importa la lista statica (i.e., di dimensione massima prefissata) dei numeri identificativi assegnati dall'utente ai nodi della rete stradale, SIN075.NMAP; rinominata a linea 9 in net_node_num
3. a linea 11 si importa la lista dinamica delle informazioni sui sensori, *NETSIM_DETECTORS_mp_DETMOD; rinominata, a linea 12, in net_det_mod
4. a linea 14 si importa abortcorsim, una funzione delle CORSIM API finalizzata al controllo dell'esecuzione della simulazione da parte della RTE

```

1 // netsim scalar non-array variables
2 DLL_IMPORT int NETSIM_DETECTORS_mp_NUMDET;
3 #define net_det_num NETSIM_DETECTORS_mp_NUMDET
4 // netsim statically allocated arrays
5 DLL_IMPORT struct
6 {
7     int NMAP[8999];
8 } SIN075;
9 #define net_node_num SIN075.NMAP
10 // netsim dynamically allocated arrays
11 DLL_IMPORT int *NETSIM_DETECTORS_mp_DTMODD;
12 #define net_det_id NETSIM_DETECTORS_mp_DTMOD
13 // netsim exported functions
14 DLL_IMPORT void __stdcall abortcorsim(void);

```

Sorgente 5.5: Importazione di oggetti delle CORSIM API

5.3 ESTENSIONE

Avendo presentato l'ambiente di sviluppo TSIS e le API che esso fornisce per la sua estensione, è ora possibile descrivere l'estensione a tempo d'esecuzione sviluppata al fine di generare i succitati dataset.

Come detto nella sottosezione 5.1.2 a pagina 53, una delle principali limitazioni di CORSIM consiste nell'impossibilità di ottenere dei dati non aggregati dal processo di simulazione. Questo perché il minimo intervallo temporale che CORSIM permette di utilizzare è pari a 1 secondo per le reti stradali urbane e al più 0.1 secondi per le reti stradali extraurbane. Perciò, è emerso il problema di sorpassare tale limitazione. La RTE sviluppata, Sensors DLL, risponde a tale necessità monitorando determinati elementi (i. e., i sensori) di un qualsiasi tipo di rete stradale e tracciando gli eventi ad essi correlati (i. e., passaggio di un veicolo) su un file di output esterno a TSIS. Lo scopo di questa sezione consiste nel presentare il funzionamento di Sensors DLL.

5.3.1 Sensors DLL

L'obiettivo ultimo di Sensors DLL consiste nella generazione di un file CSV³⁹ che rappresenti il passaggio dei veicoli sui vari sensori presenti nella rete stradale nel tempo. Il monitoraggio dei sensori deve essere effettuato con la massima granularità temporale possibile (i. e., 0.1 secondi), anche nel caso di reti stradali urbane.

A tale scopo si è utilizzato il dato NETSIM_DETECTORS_mp_DET_ON, rinominato in net_det_on, esportato da CORSIM nella memoria condivisa. Tale campo indirizza un array dinamico la cui lunghezza è pari al numero di sensori sulla rete stradale. Ognuno degli elementi di tale array è costituito da 10 bit: l'i-esimo bit rappresenta l'attivazione (i. e., 1) o meno del relativo sensore nell'i-esimo passo temporale minimo (i. e., 0.1 secondi). Ogni elemento rappresenta perciò il passaggio dei veicoli sul sensore durante un intervallo temporale fisso di 1 secondo.

Di seguito si presentano le operazioni principali che Sensors DLL effettua:

1. ottenere il nome del file TRF di input, rappresentante la rete stradale e il modello di simulazione
2. effettuare il *parsing*⁴⁰ di tale file creando gli oggetti relativi a ogni elemento (e. g., intersezioni, strade, sensori) della rete stradale
3. rilevare il passaggio dei veicoli sui sensori ogni secondo

³⁹ Comma Separated Values (CSV) è un formato basato su file di testo utilizzato per l'importazione ed esportazione (ad esempio da fogli elettronici o database) di una tabella di dati.

⁴⁰ Il *parsing* consiste nel processo atto ad analizzare un input in modo da determinare la sua struttura grammaticale grazie ad una data grammatica formale.

4. ricostruire l'intero flusso di veicoli su ogni sensore durante tutto il tempo di simulazione
5. creare un file di output che contenga le informazioni ottenute.

Le operazioni 1 e 2 vengono compiute in corrispondenza dell'inizializzazione di CORSIM e quindi della RTE. Il risultato di tali operazioni è un insieme di istanze correlate rappresentanti gli elementi della rete stradale di input e le caratteristiche di ognuno di essi. Il diagramma delle classi di Sensors DLL, mostrato in figura 5.8 a pagina 69, illustra le relazioni di associazione e aggregazione degli oggetti con cui si è scelto di rappresentare le reti stradali TSIS. La classe CNetwork rappresenta la rete stradale, composta da un insieme di intersezioni e strade, elementi rappresentati rispettivamente dalle classi CNode e CLink. Ogni strada può a sua volta essere composta da più corsie, elementi rappresentati tramite la classe CLane, e contenere dei sensori, elementi rappresentati tramite la classe CDetector. Inoltre, poiché è possibile che alcuni sensori siano posti esclusivamente su una corsia piuttosto che su tutta la superficie della strada, sussiste una relazione anche fra la classe rappresentante le corsie e quella rappresentante i sensori. Invece, la classe CBinary non rappresenta alcun elemento concreto della rete stradale: la sua funzione è esclusivamente quella di incapsulare l'intero `net_det_on` e convertirlo nella corretta sequenza di bit rappresentante il flusso dei veicoli su un sensore. La procedura che effettua tali operazioni di inizializzazione della rete stradale nella RTE, chiamata `on_initialize`, è collegata al punto di chiamata `Initialize`, così come mostrato dalla figura 5.6 a pagina 61.

Configurata la rete stradale, Sensors DLL può monitorare i sensori di pari passo con l'esecuzione della simulazione da parte di CORSIM. Tale operazione viene effettuata ad ogni intervallo temporale di NETSIM poiché la procedura che la incorpora, chiamata `on_net_signal_pre`, è collegata al punto di chiamata `PreNetsimSignal` di CORSIM. Il listato 5.6 a pagina 68 illustra una versione semplificata del metodo C++ preposto all'esecuzione di tale procedura. Essa consiste nell'iterazione della lista di sensori afferenti ad una strada: per ogni sensore (ciclo a linea 8), recuperato l'identificatore che CORSIM utilizza per rappresentarlo (istruzione a linea 10), si ottiene il relativo elemento dell'array `net_det_on` (istruzione a linea 11), un intero rappresentante il flusso di veicoli sul sensore nell'ultimo secondo di simulazione. Tale intero viene poi convertito nella corretta sequenza di 10 bit tramite la classe CBinary a linea 12. Da linea 13 a linea 32 si itera in ordine inverso la sequenza di bit al fine di estrapolare e memorizzare lo stato (i.e., 1 in caso di veicolo rilevato, 0 altrimenti) del sensore in ogni passo temporale minimo (i.e., 0.1 secondi). Quindi questa procedura, ripetuta per tutte le strade presenti nella rete stradale e ad ogni intervallo temporale, memorizza per ogni sensore una lista di valori booleani.

Completata la simulazione e di conseguenza anche la procedura di monitoraggio dei sensori, Sensors DLL, in corrispondenza del punto di chiamata *SimulationComplete* di CORSIM, genera un file CSV in cui memorizza il tempo, il *time period* della rilevazione e la dinamica di stato di ogni sensore. La sottosezione 5.3.2 nella pagina successiva si occupa di presentare in maggior dettaglio l'output di Sensors DLL.

```

1 void CLink::processDetectors(void) {
2     int det, new_state, old_state = 0;
3     POSITION pos, pos_i = NULL;
4     CDetector *detector = NULL;
5     CInteger *pi = NULL;
6     CBinarySequence *sequence = NULL;
7     pos = m_detector_list.GetHeadPosition();
8     while (pos != NULL) {
9         detector = m_detector_list.GetNext(pos);
10        int det_num = detector->getCorsimId();
11        det = net_det_on[det_num];
12        sequence = CBinarySequence::convert(det);
13        pos_i = sequence->sequence.GetTailPosition();
14        old_state = detector->getState() ? 1 : 0;
15        for (index = 0; index < 10; index++) {
16            pi = sequence->sequence.GetPrev(pos_i);
17            new_state = pi->data;
18            if ((old_state == 0) && (new_state == 1)) {
19                detector->setState(true, !is_init);
20            }
21            if ((old_state == 1) && (new_state == 1)) {
22                float atime = detector->getActivationTime();
23                detector->setState(true, !is_init);
24            }
25            if ((old_state == 1) && (new_state == 0)) {
26                detector->setState(false, !is_init);
27            }
28            if ((old_state == 0) && (new_state == 0)) {
29                detector->setState(false, !is_init);
30            }
31            old_state = new_state;
32        }
33        delete sequence;
34    }
35 }

```

Sorgente 5.6: Metodo della classe CLink per la rilevazione del passaggio dei veicoli sui sensori

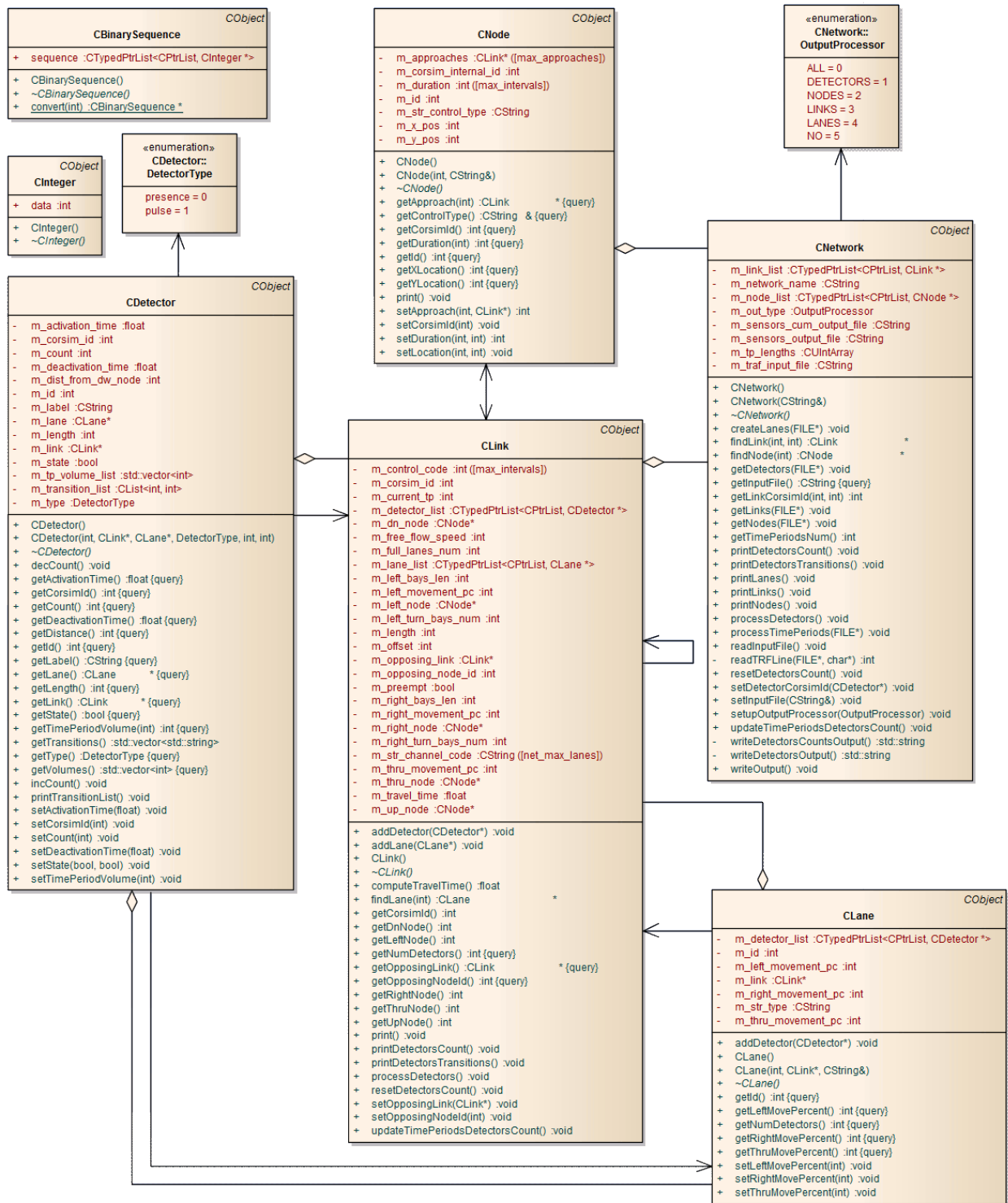


Figura 5.8: Diagramma delle classi di Sensors DLL.

5.3.2 Formato dell'output

Di seguito si presenta un esempio di file CSV di output generato da Sensors DLL. Come anticipato, esso codifica l'onda quadra di ogni

seniore a spira magnetica posto sulla rete stradale.

```
time,tp,D131,D231,D232,D211,D212,D213
0.10000,1,0,0,0,0,0,0
0.20000,1,0,0,1,0,0,0
0.30000,1,0,0,1,0,0,0
0.40000,1,0,0,0,0,0,0
0.50000,1,0,0,0,0,1,0
0.60000,1,0,0,0,0,1,0
0.70000,1,0,0,0,0,0,0
0.80000,1,0,0,0,0,0,0
0.90000,1,0,0,0,0,0,0
1.00000,1,0,1,0,0,0,0
1.10000,2,0,1,0,0,0,1
1.20000,2,0,1,0,0,0,1
1.30000,2,0,1,0,0,0,0
1.40000,2,0,0,0,0,0,0
1.50000,2,0,0,1,0,0,0
1.60000,2,0,0,1,0,0,0
1.70000,2,0,0,0,0,0,0
1.80000,2,1,0,0,0,0,0
1.90000,2,1,0,0,0,0,0
2.00000,2,1,0,0,0,0,0
```

Sorgente 5.7: Formato di output di Sensors DLL

La tabella 5.3 chiarisce il significato di ogni colonna dei dati tabulari restituiti da Sensors DLL.

Nome colonna	Descrizione
time	Tempo di simulazione a cui è stato effettuato il monitoraggio dei sensori.
tp	Indice del corrente periodo temporale (i. e., <i>time period</i>).
identificatore sensore	1 in caso di presenza di un veicolo sul rispettivo sensore, 0 altrimenti.

Tabella 5.3: Descrizione della semantica dei file CSV generati da Sensors DLL.

Si osservi che, il fatto che il sensore D232, in corrispondenza dell'istante 1.6 (5^a colonna) abbia valore 1 non indica che il veicolo sia stato rilevato esattamente in tale istante, bensì ciò indica che durante l'intervallo temporale [1.5, 1.6] tale sensore è stato attivato dal passaggio di un veicolo.

5.4 APPLICATIVI DI SUPPORTO

Al fine di automatizzare e facilitare la creazione di dataset relativi al traffico tramite la RTE trattata nel corrente capitolo, si è sviluppato un insieme di strumenti dediti alla manipolazione dei file di output di Sensors DLL.

Di seguito si elencano le operazioni di manipolazione che tali strumenti supportano:

1. sostituzione (e eventualmente rimozione) della colonna relativa ai periodi temporali con una nuova colonna che rappresenti la classe di un insieme di osservazioni; tale colonna è automaticamente generata in base a un sistema di regole (e.g., *matching* tra periodo temporale e classe).
2. partizionamento del file di output di Sensors DLL in più file in base a vincoli temporali (e.g., divisione del file in blocchi di 60 secondi ciascuno)
3. ottimizzazione del file tramite rimozione delle linee duplicate (i.e., linee in cui non è avvenuto alcun cambiamento di stato dei sensori).

Lo scopo finale di questo capitolo è presentare i risultati ottenuti dalla sperimentazione del processo di *classificazione supervisionata* utilizzando diversi modelli di classificatore CTBN (CTBNC) (riguardo tali argomenti si rimanda ai capitoli 2 e 3 a pagina 21 e a pagina 33).

Tuttavia, prima di illustrare e comparare i succitati risultati, si presentano i modelli TSIS da cui sono stati generati i dataset oggetto della sperimentazione. La generazione di dataset inerenti il traffico automobilistico è stata effettuata, come detto, tramite un software commerciale apposito, TSIS, e una estensione a tempo d'esecuzione, Sensors DLL, sviluppata a tale fine. Per maggiori dettagli riguardanti questi strumenti si rimanda al capitolo 5 a pagina 48.

Perciò, questo capitolo è così articolato: una sezione per ognuno dei 2 dataset generati, e una sezione relativa alla succitata sperimentazione. Le sezioni relative ai dataset sono suddivise in due sottosezioni: la prima finalizzata alla descrizione del modello TSIS da cui è stato generato il dataset successivamente sottoposto a sperimentazione e la seconda finalizzata alla presentazione delle varie configurazioni del dataset generato.

Al fine di valutare l'accuratezza dei metodi predittivi sviluppati, la sperimentazione è stata eseguita utilizzando la tecnica chiamata *cross-validazione*. Tale metodologia di validazione è infatti pensata per accertare la capacità di generalizzazione di un modello statistico di tipo predittivo su un insieme di dati non conosciuto. Nello specifico, le sperimentazioni dei metodi predittivi valutati sono state condotte tramite una *k-fold cross-validazione* con $k = 10$.

6.1 DATASET #1

Lo scopo di questa sezione è presentare le varie configurazioni del dataset #1 create.

6.1.1 Modello TSIS

Al fine di introdurre il dataset #1 è necessario presentare la rete stradale TSIS e il relativo modello di simulazione CORSIM da cui esso è stato generato.

La figura 6.1 nella pagina seguente mostra la rete stradale che compone il modello TSIS oggetto di questa sottosezione.

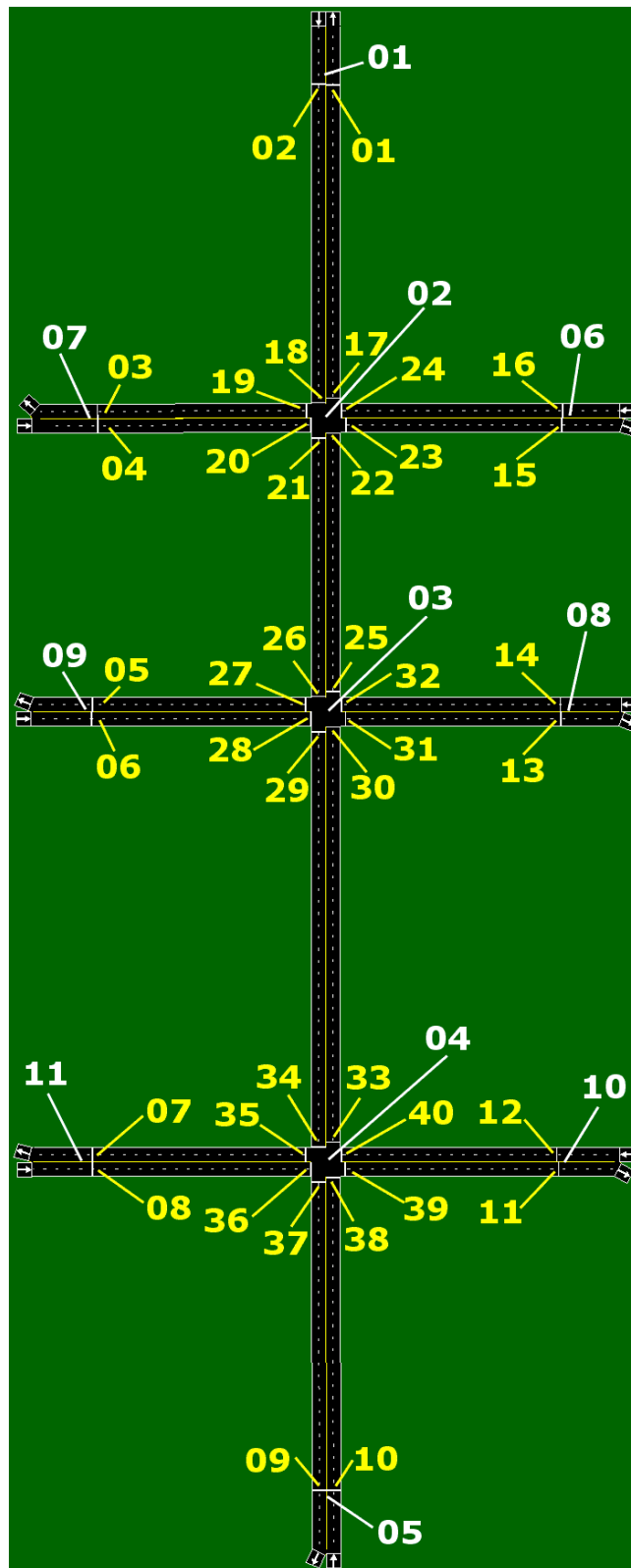


Figura 6.1: Visualizzazione del file TNO rappresentante la rete stradale da cui viene generato il dataset #1. I sensori sono etichettati tramite degli indici numerici di colore giallo mentre i nodi sono etichettati tramite degli indici numerici di colore bianco.

Tale figura evidenzia:

- la disposizione dei sensori (indicati tramite degli indici numerici di colore giallo) utilizzati per il monitoraggio del passaggio dei veicoli
- le strade, tutte composte da 2 corsie, e i nodi (indicati tramite degli indici numerici di colore bianco) da cui è composta la rete stradale
- i sensi di marcia delle corsie

Di seguito si riportano le ulteriori caratteristiche di tale rete stradale:

- la larghezza delle corsie è 3,65 m
- la lunghezza della strada principale (i.e., da nodo 01 a nodo 05) è 780 m mentre la lunghezza delle strade ad essa perpendicolari è pari a 300 m
- la dimensione dei sensori è 0,30 m, da cui consegue che la dimensione della zona sensibile (in tutte le direzioni) è 2,13 m
- la distanza dei sensori dal nodo più vicino è 6,5 m

Le intersezioni principali della rete stradale (i.e., nodi con indici 02, 03 e 04 nella figura 6.1 nella pagina precedente) sono controllate da semafori il cui ciclo di controllo dura 100 secondi. La figura 6.2 illustra la configurazione del piano semaforico per ogni intersezione controllata durante tutto il tempo di simulazione.

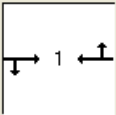
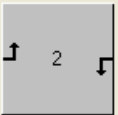
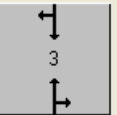

Phase Movements:				
Green Time:	30	20	20	10
Yellow Time:	1	1	1	1
All Red Time:	4	4	4	4

Figura 6.2: Piano semaforico predefinito delle intersezioni 02, 03 e 04 della rete stradale TSIS relativa al dataset #1.

Come anticipato, ogni strada che compone la rete stradale contiene, nei pressi dei nodi (sia che essi siano delle intersezioni, sia che essi siano dei nodi di ingresso o di transito), dei sensori finalizzati al rilevamento e tracciamento del passaggio dei veicoli: essi sono indicati nella figura 6.1 nella pagina precedente da indici numerici di colore giallo. La tabella 6.1 nella pagina seguente elenca il nome reale di tali sensori, utilizzato per la generazione del relativo dataset (a tal riguardo, un esempio di dataset è stata riportato nella sottosezione 5.3.2 a pagina 69).

#	Sensore	#	Sensore	#	Sensore	#	Sensore
01	D212	11	D4102	21	D231	31	D381
02	D121	12	D1041	22	D322	32	D832
03	D272	13	D382	23	D261	33	D431
04	D721	14	D831	24	D622	34	D342
05	D392	15	D262	25	D321	35	D4111
06	D931	16	D621	26	D232	36	D1142
07	D4412	17	D211	27	D391	37	D451
08	D1141	18	D122	28	D932	38	D542
09	D452	19	D271	29	D341	39	D4101
10	D541	20	D722	30	D432	40	D1042

Tabella 6.1: Corrispondenza fra gli identificatori dei sensori del dataset #1 e l'indice con cui essi sono indicati nella figura 6.1.

Il modello di simulazione CORSIM creato per questa rete stradale prevede un tempo di simulazione totale pari a 24 ore, suddiviso in 16 *time period*, ognuno dei quali composto da *time interval* della durata di 1 secondo: la tabella 6.2 illustra la durata (in ore) di ognuno di essi e introduce la corrispondenza fra essi e la fase della giornata, rappresentata come la variabile classe nel relativo dataset.

Fascia oraria	Time Period	Fase	Classe
07:00 - 08:00	1	mattino	1
08:00 - 09:00	2	mattino	1
09:00 - 10:00	3	mattino	1
10:00 - 12:00	4	giorno	2
12:00 - 14:00	5	giorno	2
14:00 - 16:00	6	giorno	2
16:00 - 17:00	7	giorno	2
17:00 - 18:00	8	pomeriggio	3
18:00 - 19:00	9	pomeriggio	3
19:00 - 20:00	10	pomeriggio	3
20:00 - 22:00	11	sera	4
22:00 - 24:00	12	sera	4
00:00 - 02:00	13	notte	5
02:00 - 04:00	14	notte	5
04:00 - 05:00	11	notte	5
05:00 - 07:00	16	alba	6

Tabella 6.2: Caratterizzazione dei periodi temporali (i.e., *time period*) del modello TSIS relativo al dataset #1.

Ogni fase della giornata (i.e., classe) è caratterizzata da un andamento diverso del traffico:

- al mattino traffico intenso su tutta la rete stradale e code in

direzione del nodo 05

- durante il giorno traffico moderato su tutta la rete e code residue verso il nodo 05 nella prima parte di tale fase
- nel pomeriggio traffico intenso su tutta la rete stradale e code in direzione del nodo 01
- durante la sera traffico moderato su tutta la rete stradale e code residue verso il nodo 01 nella prima parte di tale fase
- durante la notte pochi veicoli in tutte le direzioni
- all'alba aumento dei veicoli diretti verso il nodo 05

L'andamento del traffico è modellato in CORSIM variando i flussi di ingresso dei veicoli e le percentuali di svolta nelle varie intersezioni al variare dei periodi temporali o, in generale, delle fasi di simulazione previste. La tabella 6.3 illustra i flussi di ingresso nella rete stradale, espressi come numero di veicoli orari (i.e., vph), per ogni fase della simulazione. Si osservi che l'immissione di veicoli sulla rete stradale avviene seguendo una distribuzione di probabilità di Erlang con parametro *shape* pari a 2.

Fase	01 → 05 vph	05 → 01 vph	Strade secondarie vph
mattino	600	200	100
giorno	300	300	50
pomeriggio	200	600	100
sera	100	200	50
notte	50	50	25
alba	200	100	50

Tabella 6.3: Numero di veicoli orari immessi nella rete stradale del dataset dataset #1 al variare della fase di simulazione.

La tabella 6.4 nella pagina seguente illustra le percentuali di svolta al variare della fase di simulazione. L'intestazione di tale tabella indica i nodi di provenienza dei veicoli, il senso di marcia e le direzioni previste dall'intersezione che segue il nodo di partenza. Le celle invece indicano la distribuzione delle svolte effettuate dai veicoli nelle intersezioni.

Si consideri, ad esempio, la prima cella della tabella 6.4: l'80% dei veicoli provenienti dai nodi 01, 02 e 03, giunti alle intersezioni che seguono tali nodi (i.e., nodi 02, 03 e 04, rispettivamente; si veda a tal riguardo la figura 6.1 a pagina 73), proseguono senza svoltare; il 10% dei veicoli provenienti dai nodi 01, 02 e 03, invece, svolterà verso destra; idem verso sinistra.

Nodi	01 - 02 - 03			03 - 04 - 05			06 - 08 - 10			07 - 09 - 11		
	→			←			→			←		
	↔	↔	↓	↔	↔	↓	↘	↔	↓	↖	↔	↓
	%			%			%			%		
mattino	10	10	80	20	20	60	80	10	10	10	80	10
giorno	20	20	60	20	20	60	60	20	20	20	60	20
pomeriggio	20	20	60	10	10	80	10	80	10	80	10	10
sera	40	40	20	10	10	80	30	40	30	40	30	30
notte	33	33	33	33	33	33	33	33	33	33	33	33
alba	10	10	80	40	40	20	40	30	30	30	40	30

Tabella 6.4: Percentuali di svolta dei veicoli per ogni intersezione della rete stradale del dataset #1, al variare della fase di simulazione.

6.1.2 Configurazioni del dataset

Come detto nella sottosezione 5.3.1 a pagina 66, eseguendo il modello di simulazione CORSIM descritto in questa sezione con Sensors DLL si ottiene un file di output (si veda il listato d'esempio 5.7 a pagina 70) che, adeguatamente processato, compone un dataset sottoponibile agli algoritmi di apprendimento e classificazione presentati nei capitoli 2 e 3.

Di seguito si descrivono le caratteristiche di tale dataset comuni a tutte le sue varianti generate:

- tempo totale pari a 86400 secondi (i. e., 24 ore)
- la colonna `tp` è stata rimossa e sostituita con la colonna `class` in base alla corrispondenza riportata nella tabella 6.2 a pagina 75
- l'intestazione di ogni elemento è:
`time, class, D122, D121, D341, D342, D722, D721, D1142, D1141, D231, D232, D541, D542, D4112, D4111, D431, D432, D4101, D4102, D451, D452, D832, D831, D261, D262, D321, D322, D392, D391, D211, D212, D272, D271, D1042, D1041, D622, D621, D932, D931, D381, D382.`
- ogni elemento del dataset (i. e., file) è stato ottimizzato, cioè sottoposto all'eliminazione delle righe in cui non si verifica alcuna transizione di stato dei sensori

Per il dataset in questione si sono create due sue differenti varianti operando differenti divisioni temporali del file sorgente:

1. suddivisione in 864 elementi, ognuno dei quali include al più 100 secondi di traiettoria di ogni sensore della succitata rete stradale
2. suddivisione in 288 elementi, ognuno dei quali include al più 5 minuti di traiettorie di ogni sensore della succitata rete stradale.

Tali configurazioni sono chiamate, rispettivamente, dataset #1.100 e dataset #1.300.

6.2 DATASET #2

Questa sezione presenta il dataset #2. A tale scopo, nella sottosezione che segue, si descrive la rete stradale TSIS e il relativo modello di simulazione CORSIM da cui tale dataset è stato generato.

6.2.1 Modello TSIS

Il modello TSIS in questione, a differenza di quello presentato nella sottosezione 6.1.1, riproduce una rete stradale reale. Nello specifico esso riproduce la rete stradale circostante *Viale Cesare Battisti* (20900 Monza, MB - Italia). Disponendo di una serie di informazioni reali, tra cui i dati dei sensori posti su tale rete stradale, si sono potute inferire le caratteristiche della circolazione dei veicoli (e.g., flusso di veicoli in ingresso) e quindi riprodurre fedelmente l'andamento del traffico durante tutti i giorni della settimana tramite degli adeguati modelli di simulazione CORSIM. Nello specifico si sono realizzati 3 modelli CORSIM: essi rappresentano la stessa rete stradale ma diversi modelli di traffico, uno per i giorni lavorativi (dal lunedì al venerdì), uno per il sabato e uno per la domenica.

La figura 6.3 nella pagina successiva illustra la rete stradale oggetto di studio, evidenziando e identificando i nodi e i collegamenti stradali utilizzati per la sua riproduzione in TSIS.

La tabella 6.5 illustra invece la correlazione tra gli indici dei nodi CORSIM e ciò che essi rappresentano nel modello in questione.

Nodo/i	Descrizione
01	Intersezione <i>Battisti</i> - <i>Rossini</i> - <i>Alighieri</i>
02	Intersezione <i>Battisti</i> - <i>Brianza</i> - <i>Margherita</i>
03	Nodo di ingresso da <i>Viale Brianza</i>
04 - 05 - 06	Intersezione <i>Boccaccio</i> - <i>Margherita</i>
08	Intersezione <i>Battisti</i> - <i>Boito</i> - <i>Tognetti</i>
09	Intersezione <i>Battisti</i> - <i>Donizzetti</i> - <i>Volta</i>
10	Nodo di ingresso da <i>Via Alighieri</i>
11	Nodo di ingresso da <i>Via Rossini</i>
12	Nodo di ingresso da <i>Via Donizzetti</i>
13	Nodo di uscita da <i>Via Volta</i>
14	Nodo di ingresso da <i>Via Tognetti</i>
15	Nodo di ingresso da <i>Via Boito</i>
07 - 16	Diramazione <i>Boito</i> - <i>Battisti</i>
17	Nodo di ingresso da <i>Viale Battisti</i>
18	Nodo di ingresso da <i>Via Boccaccio</i>
19	Nodo di ingresso da <i>Viale Margherita</i>

Tabella 6.5: Caratterizzazione degli identificatori delle intersezioni (o nodi) del dataset #2.



stra *Via Boccaccio*) che svoltano in diagonale a sinistra su *Viale Margherita*.

5096

Posto nei pressi dell'intersezione 02 (i. e., *Battisti - Brianza - Margherita*), rileva i veicoli provenienti dal nodo 03 (i. e., corsia sinistra di *Viale Brianza*) diretti verso il nodo 04 (i. e., *Boccaccio - Margherita*).

5097

Posto nei pressi dell'intersezione 02 (i. e., *Battisti - Brianza - Margherita*), rileva i veicoli su *Viale Margherita* (corsia sinistra) che svoltano a sinistra su *Viale Battisti*.

5098

Posto nei pressi dell'intersezione 02 (i. e., *Battisti - Brianza - Margherita*), rileva i veicoli su *Viale Battisti* (corsia sinistra) che svoltano in diagonale a sinistra su *Viale Brianza*.

5099

Posto nei pressi dell'intersezione 02 (i. e., *Battisti - Brianza - Margherita*), rileva i veicoli su *Viale Battisti* (corsia destra) che svoltano a destra in *Viale Margherita*.

I sensori aggiunti alla rete stradale sono identificati secondo uno schema ben preciso. Il loro identificatore è composto in base al senso di marcia. Nello specifico tale numero è una concatenazione dei seguenti valori: identificatore del nodo di partenza, identificatore del nodo d'arrivo, numero indicante la corsia su cui il sensore è posto (i. e., 0 se è la strada possiede una sola corsia, 1 o 2 se il sensore è posto rispettivamente a destra o a sinistra, 7 o 8 nel caso in cui esso sia posto su ulteriori corsie di destra o sinistra). Ad esempio:

- il sensore identificato da D912 è posto sulla corsia di sinistra della strada che collega l'intersezione "Volta" (i. e., nodo 09) all'intersezione "Dante" (i. e., nodo 01)
- il sensore identificato da D1480 è posto sull'unica corsia di "Via Tognetti" (i. e., nodo 14) nei pressi dell'incrocio con l'intersezione "Boito" (i. e., nodo 08).

La figura 6.5 nella pagina successiva illustra la disposizione di tutti i sensori sulla rete stradale in esame, distinguendo quelli reali tramite un colore diverso dell'indice che li rappresenta.

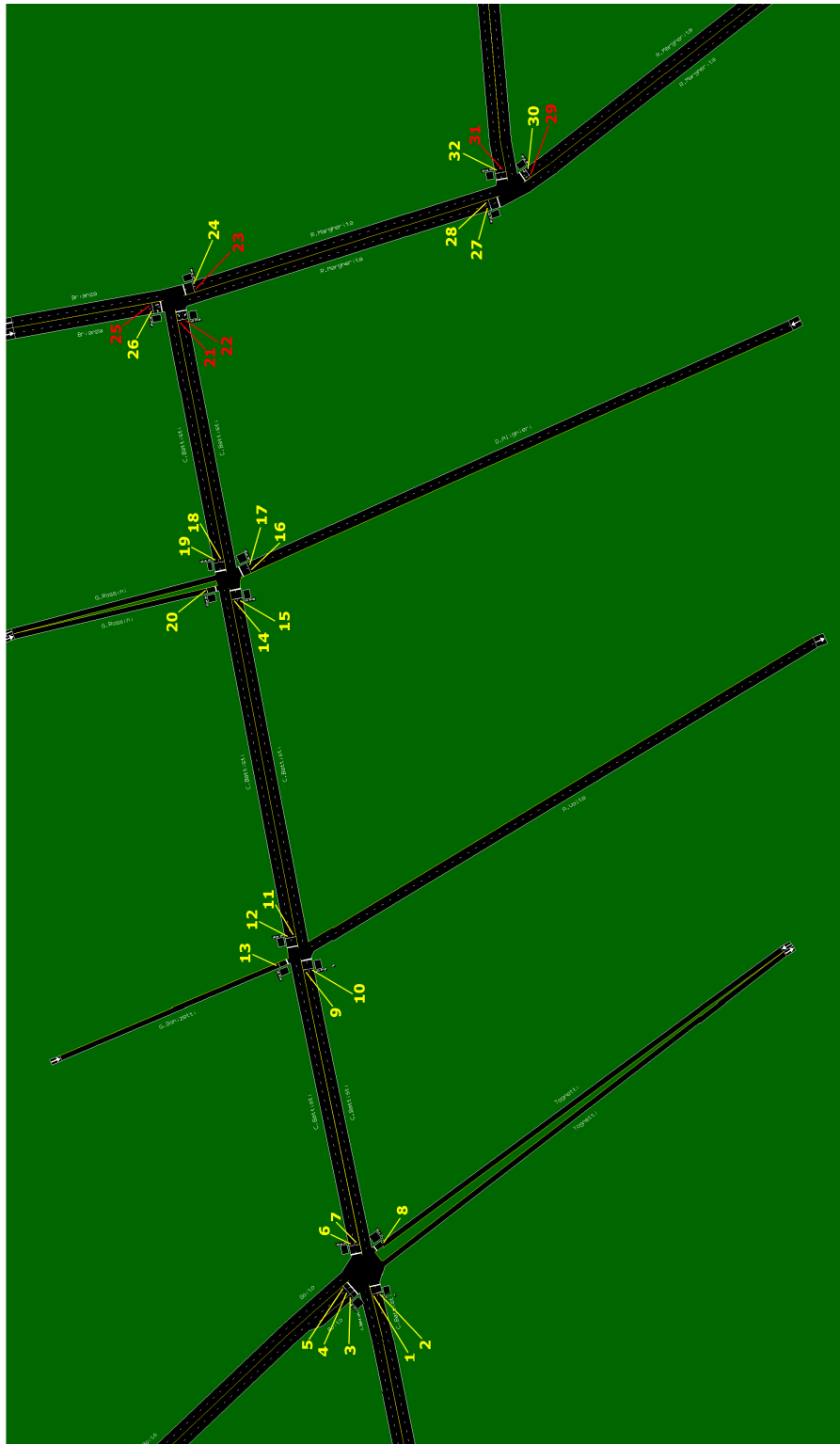


Figura 6.5: Visualizzazione del file TNO rappresentante la rete stradale da cui viene generato il dataset #2.

Nella tabella 6.6 nella pagina seguente, i sensori il cui identificatore è contrassegnato da un asterisco corrispondono ai sensori reali, evidenziati anche nella figura 6.5 tramite un indice numerico di colore

#	Sensore	#	Sensore	#	Sensore	#	Sensore
01	D782	09	D892	17	D1091	25	D5096*
02	D781	10	D891	18	D212	26	D321
03	D1687	11	D192	19	D211	27	D241
04	D1681	12	D191	20	D110	28	D242
05	D1682	13	D1290	21	D5098*	29	D5094*
06	D981	14	D912	22	D5099*	30	D642
07	D982	15	D911	23	D5097*	31	D5095*
08	D1480	16	D1092	24	D421	32	D541

Tabella 6.6: Corrispondenza fra gli identificatori dei sensori del dataset #2 e l'indice con cui essi sono indicati nella figura 6.5 nella pagina precedente.

rosso; i restanti sensori corrispondono a quelli aggiunti per estendere il modello in questione.

Di seguito si riportano le ulteriori caratteristiche della rete stradale oggetto di questa sottosezione:

- la larghezza delle corsie è 3,65 m
- le strade del modello TSIS riproducono esattamente, sia in lunghezza che in forma, la porzione di strada reale che modellano:
 - *Viale Battisti* è lungo 1,33 km
 - *Via Boito* è lunga 308 m
 - *Via Tognetti* è lunga 351 m
 - *Via Donizzetti* è lunga 176 m
 - *Via Volta* è lunga 395 m
 - *Via Rossini* è lunga 141 m
 - *Via Dante* è lunga 422 m
 - *Viale Brianza* è lungo 106 m
 - *Viale Margherita* è lungo 505 m
 - *Via Boccaccio* è lunga 336 m
- la dimensione dei sensori è 0,03 m, da cui consegue che la dimensione della zona sensibile (in tutte le direzioni) è 1,85 m
- la distanza dei sensori dall'intersezione più vicina è 6,5 m
- la velocità media dei veicoli: 67 km/h su *Viale Battisti*, 56 km/h sulle altre strade.

Come anticipato, per questo caso di studio si sono creati 3 modelli di simulazione differenti (corrispondenti quindi a 3 differenti sorgenti TRF), al fine di rappresentare i diversi flussi di traffico durante la settimana.

Il file TRF creato per modellare il traffico sulla rete stradale nei giorni lavorativi prevede un tempo di simulazione totale pari a 24 ore, suddiviso in 12 *time period*, ognuno dei quali composto da *time interval* della durata di 100 secondi: la tabella 6.7 illustra la durata (in ore) di ognuno di essi e introduce la corrispondenza fra essi e la fase della giornata, rappresentata come la variabile classe nel relativo dataset.

Nei modelli di traffico relativi al sabato e alla domenica la corrispondenza tra i *time period*, che rimangono identici a quelli mostrati nella tabella 6.7, e la variabile classe è differente. Ad esempio, il *time period* 1 del modello di traffico relativo al sabato è stato associato alla classe 11 mentre il *time period* 1 del modello di traffico relativo alla domenica è stato associato alla classe 21. Ne deriva che gli identificatori della variabile classe per questi due modelli di traffico sono incrementati di una e due decine rispetto agli identificatori usati per il modello di traffico relativo ai giorni lavorativi.

Fascia oraria	Time Period	Fase	Classe
00:00 - 02:05	1	notte	1
02:05 - 04:10	2	notte	1
04:10 - 06:15	3	alba	2
06:15 - 08:20	4	mattino	3
08:20 - 10:25	5	mattino	3
10:25 - 12:30	6	giorno	4
12:30 - 14:35	7	giorno	4
14:35 - 16:40	8	giorno	4
16:40 - 18:45	9	pomeriggio	5
18:45 - 20:50	10	pomeriggio	5
20:50 - 22:55	11	sera	6
22:55 - 24:00	12	sera	6

Tabella 6.7: Caratterizzazione dei periodi temporali (i.e., *time period*) del modello TSIS relativo al dataset #2 (giorni lavorativi).

Ogni fase della giornata (i.e., classe) è caratterizzata da un andamento diverso del traffico. Di seguito si presenta l'andamento generale del traffico durante i giorni lavorativi:

- al mattino (i.e., periodi temporali 4 e 5) traffico diretto verso il centro di *Monza*, prevalentemente da *Viale Battisti* (i.e., nodo di ingresso 17) e, in misura minore, da *Viale Monza* (i.e., nodo di ingresso 03)
- durante il giorno (i.e., periodi temporali 6, 7 e 8) incremento del traffico verso il centro di *Monza*, soprattutto da *Viale Battisti*, e incremento del traffico fra *Viale Boccaccio* e *Viale Margherita* (i.e., nodi 18 e 19)
- durante il pomeriggio (i.e., periodi temporali 9 e 10) il flusso dei veicoli inizia a invertire la sua direzione verso *Viale Battisti*;

aumento del numero di veicoli diretti verso le strade laterali (e.g., *Via Donizzetti*, *Via Alighieri*, *Via Rossini*)

- alla sera (i.e., periodi temporali 11 e 12) traffico proveniente dal centro di *Monza* diretto verso *Viale Battisti* e, al contempo, diminuzione dei veicoli provenienti da *Viale Brianza*
- durante la notte (i.e., periodi temporali 1 e 2) riduzione del traffico su tutta la rete
- all'alba (i.e., periodo temporale 3) pochi veicoli circolanti sulla rete stradale, prevalentemente su *Viale Battisti* dove si verifica un leggero incremento dei veicoli in ingresso.

Come detto, l'andamento del traffico è modellato variando i flussi di ingresso. La tabella 6.8 illustra il numero di veicoli orari (i.e., vph) in ingresso sulla rete stradale in relazione a ogni periodo temporale del modello di simulazione. Anche per i modelli di simulazione CORSIM relativi a questa rete stradale si è scelto di utilizzare una distribuzione di probabilità di Erlang per governare l'immissione dei veicoli sulla rete stradale.

Time Period	19 vph	18 vph	17 vph	15 vph	14 vph	12 vph	11,10 vph	03 vph
1	150	275	375	5	5	5	5	205
2	25	80	105	5	5	5	5	70
3	96	70	325	5	5	5	5	113
4	350	450	1950	20	50	20	20	1080
5	500	600	1855	20	20	20	20	1325
6	750	500	2100	20	20	20	20	1022
7	600	400	1950	20	20	20	20	1000
8	700	600	2390	20	20	20	20	1050
9	700	510	2410	20	20	20	20	1010
10	700	250	2275	20	12	20	20	980
11	370	190	1565	10	10	10	10	575
12	400	75	975	8	8	10	7	400

Tabella 6.8: Numero di veicoli orari immessi nella rete stradale da ogni nodo di ingresso (colonne) del dataset #2 (giorni lavorativi) al variare del periodo temporale sulle righe.

Si osservi, inoltre, che al fine di definire il succitato andamento del traffico è stata attuata un'adeguata modellazione delle percentuali di svolta in ogni intersezione. Tuttavia, a causa della complessità del modello, non è possibile riportare le variazioni delle percentuali di svolta di ogni intersezione per ogni periodo temporale.

Per chiarezza si sottolinea infine che i profili di traffico del sabato e della domenica sono stati descritti con flussi di ingresso e percentuali di svolta di diversa entità rispetto a quelli appena descritti.

6.2.2 Configurazioni del dataset

Eseguendo con Sensors DLL uno qualsiasi dei modelli di simulazione (i. e., giorni lavorativi, sabato o domenica) relativi alla rete stradale presentata, si ottiene il relativo file di output che, processato tramite gli strumenti di supporto precedentemente introdotti (si veda la sezione 5.4 a pagina 71), compone un dataset utile alla classificazione dei profili di traffico.

Affinché i dataset in questione siano perfettamente riproducibili si sono utilizzati dei file RNS⁴¹, semplici file di testo, opportunamente formattati, da cui CORSIM legge il numero di esecuzioni da effettuare e i semi numerici⁴² relativi ad alcuni aspetti stocastici della simulazione (e. g., incidenti, tipologia di guida dei conducenti).

Di seguito si riporta il file RNS relativo al modello di traffico per i giorni lavorativi della rete stradale illustrata nella precedente sezione.

5			
	32317	22061	28769
	19372	19528	8812
	10695	338	20744
	16724	27350	29969
	8267	26189	3302

Sorgente 6.1: File RNS che configura il numero di esecuzioni (i. e., 5) e i semi numerici del modello di simulazione CORSIM relativo alla rete stradale del dataset #2.

Tralasciando i profili di traffico relativi al sabato e alla domenica, si illustrano ora le varianti generate con Sensors DLL dalla simulazione CORSIM del profilo di traffico relativo ai giorni lavorativi. Dal modello TSIS illustrato si sono generate 4 diverse varianti del dataset #2 combinando la presenza o l'assenza dei sensori aggiuntivi con diversi tagli temporali dell'intero insieme di onde quadre dei sensori.

Tutte le varianti del dataset #2 generate sono accomunate dalle seguenti caratteristiche:

- tempo totale pari a 86400 secondi (i. e., 24 ore)
- la colonna `tp` è stata rimossa e sostituita con la colonna `class` in base alla corrispondenza riportata nella tabella 6.7 a pagina 84 (gli identificatori delle classi variano, come detto, per i profili di traffico relativi al sabato e alla domenica)

⁴¹ CORSIM supporta l'utilizzo di file Random Number Seed (RNS) per la configurazione di alcuni aspetti della simulazione: numero di esecuzioni da effettuare e semi numerici da utilizzare per la generazione stocastica di eventi.

⁴² Il *seed* è un numero utilizzato per inizializzare un generatore di numeri pseudo-casuali. Condividendo tale numero e utilizzando lo stesso generatore di numeri pseudo-casuali utilizzato da CORSIM è possibile riprodurre la stessa identica simulazione su altri calcolatori.

- ogni elemento è stato ottimizzato, cioè sottoposto all'eliminazione delle righe in cui non si verifica alcuna transizione di stato dei sensori

Si osservi come tali caratteristiche siano comunque indipendenti dal profilo di traffico simulato. Le differenze tra i 3 diversi modelli di traffico sviluppati si sostanziano nel diverso andamento dei flussi di traffico sulla rete stradale, il che porta a diversi dati di rilevazione da parte dei sensori; nel numero di esecuzioni della simulazione (i.e., 5 per i giorni lavorativi e 1 per il sabato e la domenica) e nei valori assunti dalla colonna relativa alla classe di traffico (e.g., classe 21 piuttosto che classe 1).

Si descrivono infine le varianti del dataset #2 sottoposte a sperimentazione.

DATASET #2.B.100

Dataset composto da 864 elementi (i.e., file) ognuno dei quali rappresenta al più 100 secondi di traiettoria dei sensori. Contiene solamente le onde quadre relative ai 6 sensori reali (i.e., D5094, D5095, D5096, D5097, D5098 e D5099).

DATASET #2.B.300

Come il precedente, tale dataset contiene solo le onde quadre relative ai sensori reali. È composto da 288 elementi ognuno dei quali rappresenta al più 5 minuti di traiettorie dei succitati sensori.

DATASET #2.E.100

Dataset composto da 864 elementi ognuno dei quali rappresenta al più 100 secondi di traiettorie dei sensori. Poiché esso considera tutti i sensori del modello TSIS illustrato in questa sezione, l'intestazione di ognuno dei suoi elementi è la seguente:

```
time, class, D5096, D341, D5098, D5099, D211, D212,
D241, D242, D5097, D421, D5095, D541, D5094, D641,
D891, D892, D981, D982, D911, D912, D191, D192, D1290,
D1682, D1681, D1687, D1110, D1091, D1092, D1480, D781,
D782.
```

DATASET #2.E.300

Dataset composto da 288 elementi ciascuno dei quali rappresenta al più 5 minuti di traiettorie dei sensori. Come il precedente, anche questo dataset, è stato generato dal modello TSIS provvisto dei sensori aggiuntivi.

6.3 SPERIMENTAZIONE

In questa sezione si presentano i risultati ottenuti dalla sperimentazione della classe dei classificatori CTBN (CTBNC) descritta in questo lavoro di tesi.

Prima di presentare tali risultati si introduce la metodologia di sperimentazione utilizzata sui succitati 6 dataset.

Come anticipato, dato un modello di classificatore CTBN, si è svolta la stessa *k-fold cross-validazione* con $k = 10$ per ognuno dei dataset generati. Si è cioè suddiviso ognuno dei succitati dataset in 10 campioni di uguale numerosità utilizzandone uno come *validation set* (i. e., *test set*) e i restanti come *training set*. Quindi tale processo è stato ripetuto 10 volte, affinché ognuno dei 10 campioni venisse utilizzato come *test set* esattamente una sola volta.

Il fine della *cross-validazione* è quello di valutare la capacità di apprendimento e generalizzazione di un modello predittivo (si veda Hastie *et al.*, 2001, capitolo 7), qual è il classificatore CTBN. La *cross-validazione* è considerata una delle tecniche possibili utilizzabili al fine di rilevare e evitare il problema dell'*overfitting*⁴³. A tale scopo, i risultati di ognuno dei 10 processi di classificazione possono essere mediati (o combinati in modi alternativi) al fine di produrre singoli valori che stimino l'efficacia⁴⁴ del modello predittivo utilizzato. In questa sperimentazione, avendo affrontato un problema di *classificazione multi-classe*, si sono mediati i risultati relativi a ogni *fold*⁴⁵ utilizzando i cosiddetti approcci *micro-average* e *macro-average*.

La differenza tra queste due tecniche di sintesi dei risultati consiste nel fatto che mentre l'approccio *macro-average* è una media (non pesata) delle metriche relative ad ogni classe, l'approccio *micro-average*, al contrario, consiste in una media delle metriche attuata sul totale delle istanze sottoposte a classificazione. Da ciò consegue che la tecnica *micro-average* tende a conferire maggiore peso alle classi composte da più istanze e costituisce, in caso di classi popolose nel *test set* una reale misura dell'efficacia del classificatore. Invece, per sintetizzare l'efficacia del classificatore su classi composte da poche istanze è preferibile considerare le indicazioni di efficacia date dall'approccio *macro-average* (si veda Manning *et al.*, 2008, sezione 13.6).

Si precisa che i risultati presentati di seguito sono quelli ottenuti dall'approccio *micro-average* poiché le classi dei dataset generati sono

⁴³ Il problema dell'*overfitting* è un problema frequente nei campi di ricerca che utilizzano modelli e tecniche statistiche (e. g., apprendimento automatico, data mining). Tale problema si verifica quando un modello statistico si adatta eccessivamente al *training set* perdendo così la capacità di riconoscere (i. e., classificare, nel caso di modelli predittivi) istanze del problema non conosciute; non acquisendo cioè una adeguata capacità di generalizzazione.

⁴⁴ L'efficacia di un classificatore è un termine generico che indica un insieme di misure (e. g., *precision*, *recall*, *F-measure* e *accuracy*) della qualità di un classificatore.

⁴⁵ Un *fold* indica una configurazione, comprensiva di *training set* e *test set*, delle k utilizzate durante una *k-fold cross-validazione*.

tutte composte da numeri di istanze comparabili e nessuna di esse domina le altre in termini quantitativi.

Inoltre, si specifica che le procedure di apprendimento dei parametri di ogni classificatore CTBN (CTBNC) appreso sono state effettuate utilizzando la *stima bayesiana* (si veda l'intera sottosezione 1.4.3 a pagina 16) con conteggi immaginari $\alpha_{xx'}$ e τ_x impostati rispettivamente ai valori 1 e 0.005.

La procedura di sperimentazione descritta è stata attuata per i 4 differenti modelli di classificatori CTBN elencati di seguito:

- classificatore CTNB (CTNBC)
- classificatore CTBN appreso dal *training set* (tramite la procedura di apprendimento strutturale presentata nel capitolo 3 a pagina 33) fissando il numero massimo di genitori a 2
- classificatore CTBN appreso dal *training set* fissando il numero massimo di genitori a 3
- classificatore CTBN appreso dal *training set* fissando il numero massimo di genitori a 4.

L'obiettivo delle prossime sezioni è quindi quello di comparare l'efficacia di tali classificatori.

6.3.1 Metriche di valutazione

Prima di presentare i risultati relativi ai succitati classificatori si illustra la metodologia con cui si sono state calcolate le metriche di valutazione per ognuna delle 6 sperimentazioni effettuate.

Come anticipato, dato uno qualunque dei dataset generati si è configurata per esso una sperimentazione *k-fold cross-validazione* con $k = 10$.

Ottenuti i risultati relativi ai processi di classificazione, per ognuno dei 10 *fold* si è effettuato il processo di *cross-tabulazione*⁴⁶ tra le classi reali delle istanze e le classi ad esse assegnate dal processo di *classificazione supervisionata*. Calcolate così le *matrici di confusione*, si è proceduto alla computazione delle metriche di valutazione per ogni *fold* e successivamente all'effettuazione delle medie *macro-average* e *micro-average*.

La tabella 6.9 nella pagina successiva riproduce un esempio di *matrice di confusione* calcolata per ogni singolo processo di classificazione. Essa rappresenta le classi reali delle istanze sulle colonne e le classi inferite sulle righe. Ogni cella (i, j) contiene il numero di istanze della classe j classificate come appartenenti alla classe i . Ne deriva

⁴⁶ Il processo di cross-tabulazione consiste in un processo statistico il cui scopo è riassumere le interrelazioni che intercorrono fra due variabili categoriche dei dati. L'output di questo processo corrisponde solitamente a una *matrice di confusione*.

perciò che una classificazione perfetta (il ché, si noti, non corrisponde a aver appreso un classificatore perfetto) corrisponde ad una matrice diagonale⁴⁷.

	Classe 1	Classe 2	Classe 3	Classe 4	Classe 5	Classe 6
Classe 1	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)
Classe 2	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)
Classe 3	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)
Classe 4	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)
Classe 5	(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, 6)
Classe 6	(6, 1)	(6, 2)	(6, 3)	(6, 4)	(6, 5)	(6, 6)

Tabella 6.9: Esempio di *matrice di confusione* specifico del problema di classificazione multi-classe affrontato.

A partire dalla tabella 6.9, per ognuna delle classi, è possibile calcolare una ulteriore *matrice di confusione* che riassume la classificazione delle istanze appartenenti ad una singola classe piuttosto che alle altre (approccio detto “*one versus all*”). La tabella 6.10 di seguito riporta un esempio di tale matrice. Fissata una classe, tale tabella, tramite alcune metriche di base che si descrivono di seguito, mette in relazione le classi reali delle istanze con le rispettive classi inferite.

Inferite ↓ / Reali →	Positivi	Negativi	
Positivi	TP	FP	PPV
Negativi	FN	TN	NPV
	TPR	TNR	

Tabella 6.10: Esempio di matrice di confusione relativa al processo di classificazione di una singola classe.

Si osservi che la computazione di tali matrici di confusione è un passo necessario al calcolo delle medie *macro-average*, ottenute appunto dalle metriche relative alle singole classi. Invece, come detto, per il calcolo delle medie *micro-average* si procede sul totale delle istanze del *test set*.

Di seguito si elencano e presentano le metriche di valutazione calcolate.

TRUE POSITIVE

Il valore della metrica True Positive (TP) corrisponde al numero di istanze classificate correttamente.

TRUE NEGATIVE

Il valore della metrica True Negative (TN) corrisponde al numero di istanze non appartenenti alla classe corrente classificate correttamente (e.g., numero di istanze non appartenenti alla

⁴⁷ Una matrice diagonale è una matrice quadrata in cui solamente i valori della diagonale principale sono diversi da 0.

classe 1 che sono state classificate come appartenenti a un'altra classe).

FALSE POSITIVE

Il valore della metrica False Positive (FP) corrisponde al numero di istanze non classificate correttamente.

FALSE NEGATIVE

Il valore della metrica False Negative (FN) corrisponde al numero di istanze non appartenenti alla classe corrente ma classificate come tali (e.g., numero di istanze non appartenenti alla classe 1 ma classificate come tali).

TRUE POSITIVE RATE

Il True Positive Rate (TPR), detto anche *recall* o *sensitivity*, esprime l'abilità del modello nel classificare le istanze in modo corretto. È così calcolato:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = 1 - \text{FNR}.$$

FALSE NEGATIVE RATE

Il False Negative Rate (FNR) è l'inverso della metrica TPR. È quindi calcolato nel seguente modo:

$$\text{FNR} = \frac{\text{FN}}{\text{TP} + \text{FN}} = 1 - \text{TPR}.$$

TRUE NEGATIVE RATE

Il True Negative Rate (TNR), detto anche *specificity*, misura l'abilità del modello nel non sbagliare classificazione, cioè nell'identificare le istanze da non classificare come appartenenti alla classe in esame. È così calcolato:

$$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}} = 1 - \text{FPR}.$$

FALSE POSITIVE RATE

Il False Positive Rate (FPR), detto anche *false alarm rate* o *fall-out*, è l'inverso della metrica TNR. È calcolato nel seguente modo:

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}} = 1 - \text{TNR}.$$

POSITIVE PREDICTIVE VALUE

Il Positive Predictive Value (PPV), detto anche *precision*, rappresenta la proporzione delle istanze classificate correttamente. Esso è calcolato come segue:

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}} = 1 - \text{FDR}.$$

NEGATIVE PREDICTIVE VALUE

Il Negative Predictive Value (NPV) è una metrica che rappresenta la proporzione delle istanze correttamente classificate come non appartenenti a quella in esame. È calcolato nel seguente modo:

$$\text{NPV} = \frac{\text{TN}}{\text{TN} + \text{FN}}.$$

F-MEASURE

Il punteggio F-measure (F_1) è una misura dell'accuratezza che un modello possiede. Nello specifico esso consiste nella media armonica delle metriche *precision* e *recall*:

$$F_1 = 2 \cdot \frac{\text{PPV} \cdot \text{TPR}}{\text{PPV} + \text{TPR}}.$$

ACCURATEZZA

L'accuratezza di un modello (i. e., *accuracy*), intesa come la sua capacità di classificare correttamente tutte le istanze (che appartengano o meno alla classe in esame), è computata nel seguente modo:

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}.$$

Per la computazione delle metriche sono stati usati intervalli di confidenza al 90%.

A titolo informativo si comunica che l'implementazione di quanto descritto in questa sottosezione è stata attuata con l'ausilio di una libreria R, il pacchetto *caret* (Kuhn *et al.*, 2013).

6.3.2 Accuratezza

In questa sottosezione si presentano i risultati dei processi di *classificazione supervisionata* dei 6 dataset effettuati per ognuno dei 4 modelli di classificatori presi in considerazione. Per ogni dataset sottoposto a sperimentazione si illustra e commenta il valore della metrica *accuracy* relativo ad ognuno dei classificatori utilizzati. Si rimarca che la metrica *accuracy* è stata calcolata tramite approccio *micro-average* su tutti i *fold* di ogni sperimentazione.

	CTNBC	CTBNC2	CTBNC3	CTBNC4
Dataset #1.100	0.9108	0.9236	0.9259	0.9259
Dataset #1.300	0.9548	0.9513	0.9513	0.9513
Dataset #2.B.100	0.6254	0.6250	0.6233	0.6206
Dataset #2.B.300	0.7159	0.7013	0.7013	0.7020
Dataset #2.E.100	0.7502	0.7270	0.7307	0.7231
Dataset #2.E.300	0.8208	0.7861	0.7881	0.7791

Tabella 6.11: Comparazione del valore di *accuracy* ottenuto dal classificatore CTNB e dai classificatori CTBN (appresi con numero massimo di genitori variabile da 2 a 4) su ognuno dei 6 dataset generati. In rosso i valori maggiori per ogni dataset.

Seppur la tabella 6.11 evidenzia come il classificatore CTNB (CTNBC) risulti essere, per 5 dataset su 6, maggiormente accurato rispetto ai classificatori CTBN appresi tramite apprendimento strutturale, è necessario osservare come tale vantaggio corrisponda in realtà a un'accuratezza superiore a quella degli altri modelli solo alla seconda o addirittura terza cifra decimale. Tale considerazione vale anche per il dataset #1.100 laddove sono i classificatori CTBN con k (i.e., massimo numero di genitori) pari a 3 e 4 a riportare un leggero incremento della metrica *accuracy*.

Perciò, non essendoci un classificatore che sia apprezzabilmente più accurato degli altri, possiamo asserire che i 4 classificatori utilizzati ottengono pressappoco gli stessi risultati in termini di accuratezza della classificazione.

Tuttavia, osservando la figura 6.6 nella pagina successiva è possibile dedurre una serie di considerazioni ulteriori.

Infatti, da tale figura è possibile notare come, per ogni dataset, all'aumentare dell'intervallo temporale rappresentato dalle istanze (i.e., 100 secondi contro 300), aumenti anche il livello di accuratezza della classificazione per tutti i classificatori. Tale comportamento dei classificatori è giustificabile: la suddivisione di un dataset in istanze che rappresentano un determinato intervallo temporale di dati comporta necessariamente la perdita delle transizioni di stato a cavallo fra le istanze. Ciò è correlato al comportamento descritto: i dataset le cui istanze rappresentano al più 300 secondi di dati, essendo queste più lunghe, hanno conservato un maggior numero di transizioni di stato. Se ne deduce perciò che l'incremento di accuratezza del processo di classificazione supervisionata per i dataset da 300 secondi è correlato a un aspetto quantitativo dell'informazione contenuta da tali dataset. Più semplicemente possiamo affermare che l'accuratezza dei classificatori è proporzionale all'aumento dell'intervallo temporale rappresentato dalle istanze dei dataset.

L'osservazione appena riportata è facilmente riscontrabile soprattutto osservando le due varianti del dataset #2.B nella figura 6.6: il

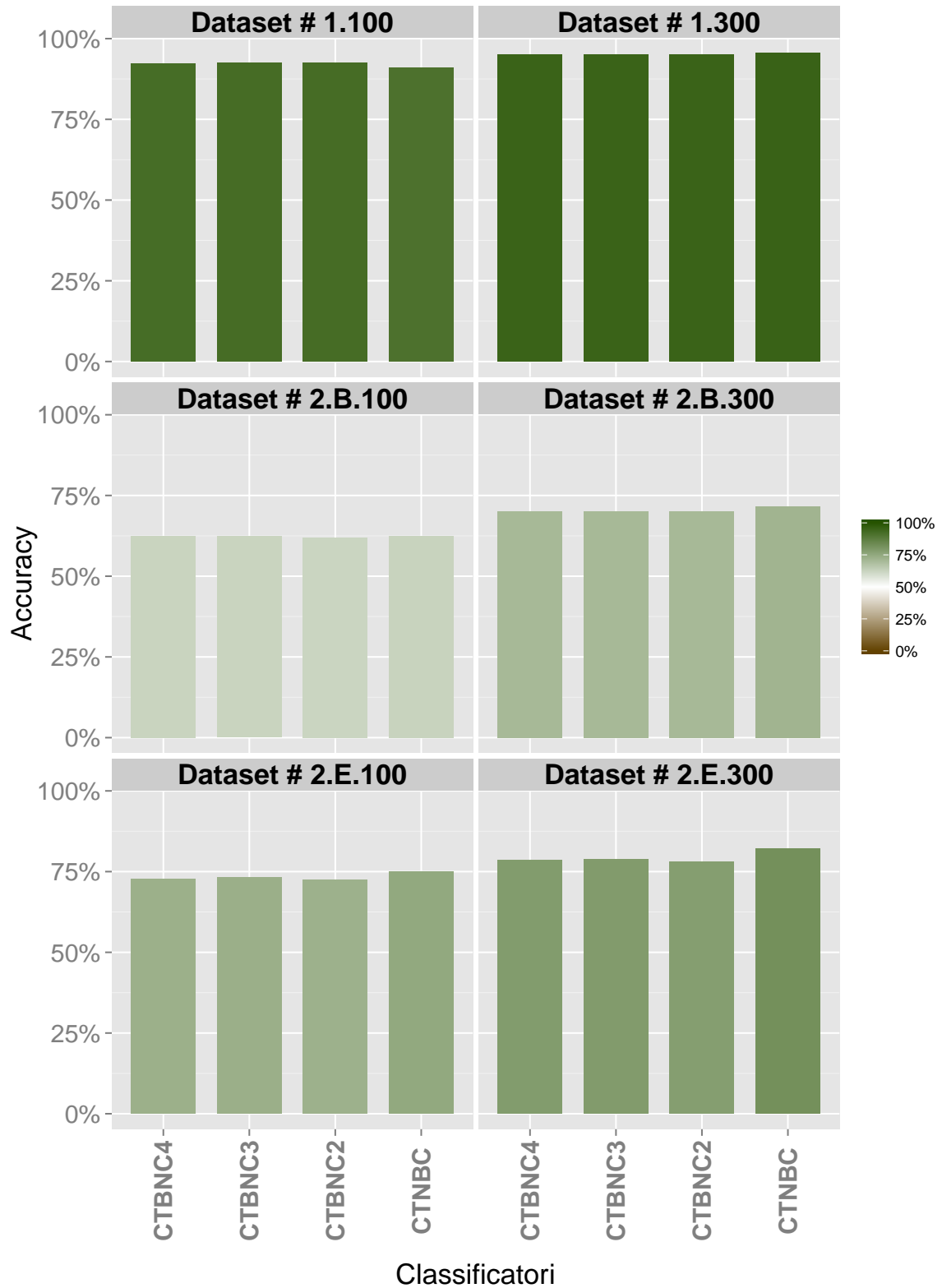


Figura 6.6: Comparazione del valore di *accuracy* ottenuto dal classificatore CTNB e dai classificatori CTBN (appresi con numero massimo di genitori variabile da 2 a 4) su ognuno dei 6 dataset generati.

dataset #2.B.100 e il dataset #2.B.300.

Riguardo i 2 dataset appena citati, inoltre, è possibile notare come il livello generale di accuratezza che tutti i classificatori ottengono per essi sia ben inferiore rispetto a quello ottenuto sugli altri dataset. Anche in questo caso, si ritiene che ciò sia dovuto a un aspetto informativo dei dataset. Infatti, le istanze del dataset #2.B.100 e del dataset #2.B.300 contengono solamente le traiettorie relative ai 6 sensori reali del dataset #2 (si veda la sottosezione 6.2.2 a pagina 86).

Infine, è possibile notare come l'accuratezza dei classificatori tenda a degradare all'aumentare della complessità del sistema dinamico rappresentato dal dataset. Infatti, la differenza tra il livello di accuratezza ottenuto per il dataset #1 e quello ottenuto per il dataset #2 è significativa, tanto quanto è significativa la differenza del livello di complessità delle reti stradali che essi rappresentano.

6.3.3 Curve ROC

L'obiettivo che questa sottosezione si prefigge è quello di completare la presentazione dei risultati ottenuti dalla sperimentazione dei classificatori CTBN (CTBNC). A tale scopo si presentano e comparano le curve Receiver Operating Characteristic (ROC) di ogni classificatore utilizzato per la sperimentazione (i. e., CTNBC, CTBNC2, CTBNC3 e CTBNC4).

Prima di tale discussione, tuttavia, è necessario introdurre i concetti relativi alla metrica ROC. La visualizzazione della metrica ROC di uno o più classificatori è una tecnica finalizzata alla valutazione e alla comparazione dell'efficacia dei classificatori. La metrica ROC raffigura la correlazione, e spesso il *tradeoff*, tra il valore della metrica *recall* (i. e., *sensitivity*) e il tasso di falsi positivi (i. e., *false alarm error*; metrica FPR).

Questa metrica, inizialmente utilizzata in altri campi di ricerca (e. g., teoria delle decisioni), è sempre più frequentemente utilizzata anche nel campo dell'apprendimento automatico. Ciò poiché l'utilizzo della sola *accuracy* come metrica per la valutazione dell'efficacia dei classificatori si rivela spesso carente dal punto di vista informativo e perciò inadeguato (per approfondimenti si veda Provost e Fawcett, 2001, 1997).

Inoltre, la metrica ROC gode di una serie di proprietà utili nel caso di situazioni molto comuni nel mondo reale (e. g., insensibilità all'asimmetria delle classi): essa disaccoppia la valutazione dell'efficacia del classificatore dalla presenza o meno di classi asimmetriche o di funzioni di costo sbilanciate.

I grafici ROC sono grafici bidimensionali la cui ordinata riporta i valori assunti dalla metrica TPR mentre l'ascissa riporta i valori assunti dalla metrica FPR.

Si osservi quindi che un grafico ROC è un tipico grafico raffigurante il *tradeoff* tra costi (i. e., FPR) e benefici (i. e., TPR).

Si presentano di seguito alcuni punti chiave di ogni grafico ROC.

- il punto con coordinate $(0,0)$ rappresenta il caso in cui il classificatore non commette mai errori di primo tipo (i.e., False Positive (FP)) ma nemmeno classifica positivamente le istanze (i.e., True Positive (TP))
- il punto con coordinate $(1,1)$ rappresenta la situazione opposta alla precedente: il classificatore classifica tutte le istanze in modo positivo, anche quelle che dovrebbe invece rigettare
- il punto $(0,1)$ rappresenta la classificazione perfetta.

L'interpretazione dei punti nei grafici ROC può essere informalmente descritta in base alla loro posizione: un punto più a nord-est rispetto agli altri rappresenta una classificazione migliore. Si osservi che tale considerazione deriva dal fatto che tale punto avrebbe un'ordinata maggiore degli altri, e quindi un più alto valore TPR, e al contempo un'ascissa inferiore, cioè un valore FPR inferiore.

È bene precisare che, nel caso di *classificatori probabilistici*, quali sono i classificatori CTBN, è necessario trasformarli in *classificatori discreti* (i.e., classificatori che non forniscono un valore di probabilità come risultato bensì valori di verità) al fine di ottenere un singolo punto nello spazio del grafico ROC (il quale corrisponde ad un'unica *matrice di confusione*). Idealmente tale processo è svolto variando un valore di soglia in un intervallo. Per ogni soglia, si seleziona un esito positivo o negativo della classificazione in base alla probabilità restituita dal classificatore probabilistico; cioè qualora essa sia maggiore o minore del corrente valore soglia. Così facendo si ottiene quindi una curva ROC per un qualsiasi classificatore probabilistico.

Si noti che la procedura appena descritta non è computazionalmente efficiente. Tuttavia in letteratura sono state presentate procedure efficienti (si veda Fawcett, 2006, sezione 5).

Nel caso di un problema di *classificazione multi-classe*, come illustrato precedentemente, la *matrice di confusione* corrisponde a una matrice $n \times n$ (in questo caso $n = 6$ per tutti i dataset generati, come mostrato dalla tabella 6.9 a pagina 90) contenente le n classificazioni corrette (i.e., benefici) sulla diagonale e i $n^2 - n$ possibili errori (i.e., costi) nelle celle non sulla diagonale. Un possibile approccio risolutivo a tale problema consiste nel produrre n differenti grafici ROC, uno per classe (Fawcett, 2006). Per le sperimentazioni effettuate in questo lavoro di tesi si è appunto attuato tale approccio. Anche se, come già specificato, le classi dei dataset generati non presentano asimmetrie quantitative considerevoli, è tuttavia necessario notare che tale approccio provoca la perdita della proprietà di insensibilità all'asimmetria delle classi delle metriche ROC.

Si rimarca infine che le metriche ROC sono state calcolate tramite l'approccio *micro-average*.

Le figure da 6.7 a 6.12 nelle pagine 98–103 riportano i grafici delle curve ROC di ognuno dei 4 classificatori utilizzati per la classificazione del dataset #1.100. Ognuna di tali figure, come detto, è relativa alla curva ROC di una classe del dataset in questione.

Si osservi che nessuna curva ROC dei classificatori differisce dalle altre in modo significativo. Al contempo, si osservi che poiché esse si avvicinano considerevolmente alla coordinata $(0, 1)$ ciò indica che tutti i classificatori, in questo caso, raggiungono degli ottimi livelli di classificazione.

Il fatto che i classificatori CTBN appresi non ottengano risultati che si discostino considerevolmente (né in positivo, né in negativo) da quelli ottenuti dal classificatore CTNB è, con tutta probabilità, da attribuire alla complessità dei sistemi dinamici rappresentati dai dataset relativi al traffico. Con tali dataset, i classificatori appresi con la procedura di apprendimento strutturale descritta in questo lavoro di tesi, ricevono una penalità dovuta alla loro eccessiva aderenza alle interrelazioni presenti nei modelli di traffico rappresentati.

Codecasa e Stella (2013) hanno infatti mostrato come l'apprendimento strutturale costituisca una procedura sensata per questa classe di modelli poiché i classificatori CTBN appresi tramite un diverso approccio (i. e., basato sulla log-likelihood condizionale) risultano essere maggiormente efficaci.

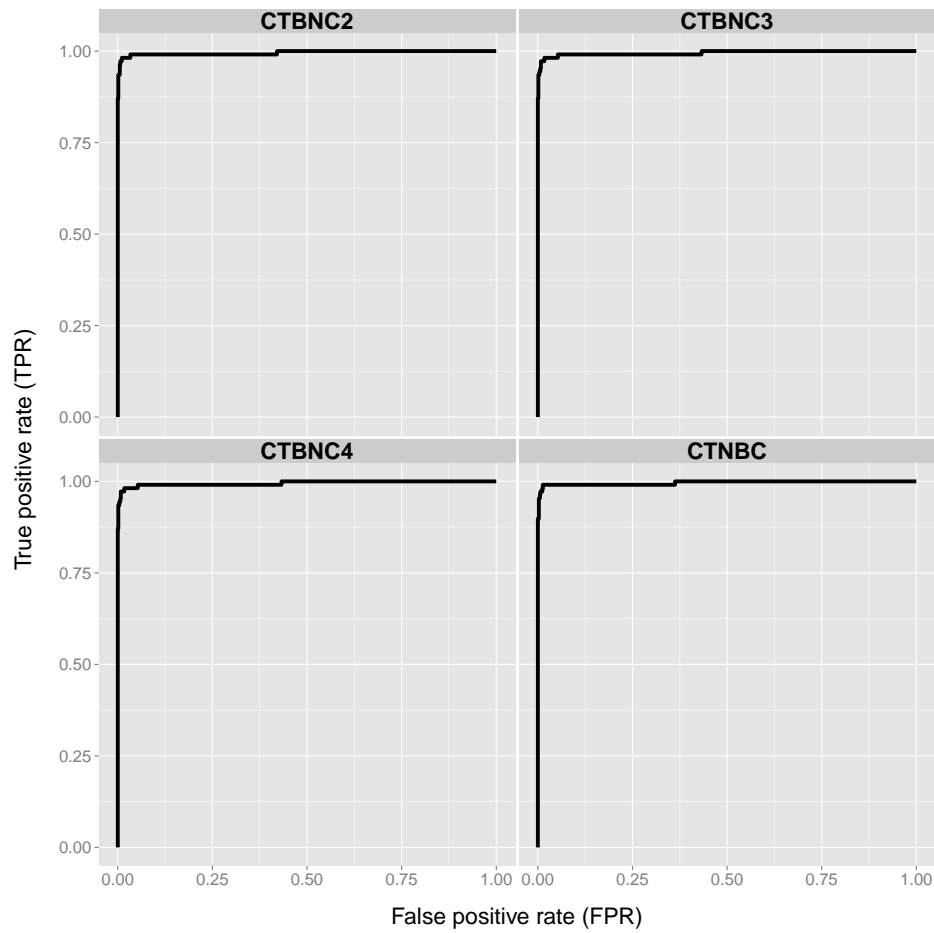


Figura 6.7: Curve ROC, relative alla classe 1, di ogni classificatore utilizzato nel processo di sperimentazione.

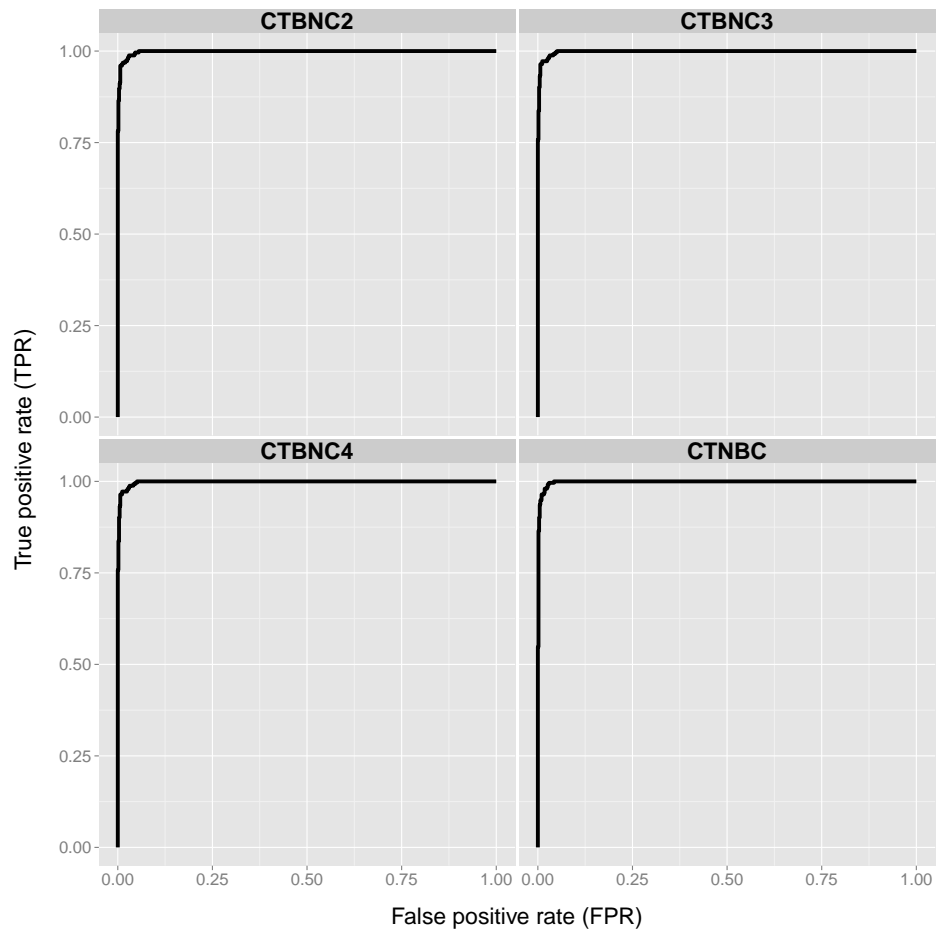


Figura 6.8: Curve ROC, relative alla classe 2, di ogni classificatore utilizzato nel processo di sperimentazione.

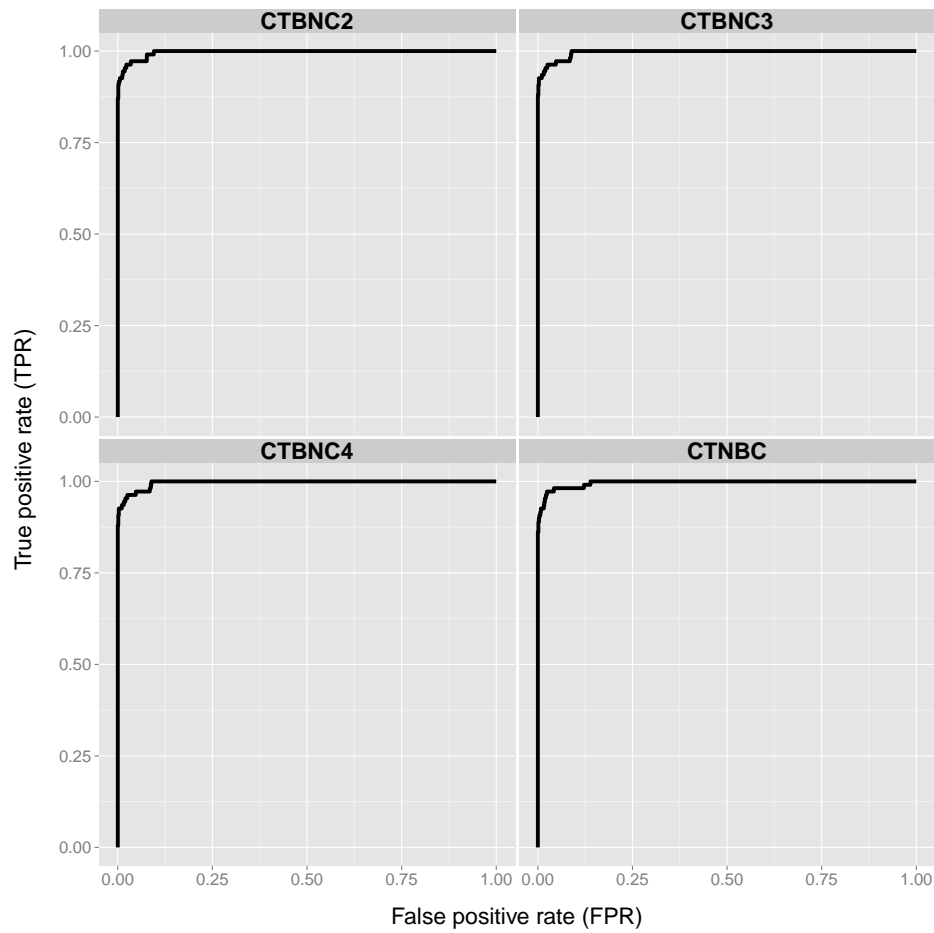


Figura 6.9: Curve ROC, relative alla classe 3, di ogni classificatore utilizzato nel processo di sperimentazione.

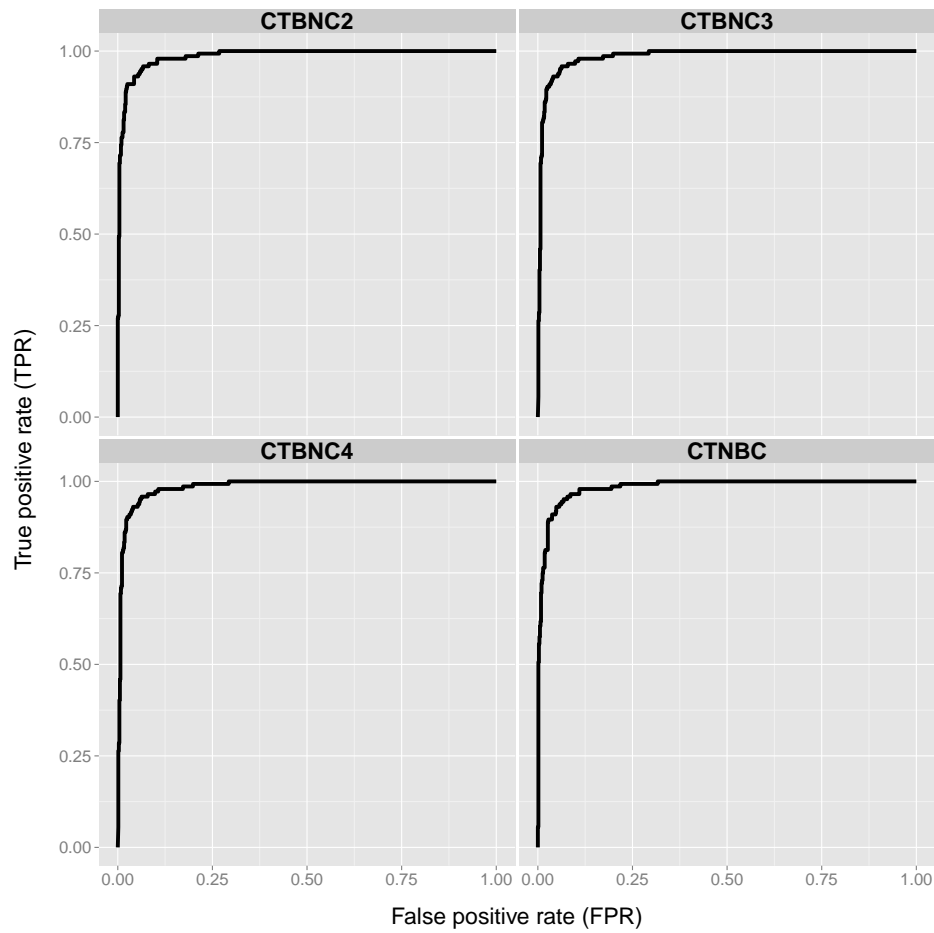


Figura 6.10: Curve ROC, relative alla classe 4, di ogni classificatore utilizzato nel processo di sperimentazione.

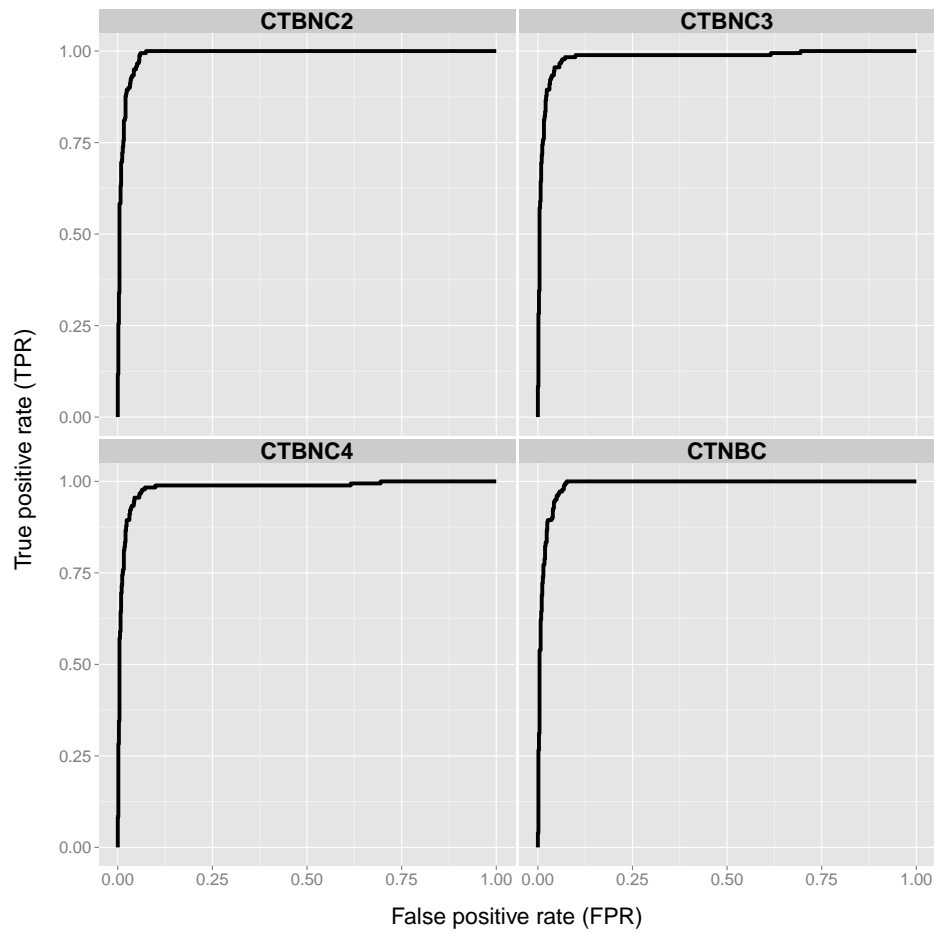


Figura 6.11: Curve ROC, relative alla classe 5, di ogni classificatore utilizzato nel processo di sperimentazione.

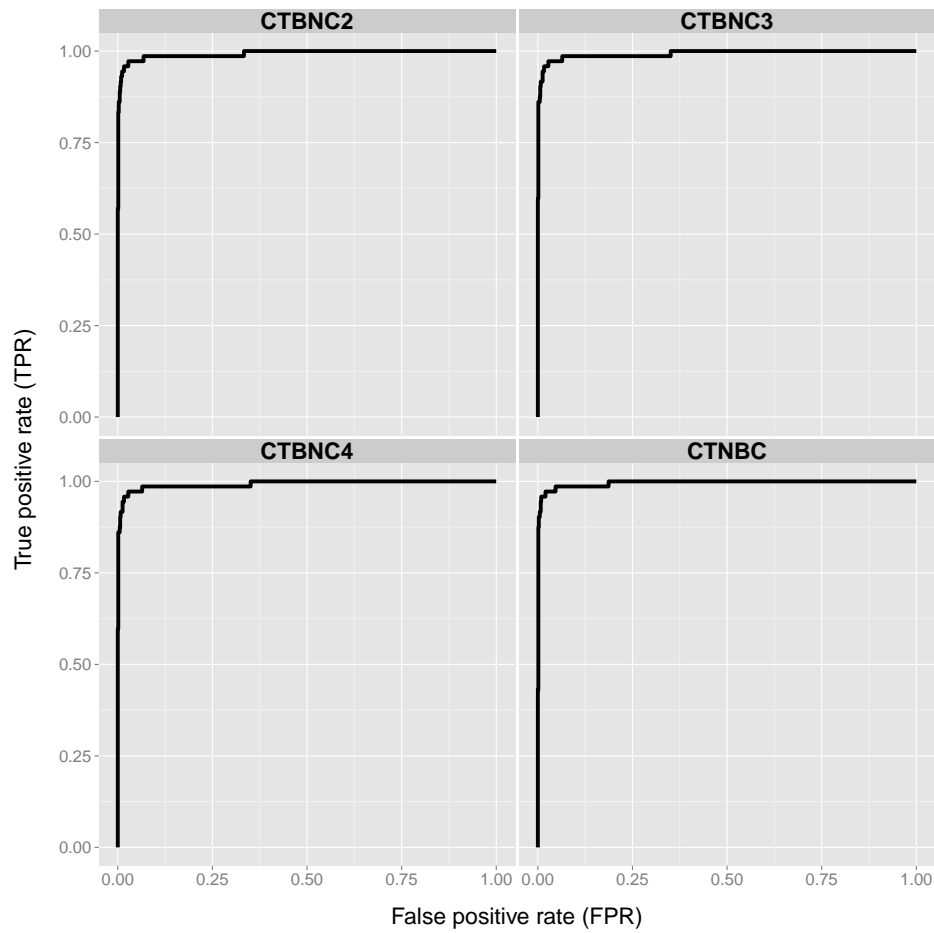


Figura 6.12: Curve ROC, relative alla classe 6, di ogni classificatore utilizzato nel processo di sperimentazione.

CONCLUSIONI

Questo lavoro di tesi è stato incentrato sul framework delle Continuous time Bayesian Network (CTBN), e sulla descrizione di una classe di modelli di *classificatori* da esso derivanti.

Come ampiamente descritto e motivato, il principale vantaggio apportato dalle CTBN consiste nella loro capacità di rappresentare esplicitamente *sistemi dinamici* che evolvono nel *tempo continuo* senza necessitare della scelta a priori di un'arbitraria granularità temporale, a differenza di altri modelli grafico probabilistici con scopi simili (e.g., Dynamic Bayesian Networks). Inoltre, tale modello non possiede il vincolo di aciclicità. Ciò permette di modellare in modo naturale le mutue influenze fra le variabili di un dato sistema dinamico e di facilitare il processo di apprendimento strutturale, permettendo di ricercare l'insieme di nodi genitori di ogni nodo in modo indipendente da quello dei restanti nodi del modello.

Si è quindi descritto il processo di *apprendimento dei parametri* per le CTBN e il conseguente algoritmo di *apprendimento dei classificatori* CTBN (CTBNC), entrambi relativi a dati completi. Come naturale prosecuzione del processo di apprendimento dei classificatori si è descritto l'algoritmo di *inferenza esatta* per i CTBNC, il quale implementa un *tradeoff* fra complessità computazionale e efficacia di classificazione.

Come detto, i classificatori CTBN (CTBNC) sono un modello grafico probabilistico applicabile al problema della *classificazione supervisionata* nel caso in cui le seguenti condizioni sussistano:

- gli attributi sono discreti (o discretizzabili)
- il tempo fluisce continuamente
- si dispone di dati completi
- la classe si verifica nel futuro.

Si è infine descritto un processo di *apprendimento strutturale* basato sull'ottimizzazione, attuata tramite una procedura di ricerca euristica (i.e., *hill climbing*), di una *funzione di punteggio* per le CTBN.

Nella seconda parte del presente elaborato, invece, si sono affrontati argomenti di tipo pratico.

In primis, si è presentato RCTBN, il *pacchetto* R sviluppato al fine di implementare il framework delle CTBN e gli algoritmi descritti nella parte teorica della tesi.

Successivamente si è descritto il software commerciale TSIS utilizzato per la *modellazione di profili di traffico su reti stradali urbane* e Sensors DLL, una sua estensione a tempo d'esecuzione sviluppata al fine di

monitorare e tracciare il passaggio dei veicoli sui *sensori* presenti in tali reti stradali.

La progettazione e l'implementazione di tali applicativi si è resa necessaria allo scopo di generare dei *dataset* che rappresentassero dei sistemi dinamici complessi, qual è il traffico automobilistico, da sottoporre ai classificatori CTBN (CTNBC). Grazie a tali dataset si è potuto effettuare una *sperimentazione* di 4 diverse istanze di classificatori CTBN allo scopo di valutare la loro applicazione a un problema noto e complesso qual è la *classificazione dei profili di traffico*.

Tale sperimentazione ha permesso inoltre di comparare l'efficacia dei vari modelli di classificatore CTBN utilizzati: il classificatore Continuous time Naive Bayes (CTNBC) e i classificatori appresi tramite il succitato algoritmo di apprendimento strutturale. Dai *risultati* ottenuti si è evinto che tali modelli ottengono un buon livello di classificazione. Tuttavia, a causa della natura dei sistemi dinamici descritti dai dataset, non si sono rilevate differenze sostanziali fra i vari modelli di classificatori utilizzati.

Ciò pone quindi un primo spunto per gli eventuali *sviluppi futuri* del corrente lavoro di tesi: l'elaborazione di una procedura di apprendimento strutturale alternativa che porti all'ottenimento di classificatori CTBN maggiormente efficaci.

Riguardo tale argomento, è sicuramente di notevole interesse, per i risvolti che ciò può comportare, affrontare nel futuro gli stessi argomenti presentati in questo lavoro di tesi relativamente però a insiemi di dati non completamente osservati.

Anche dal punto di vista pratico, questo lavoro di tesi può essere esteso in vari modi. Ad esempio, l'applicativo per la generazione di dataset relativi al traffico necessita di conformare completamente i dati che esso genera alla forma che le CTBN richiedono. Ci si riferisce, nello specifico, alla rilevazione degli istanti temporali in cui più di un sensore effettua una transizione di stato; operazione attualmente non implementata. È importante rimarcare, infatti, che i modelli CTBN assumono che due variabili non possano effettuare una transizione contemporaneamente. Tale assunzione può essere vista come una formalizzazione del fatto che le variabili devono rappresentare aspetti distinti del mondo: in generale, non si dovrebbe modellare un dominio in cui due o più variabili cambiano stato contemporaneamente in modo deterministico.

Lo scopo di questa appendice è formulare delle guide all'utilizzo del software sviluppato a sostegno del lavoro di tesi.

Le sezioni tematiche sono due, una relativa all'utilizzo delle principali funzionalità offerte dal pacchetto RCTBN e una relativa all'utilizzo di Sensors DLL per la generazione di dataset da modelli TSIS.

Tutti gli esempi presentati si riferiscono alla variante estesa, con elementi da 100 secondi, del dataset relativo a *Viale Cesare Battisti* (i.e., dataset #2.E.100). In questo capitolo, per brevità, ci si riferirà a tale dataset semplicemente con il termine dataset #2.

A.1 UTILIZZO DEL PACCHETTO RCTBN

Nelle seguenti sottosezioni si illustrano le funzionalità principali offerte da RCTBN. Lo scopo è quello di guidare l'utente all'utilizzo del framework delle CTBN tramite il succitato pacchetto R.

Prima di addentrarci nella guida all'utilizzo del pacchetto RCTBN è necessario verificare che il sistema disponga dell'adeguato ambiente R (versione ≥ 3.0). Inoltre è necessario che il pacchetto `devtools`⁴⁸ sia correttamente installato. In caso contrario, purché si disponga di una connessione a Internet, è possibile installare l'ultima versione stabile digitando il seguente comando in una nuova sessione R.

```
install.packages("devtools", dependencies = TRUE)
```

Sorgente A.1: Installazione del pacchetto `devtools`.

A questo punto è possibile procedere all'installazione del pacchetto RCTBN. Poiché esso non è ancora stato pubblicato nell'archivio online dei pacchetti R⁴⁹, la procedura di installazione differisce da quella precedentemente mostrata. A tale scopo si utilizza quindi il pacchetto `devtools` per installare RCTBN. Si osservi che il pacchetto `devtools` permette di creare e installare un pacchetto R a partire dal suo sorgente (purché esso rispetti le dovute convenzioni e contenga codice eseguibile senza errori), che sia esso online o sul sistema dell'uten-

⁴⁸ Il pacchetto `devtools` (Wickham e Chang, 2013) è un insieme di funzioni e strumenti finalizzati all'espletamento di processi comuni e fondamentali per lo sviluppo in R. Il codice sorgente di tale pacchetto è disponibile all'indirizzo: <https://github.com/hadley/devtools>.

⁴⁹ Una delle modalità di installazione di pacchetti R è tramite connessione a un archivio online in cui essi vengono memorizzati, il CRAN.

te, o da un archivio compresso. Perciò, disponendo del sorgente di RCTBN, è possibile installarlo come pacchetto R digitando le seguenti istruzioni.

```
1 library(devtools)
2 install("percorso/sorgente/rctbn", dependencies = TRUE)
```

Sorgente A.2: Installazione del pacchetto RCTBN.

Si osservi che l'opzione `dependencies` fa sì che tutte i pacchetti da cui RCTBN dipende vengano scaricati e installati dall'archivio CRAN: anche in questo caso è quindi necessaria una connessione a Internet.

È ora possibile utilizzare il pacchetto RCTBN per l'esecuzione dei vari processi relativi al framework delle CTBN. Si osservi che esso viene installato con il nome `ctbn` nella libreria R.

Tutti gli esempi d'utilizzo che vengono mostrati in questa sezione assumono che la cartella di lavoro sia `~/Workspace/R/test/ctbn`. Perciò, come mostrato in seguito, si utilizza il comando `setwd` per impostare la cartella di lavoro.

A.1.1 Gestione dei dataset

Come detto nella sezione 4.2 a pagina 43, il pacchetto RCTBN implementa una serie di funzionalità, non correlate concettualmente al framework delle CTBN, utili allo svolgimento di operazioni di gestione dei dataset. Lo scopo principale di questo insieme di funzionalità è permettere il caricamento e la compressione su disco dei dataset per le CTBN, così da evitare all'utente la ripetizione di tale operazione ogni qual volta egli necessiti di un dataset precedentemente già importato.

Perciò, di seguito si illustra come importare il dataset #2 (descritto nella sezione 6.2 a pagina 78).

```
1 setwd("~/Workspace/R/test/ctbn")
2 library(ctbn)
3 read_dataset("monza100e")
```

Sorgente A.3: Importazione e serializzazione del dataset #2.

Si osservi che, come anticipato, è indispensabile impostare la corrente cartella di lavoro (comando `setwd`) poiché il pacchetto RCTBN utilizza tale informazione per cercare il dataset da importare. Nello specifico, RCTBN cercherà nella cartella di lavoro una cartella `dataset` e dentro questa cercherà una cartella con il nome del primo parametro attuale della funzione `read_dataset`. È possibile modificare la cartella in cui RCTBN cerca i dataset modificando l'opzione `ctbn.data.dir`, impostata di default al valore `dataset`, tramite il comando `options`.

La funzione **read_dataset** effettua varie operazioni, che vengono descritte di seguito.

- Legge ogni file CSV presente nella cartella:
`"~/Workspace/R/test/ctbn/dataset/monza100e".`
 È possibile importare file con altre estensioni semplicemente modificando il valore dell'opzione `ctbn.data.ext`, impostato all'avvio al valore `"csv"`.
- Controlla che ogni struttura tabulare letta contenga la colonna relativa al tempo. Tale informazione è specificabile tramite l'opzione `ctbn.col.time`, impostata di default al valore `"time"`.
- Controlla che ogni struttura tabulare letta contenga la colonna relativa alla classe. Tale informazione è specificabile tramite l'opzione `ctbn.col.class`, impostata all'avvio a `"class"`. Nel caso in cui non si sia interessati ad eseguire tale controllo è possibile ometterlo semplicemente modificando la chiamata in questione come mostrato di seguito:
`read_dataset("monza100e", classcheck = FALSE).`
- Controlla che ogni file contenga altre colonne oltre a quelle relative al tempo e, opzionalmente, alla classe.
- Ordina le strutture tabulari importati in base alle rispettive colonne temporali.
- Crea un oggetto R simile a una tabella hash dove ogni chiave è rappresentata da una stringa, il nome del file importato (senza estensione), e ogni elemento è una tabella (nello specifico un oggetto della classe `data.table`) contenente la struttura tabulare importata.
- Serializza in modo compresso tale oggetto su file salvando un file `RDATA`⁵⁰ nella cartella adibita alla memorizzazione dei dataset importati (specificata dall'opzione `ctbn.store.dir`, il cui valore di default è `"data"`). È possibile, ma non consigliato, disattivare la serializzazione del dataset importato modificando la chiamata in questione come mostrato:
`read_dataset("monza100e", autosave = FALSE).`

L'output del sorgente A.3 corrisponde all'oggetto R contenente il dataset nelle modalità descritte. Ciò significa che assegnando tale funzione a una variabile sarà possibile operare con il dataset in questione. Ad esempio:

```
monzads = read_dataset("monza100e").
```

⁵⁰ I file con estensione `RDATA`, o `RDA`, sono dei formati binari che R utilizza per la memorizzazione su disco di oggetti appartenenti al suo linguaggio.

Si osservi che, nel caso in cui non l'opzione `ctbn.verbose` (posta di default a `TRUE`) non sia stata modificata, la funzione **read_dataset** stampa a video il seguente messaggio:

```
Dataset directory: "~/Workspace/R/test/ctbn/dataset/monza100e"
Number of files imported: 4320
Dataset store: "~/Workspace/R/test/ctbn/data/monza100e/store.rdata"
```

Quindi, eseguito il sorgente A.3, il dataset sarà stato importato e serializzato come oggetto R al seguente percorso:

```
"~/Workspace/R/test/ctbn/data/monza100e/store.rdata".
```

Si osservi che anche tale comportamento è personalizzabile tramite l'utilizzo delle opzioni che RCTBN fornisce all'utente. Nello specifico è possibile modificare l'opzione `ctbn.store.name`, impostata di default al valore `"store"`, e l'opzione `ctbn.store.ext`, impostata all'avvio al valore `"rdata"`.

Si mostra ora come sfruttare l'operazione appena illustrata per utilizzare il dataset #2 in una nuova sessione R senza necessità di importarlo nuovamente, processo che ha richiesto circa 45 secondi. Tale caratteristica di RCTBN permette di ottenere un notevole risparmio di tempo nel flusso di lavoro.

```
1 setwd("~/Workspace/R/test/ctbn")
2 library(ctbn)
3 load_dataset("monza100e", verbose = TRUE)
```

Sorgente A.4: Caricamento del dataset #2.

Come detto, la funzione **load_dataset** non importa nuovamente il dataset `"monza100e"`, bensì essa cerca il relativo file RDATA nella cartella apposita e inietta dinamicamente l'oggetto da esso contenuto nell'attuale ambiente R. Ciò significa che l'esecuzione del sorgente A.4 permette di avere istantaneamente accesso a un oggetto R memorizzato in una variabile `monza100e`.

Si osservi infine che l'intero processo di gestione dei dataset contempla la possibilità che l'utente utilizzi dei dataset il cui nome non corrisponde a un nome di variabile valido per R. In tale situazione il pacchetto RCTBN provvede a generare un nome di variabile valido a partire dal nome del dataset specificato. Perciò, una volta importato o caricato tale dataset sarà possibile ricavare il suo nome ispezionando l'ambiente R eseguendo il comando `ls(all = TRUE)`, il quale elenca tutte le variabili disponibili nell'ambiente R attuale. Alternativamente, qualora l'opzione `ctbn.verbose` abbia valore `TRUE`, la funzione **load_dataset** mostrerà a video un messaggio, finalizzato alla comunicazione del nome della variabile in cui il dataset è stato iniettato.

```
Dataset injected into variable: monza100e.
```

A.1.2 Calcolo delle statistiche sufficienti

In questa sottosezione si illustrano le funzionalità che RCTBN fornisce affinché l'utente possa computare le *statistiche sufficienti*.

I sorgenti mostrati assumono che il pacchetto RCTBN sia stato caricato e che il dataset #2 sia stato importato e caricato.

Si suppone innanzitutto di voler calcolare le *statistiche sufficienti* di una o più variabili del dataset #2 dato uno specifico insieme di genitori. A tale scopo RCTBN fornisce la funzione **fsstats**.

Perciò, di seguito si illustra come computare le *statistiche sufficienti* su tutto il dataset #2 del nodo "D241" impostando come suo insieme dei genitori l'insieme composto dai nodi "D911", "D191" e "D641".

```

1 options(width = 190)
2 options(ctbn.verbose = FALSE)
3 fsstats(monza100e, "D241", c("D911", "D191", "D641"))

```

Sorgente A.5: Esempio di calcolo delle *statistiche sufficienti* sul dataset #2.

Il risultato di tale esecuzione è una tabella hash contenente un solo elemento: le *statistiche sufficienti* del nodo "D241" dati i nodi genitori "D911", "D191" e "D641". La chiave di tale elemento della tabella hash è una rappresentazione di tale informazione mentre l'elemento è un oggetto tabulare (classe `data.table`) rappresentante i valori delle *statistiche sufficienti* T e M per ogni combinazione dei valori assunti dai nodi genitori.

Di seguito si riporta l'output fornito dal sorgente A.5.

\$ 'D241 D911,D191,D641'			
	PARENT_VAL	T	MM
1:	0 0 0	58447.50500000026,2406.80499999877	0,21594,21544,0
2:	0 0 1	4561.4049999992,237.904999999637	0,760,726,0
3:	0 1 0	1586.1050000002,57.4050000001183	0,133,134,0
4:	0 1 1	249.605000000317,5.00500000000728	0,6,3,0
5:	1 0 0	15379.2050000001,881.904999999659	0,5078,4960,0
6:	1 0 1	1725.80499999955,89.6049999997475	0,198,207,0
7:	1 1 0	340.005000000116,16.3049999999629	0,26,25,0
8:	1 1 1	72.1050000000749,1.10500000000582	0,4,2,0

Si osservi che **fsstats** non permette di computare le *statistiche sufficienti* di più variabili specificando per ognuna di esse un insieme di genitori diverso. Tale funzione è quindi utilizzabile con profitto nel caso in cui si intenda calcolare le *statistiche sufficienti* di più variabili (eventualmente anche tutte) rispetto a un unico insieme di genitori: si pensi ad esempio a ciò che è necessario fare quando si intende apprendere un classificatore CTNB (CTNBC) (mostrato in figura 2.2 a pagina 23), caratterizzato proprio dal fatto che ogni sua variabile ha un solo nodo genitore, il nodo classe.

Ipotizziamo perciò di volere utilizzare la funzione **fsstats** per calcolare le *statistiche sufficienti* di alcune variabili (ad esempio le variabili "D241", "D781" e "D782") data la sola variabile "class" (i.e., nodo classe) come genitore. Il sorgente che segue illustra come espletare tale obiettivo.

```

1 options(width = 190)
2 options(ctbn.verbose = FALSE)
3 fsstats(monza100e, c("D241", "D781", "D782"), "class")

```

Sorgente A.6: Esempio di calcolo delle *statistiche sufficienti* di più nodi dato un insieme di genitori comune, il nodo classe (dataset #2).

Come visibile dall'output conseguente l'esecuzione del sorgente A.6, in questo caso, la funzione **fsstats** restituisce una tabella hash con più elementi, una per ogni nodo oggetto della computazione delle *statistiche sufficienti*.

```

$'D241|class'
  PARENT_VAL          T          MM
1:         1 14712.405,206.405000000052 0,1130,1127,0
2:         2 7327.204999999996,155.705000000003 0,1116,1112,0
3:         3 14126.705,815.1050000000082 0,11579,11585,0
4:         4 21124.805000000008,1282.80499999921 0,20536,20527,0
5:         5 14127.505000000008,816.604999999242 0,12379,12382,0
6:         6 10943.10500000006,419.404999999296 0,4224,4225,0

$'D781|class'
  PARENT_VAL          T          MM
1:         1 14246.905,671.904999999989 0,2562,2571,0
2:         2 6867.804999999994,615.1050000000049 0,3239,3236,0
3:         3 11750.305000000001,3191.505000000003 0,36858,36870,0
4:         4 17491.705,4915.905000000003 0,63441,63467,0
5:         5 11409.005000000003,3535.10499999977 0,42700,42668,0
6:         6 9245.60499999988,2116.905000000107 0,14291,14280,0

$'D782|class'
  PARENT_VAL          T          MM
1:         1 14628.505,290.304999999998 0,1018,1024,0
2:         2 7219.304999999984,263.6050000000151 0,1338,1335,0
3:         3 12751.7049999997,2190.105000000045 0,23599,23601,0
4:         4 18641.0049999993,3766.605000000076 0,43463,43465,0
5:         5 12214.20499999988,2729.905000000127 0,30061,30055,0
6:         6 9943.105000000006,1419.40499999981 0,9529,9534,0

```

Si osservi che la funzione **fsstats** permette anche di calcolare le *statistiche sufficienti* di uno o più nodi senza condizionare la loro computazione ad alcun genitore.

Di seguito viene illustrato come effettuare tale operazione.

```

1 options(width = 190)
2 options(ctbn.verbose = FALSE)

```

```
3 fsstats(monza100e, "D241", parents = NULL)
```

Sorgente A.7: Esempio di calcolo delle *statistiche sufficienti* di un nodo senza condizionare la computazione a un insieme di genitori (dataset #2).

Nel caso in cui si desideri computare le *statistiche sufficienti* di più nodi rispetto a diversi insiemi di genitori è invece necessario utilizzare la funzione **fsstats_graph**. La differenza tra tale funzione e quella appena illustrata consiste principalmente nella firma. Infatti, essa non richiede in input un parametro relativo ai nodi e uno relativo all'insieme di genitori comune, bensì una matrice binaria avente sulle righe i nodi genitori e sulle colonne i nodi figli, il cui scopo è rappresentare la struttura di una CTBN. Un 1 posto nella posizione (3,2) (i. e., riga 3, colonna 2) indica che il nodo 2 è figlio del nodo 3.

A titolo esemplificativo si ipotizzi di voler calcolare le *statistiche sufficienti*:

- del nodo "D241" dato il solo nodo "class" come genitore
- del nodo "D781" dati nodi genitori "D981" e "D241"
- del nodo "D782" dati nodi genitori "class" e "D981".

Tale insieme di informazioni corrisponde alla matrice mostrata dalla seguente tabella.

	D241	D781	D782
class	1	0	1
D981	0	1	1
D241	0	1	0

Tabella A.1: Struttura d'esempio di una CTBN espressa tramite la corrispondenza fra i nodi e relativi insiemi di genitori.

Il sorgente che segue illustra come costruire tale matrice e utilizzarla come parametro di input della funzione **fsstats_graph**.

```
1 options(width = 190)
2 options(ctbn.verbose = FALSE)
3 am = matrix(c(1, 0, 1, 0, 1, 1, 0, 1, 0), 3, 3, byrow = T)
4 colnames(am) = c("D241", "D781", "D782")
5 rownames(am) = c("class", "D981", "D241")
6 fsstats_graph(monza100e, am)
```

Sorgente A.8: Esempio di calcolo delle *statistiche sufficienti* di più nodi aventi diversi insiemi di nodi genitori (dataset #2).

L'output ottenuto dall'esecuzione del sorgente A.8 illustra le *statistiche sufficienti* dei nodi scelti, dati i loro rispettivi insiemi di nodi genitori, computate sul dataset #2.

\$'D241 class'			
	PARENT_VAL	T	MM
1:	1	14712.405,206.405000000052	0,1130,1127,0
2:	2	7327.20499999996,155.705000000003	0,1116,1112,0
3:	3	14126.705,815.1050000000082	0,11579,11585,0
4:	4	21124.80500000008,1282.80499999921	0,20536,20527,0
5:	5	14127.50500000008,816.604999999242	0,12379,12382,0
6:	6	10943.10500000006,419.404999999296	0,4224,4225,0
\$'D781 D981,D241'			
	PARENT_VAL	T	MM
1:	0 0	60540.005,12220.10500000016	0,110269,109849,0
2:	0 1	2558.60499999829,591.404999999786	0,561,578,0
3:	1 0	7482.605000000092,2119.00499999965	0,5477,5477,0
4:	1 1	430.104999999902,115.90499999929	0,35,40,0
\$'D782 class,D981'			
	PARENT_VAL	T	MM
1:	1 0	14021.505,274.704999999976	0,925,919,0
2:	1 1	607.0050000000023,15.6050000000035	0,13,12,0
3:	2 0	7018.30499999982,257.7050000000139	0,1284,1288,0
4:	2 1	201.0050000000015,5.90500000001237	0,7,6,0
5:	3 0	10760.00499999994,1717.305000000055	0,15531,15634,0
6:	3 1	1991.705000000027,472.804999999901	0,1254,1231,0
7:	4 0	15555.0049999997,2966.505000000079	0,28569,28382,0
8:	4 1	3086.00499999953,800.104999999962	0,2566,2562,0
9:	5 0	10448.0049999985,2236.105000000137	0,21444,21513,0
10:	5 1	1766.205000000029,493.804999999901	0,1382,1365,0
11:	6 0	9333.7049999995,1321.30499999987	0,8542,8561,0
12:	6 1	609.4050000000562,98.1049999999331	0,103,98,0

A.1.3 Apprendimento dei parametri

Si illustra ora come effettuare l'apprendimento dei parametri di una CTBN da un insieme di *dati completi*, nello specifico dal dataset #2, tramite la relativa funzionalità offerta dal pacchetto RCTBN. Si ricorda, come discusso nella sottosezione 1.4.3 a pagina 16, che è possibile effettuare la stima dei parametri tramite due approcci: la stima maximum-likelihood o la stima bayesiana.

Supponendo di considerare la CTBN con la struttura utilizzata per l'esempio precedente (si veda la tabella A.1 nella pagina precedente) e di aver salvato le *statistiche sufficienti* calcolate nel sorgente A.6 in una variabile chiamata *ss*, è possibile apprendere i parametri fornendo tale variabile in input alla funzione **params**.

```

1 options(width = 190)
2 options(ctbn.verbose = FALSE)
3 params(ss)

```

Sorgente A.9: Apprendimento (i. e., stima esatta) dei parametri di una CTBN dal dataset #2.

Tale funzione richiede in input una tabella hash contenente le *statistiche sufficienti* di uno o più nodi (qualsiasi sia il loro insieme di genitori), cioè un oggetto restituito dalle funzioni **fsstats_graph** o **fsstats**.

L'output restituito è il seguente, dove le colonne T_PARAM e Q_PARAM si riferiscono rispettivamente ai parametri θ e q .

\$'D241 class'					
	PARENT_VAL	T	MM	T_PARAM	Q_PARAM
1:	1	0,1,1,0	0.0766020239383026,5.47467357864255
2:	2	0,1,1,0	0.151763189374394,7.1673998908178
3:	3	0,1,1,0	0.820078001204101,14.2055318026498
4:	4	0,1,1,0	0.971701277242522,16.0086685037965
5:	5	0,1,1,0	0.87644633641958,15.159103850713
6:	6	0,1,1,0	0.386087860803655,10.0714106889691

\$'D781 D981,D241'					
	PARENT_VAL	T	MM	T_PARAM	Q_PARAM
1:	0 0	0,1,1,0	1.81448614019771,9.02357221971381
2:	0 1	0,1,1,0	0.22590435022244,0.948588530702653
3:	1 0	0,1,1,0	0.731964335949756,2.58470366988323
4:	1 1	0,1,1,0	0.0930005463782311,0.301971442129515

\$'D782 class,D981'					
	PARENT_VAL	T	MM	T_PARAM	Q_PARAM
1:	1 0	0,1,1,0	0.0655421796732947,3.36724850293981
2:	1 1	0,1,1,0	0.0197691946524321,0.83306632489568
3:	2 0	0,1,1,0	0.183520094951706,4.982441163343
4:	2 1	0,1,1,0	0.0298500037312483,1.18543607112368
5:	3 0	0,1,1,0	1.45297330252178,9.04382156925825
6:	3 1	0,1,1,0	0.61806341802618,2.65225621556511
7:	4 0	0,1,1,0	1.8246217214331,9.63052480949547
8:	4 1	0,1,1,0	0.830199562217298,3.2070790708721
9:	5 0	0,1,1,0	2.05905337909037,9.58988956242524
10:	5 1	0,1,1,0	0.772843469472557,2.79867559056769
11:	6 0	0,1,1,0	0.91721347524916,6.46482076432075
12:	6 1	0,1,1,0	0.160812595892567,1.04989552010673

Per controllare il modo in cui i parametri vengono computati, cioè se essi vengono appresi tramite stima maximum-likelihood o tramite stima bayesiana, il pacchetto RCTBN fornisce 3 opzioni, presentate di seguito.

- L'opzione `ctbn.smoothing`, impostata al valore `TRUE` all'avvio di RCTBN. Ciò indica che, qualora non si modifichi tale opzione, RCTBN effettua la *regolarizzazione bayesiana* dei parametri (corrispondente all'implementazione dell'equazione 1.21 a pagina 20)
- Le opzioni `ctbn.imaginary.count` e `ctbn.imaginary.time` utili a specificare i valori dei *conteggi immaginari*. Esse sono impostate di default rispettivamente ai valori 1 e 0.005.

Perciò, l'esempio presentato dal sorgente A.9 effettua la *stima bayesiana* dei parametri, calcolando \hat{q} e $\hat{\theta}$.

A.1.4 Calcolo delle CIM

Riguardo il calcolo delle matrici di intensità condizionali (CIM), il pacchetto RCTBN offre una funzione, **cims**, che è il naturale proseguimento della funzione **params** illustrata nella sottosezione precedente.

Per calcolare le matrici di intensità condizionali è necessario disporre di un oggetto, che chiamiamo **pp**, contenente i parametri appresi tramite la funzione **params**. A tal fine è sufficiente modificare l'ultima linea del sorgente A.9 affinché il risultato della funzione **params** venga salvato in una variabile con tale nome.

La porzione di codice seguente illustra la chiamata di tale funzione.

```

1 options(width = 190)
2 options(ctbn.verbose = FALSE)
3 cims(pp)

```

Sorgente A.10: Calcolo delle CIM della CTBN d'esempio dal dataset #2.

Di seguito viene mostrato un estratto dell'output ottenuto dall'esecuzione del sorgente A.10.

```

$'D241|class'
      PARENT_VAL                      CIM
1:      1 -0.0766020239,5.47467357,0.0766020239,-5.47467357
2:      2 -0.151763189,7.1673998,0.151763189,-7.1673998
3:      3 -0.820078001,14.2055318,0.820078001,-14.2055318
4:      4 -0.971701277,16.0086685,0.971701277,-16.0086685
5:      5 -0.87644633,15.159103,0.87644633,-15.159103
6:      6 -0.386087860,10.0714106,0.386087860,-10.0714106

$'D781|D981,D241'
      PARENT_VAL                      CIM
1:      0|0 -1.81448614,9.02357221,1.81448614,-9.02357221
2:      0|1 -0.225904350,0.948588530,0.225904350,-0.948588530
3:      1|0 -0.731964335,2.58470366,0.731964335,-2.58470366
4:      1|1 -0.0930005463,0.301971442,0.0930005463,-0.301971442

$'D782|class,D981'
      PARENT_VAL                      CIM
1:      1|0 -0.0655421796,3.36724850,0.0655421796,-3.36724850
2:      1|1 -0.0197691946,0.83306632,0.0197691946,-0.83306632
3:      2|0 -0.183520094,4.982441,0.183520094,-4.982441
4:      2|1 -0.0298500037,1.18543607,0.0298500037,-1.18543607
5:      3|0 -1.45297330,9.04382156,1.45297330,-9.04382156
6:      3|1 -0.61806341,2.65225621,0.61806341,-2.65225621
7:      4|0 -1.8246217,9.63052480,1.8246217,-9.63052480
8:      4|1 -0.830199562,3.2070790,0.830199562,-3.2070790
9:      5|0 -2.05905337,9.58988956,2.05905337,-9.58988956
10:     5|1 -0.772843469,2.79867559,0.772843469,-2.79867559
11:     6|0 -0.91721347,6.46482076,0.91721347,-6.46482076
12:     6|1 -0.160812595,1.04989552,0.160812595,-1.04989552

```

A.1.5 Apprendimento di un CTBNC

In questa sottosezione si presenta l'utilizzo del pacchetto RCTBN per l'apprendimento di un classificatore CTBN (CTBNC).

A tale scopo si presenta una porzione di codice R che, generato in modo casuale un classificatore CTBN valido (si veda la definizione 2.1 nella sezione 2.1 a pagina 21) per il dataset #2, utilizza la funzione **learn_ctbnc** per apprenderne i parametri e la distribuzione a priori delle classi rispetto a un *training set* di tale dataset.

Prima di illustrare il sorgente necessario all'apprendimento del suddetto classificatore CTBN, si riporta la sua struttura tramite la tabella A.2, la quale illustra l'insieme di nodi genitori di ogni nodo.

Nodo	Nodi genitori	Nodo	Nodi genitori
D1091	class, D912	D5094	class, D1092, D781
D1092	class, D1091	D5095	class, D5097, D982, D1091
D1110	class, D5098	D5096	class, D212, D1687
D1290	class, D5095	D5097	class
D1480	class, D212, D1110	D5098	class, D341
D1681	class, D982	D5099	class
D1682	class, D5094, D191	D541	class, D421
D1687	class, D541, D1290, D1092	D641	class, D341, D911, D192
D191	class	D781	class
D192	class	D782	class
D211	class, D982, D1091	D891	class, D242
D212	class, D5099	D892	class, D241, D1682, D781
D241	class,	D911	class
D242	class, D5096, D911	D912	class, D5098, D892
D341	class, D242, D892, D1091	D981	class, D341, D982, D1110, D781
D421	class, D241, D982, D1091	D982	class, D5094

Tabella A.2: Struttura d'esempio di un classificatore CTBN (CTBNC) espressa tramite la corrispondenza fra i nodi e relativi insiemi di genitori.

Si osservi che tale struttura è perfettamente riproducibile su qualsiasi macchina esegua il codice utile a generarla. Ciò è possibile grazie all'utilizzo del comando **set.seed** che fissa il seme del generatore dei numeri casuali di R.

Si presenta ora sinteticamente la firma della funzione **learn_ctbnc**. Essa richiede in input due parametri principali:

- il *training set*
- una matrice di adiacenza, della stessa entità di quella richiesta dalla funzione **fsstats_graph**, che rappresenti una struttura valida di un classificatore CTBN.

Tuttavia, si noti che essa prevede ulteriori parametri formali che non vengono presentati poiché non ritenuti di primaria importanza.

In sostanza, la funzione **learn_ctbnc** implementa l'algoritmo 2.2 presentato a pagina 27.

Di seguito si illustra il codice sorgente che, utilizzando tale funzione (si veda la linea 17), apprende un CTBNC da un *training set* del dataset #2 (i.e., da una porzione di elementi di tale dataset). Si noti che prima d'eseguire tale operazione, da linea 4 a linea 13, viene generata una struttura valida e casuale per un classificatore CTBN; a linea 15 viene invece generato l'ipotetico *training set*.

```

1  options(width = 190)
2  options(ctbn.verbose = FALSE)
3  # generazione casuale di un classificatore CTBN
4  dt1 = monza100e[[1]]
5  timecol = getOption("ctbn.col.time")
6  classcol = getOption("ctbn.col.class")
7  cols = setdiff(colnames(dt1), c(timecol, classcol))
8  ncols = length(cols)
9  set.seed(ncols)
10 amat = matrix(rbinom(ncols^2, 1, .045), ncols, ncols)
11 dimnames(amat) = list(cols, cols)
12 diag(amat) = 0
13 amat = rbind(class = rep(1, ncols), amat)
14 # generazione di un training set
15 faketrainset = monza100e[seq(1, length(monza100e), 20)]
16 # apprendimento del classificatore CTBN sul training set
17 learn_ctbnc(faketrainset, amat)

```

Sorgente A.11: Apprendimento di un classificatore CTBN, la cui struttura viene generata in modo casuale, su un *training set* facente parte del dataset #2.

Affinché l'output ottenuto dall'esecuzione del sorgente A.11 sia leggibile, se ne riporta solamente la sua prima parte.

```

$priors
      1      2      3      4      5      6
0.1712963 0.1273148 0.1689815 0.2129630 0.1712963 0.1481481

$params
$params$'D1091|class,D912'
      PARENT_VAL T_PARAM Q_PARAM
1:      1|0 0,1,1,0 0.000274817667065954,1.59680638722439
2:      1|1 0,1,1,0      0.00307309352960158,200
3:      2|0 0,1,1,0 0.000793975840638463,1.3620885357555
4:      2|1 0,1,1,0      0.0270233752195596,200
5:      3|0 0,1,1,0      0.00011601863491314,200
6:      3|1 0,1,1,0 0.000486538690773037,1.5748031496051
7:      4|0 0,1,1,0 0.000178386617911624,0.88105726872209
8:      4|1 0,1,1,0 0.000539868865852468,1.75953079179186
9:      5|0 0,1,1,0 0.00027390744028475,2.13523131673482
10:     5|1 0,1,1,0 0.000236272284906581,1.65289256196757
11:     6|0 0,1,1,0 0.000461610987726345,1.1747430249653
12:     6|1 0,1,1,0      0.000451018286536413,200

$params$'D1092|class,D1091'
      PARENT_VAL T_PARAM Q_PARAM
1:      1|0 0,1,1,0 0.000134413073553186,1.99004975124378
2:      1|1 0,1,1,0      0.399201596806097,200
3:      2|0 0,1,1,0 0.000526967218028076,1.21028744326671
4:      2|1 0,1,1,0      0.22701475595925,200
5:      3|0 0,1,1,0 0.000207123602045138,0.00995999402400356
6:      3|1 0,1,1,0      0.524934383201699,200
7:      4|0 0,1,1,0 0.000134090168039564,2.29885057472033
8:      4|1 0,1,1,0      0.195886385896236,200
9:      5|0 0,1,1,0 0.000197580234863625,2.2988505746947
10:     5|1 0,1,1,0      0.498753117206983,200
11:     6|0 0,1,1,0 0.000275702225054808,2.48962655602261
12:     6|1 0,1,1,0      0.293685756241324,200

$params$'D1110|class,D5098'
      PARENT_VAL T_PARAM Q_PARAM
1:      1|0 0,1,1,0 0.000270069451735382,1.59680638722526
2:      1|1 0,1,1,0      0.0143874541399895,200
3:      2|0 0,1,1,0 0.000796072179864548,0.125772979771525
4:      2|1 0,1,1,0      0.0735023888276251,200
5:      3|0 0,1,1,0 0.00122835988160248,1.14707087259313
6:      3|1 0,1,1,0 0.00156213082455124,1.66389351081531
7:      4|0 0,1,1,0 0.000260353866466456,1.78253119429405
8:      4|1 0,1,1,0 0.00085643262979771,0.84447572132232
9:      5|0 0,1,1,0 0.000832569758938669,0.145855279150753
10:     5|1 0,1,1,0 0.000927771805246809,0.998751560547499
11:     6|0 0,1,1,0 0.000292531523928345,1.49625935162095
12:     6|1 0,1,1,0      0.00159082412643894,200

[...]
```

Si osservi che l'oggetto restituito dalla funzione **learn_ctbnc** è una

lista etichettata (i.e., un tipo del linguaggio R simile a una tabella hash) composta da 2 elementi, chiamati `priors` e `params`.

Ipotizzando di assegnare il risultato di `learn_ctbnc` a un oggetto chiamato `model`, è possibile accedere ai suoi elementi tramite l'operatore R per liste nominate `$`. Ad esempio, il comando `model$priors` permette di accedere alla distribuzione di probabilità a priori della variabile classe come visibile dall'output mostrato.

A.1.6 Classificazione di un CTNBC

Appreso un classificatore CTBN su un *training set* è chiaramente possibile, come discusso in questo lavoro di tesi, utilizzarlo per compiere la classificazione di una qualsiasi istanza del *test set*. A tale scopo si necessita di un algoritmo di classificazione che, dato in input il modello di un classificatore appreso e una istanza da classificare, inferisca la classe a cui è più probabile che tale istanza appartenga.

Lo scopo di questa sottosezione è illustrare il processo di classificazione tramite un modello di classificatore CTNB (CTNBC).

In questo lavoro di tesi, a pagina 31, si è presentato un algoritmo di inferenza (algoritmo 2.3) dedito alla classificazione tramite classificatori Continuous time Bayesian Network (CTBNC), classe di modelli di *classificazione supervisionata* cui appartengono i modelli di classificatori CTNB. Il pacchetto RCTBN fornisce tale procedura di inferenza esponendo la funzione `infer_ctbnc`.

Prima di presentare la suddetta funzione, si fa notare che RCTBN fornisce una funzione apposita per l'apprendimento dei classificatori CTNB (CTNBC): `learn_ctbnc`. Una chiamata a tale funzione è equivalente a una chiamata alla funzione `learn_ctbnc` con parametro `amat` posto al valore `NULL`. Tuttavia, essa è ben più performante di `learn_ctbnc` poiché attua delle ottimizzazioni di calcolo dovute a delle assunzioni che la struttura dei classificatori CTNB permettono di fare.

I parametri principali per cui la funzione `infer_ctbnc` richiede un input sono:

- il modello di un classificatore CTBN (CTBNC)
- un'istanza su cui effettuare la procedura di inferenza.

Anche in questo caso si tralascia la discussione dei restanti parametri poiché essi non sono di primaria importanza.

Di seguito si presenta il sorgente che, selezionata un'istanza dal dataset #2 (i.e., il primo elemento nel dataset), apprende un classificatore CTNB sui restanti elementi presenti del dataset (linea 15) e utilizza tale modello per classificare la suddetta istanza (linea 17).

```

1 options(width = 190)
2 options(ctbn.verbose = FALSE)
3 # selezione di una istanza (la prima) dal dataset
4 instance1 <- monza100e[[1]]
5 monza100e[[1]] <- NULL
6 instance1[, class := NULL]
7 # generazione di una struttura valida per un CTNBC
8 timecol = c(getOption("ctbn.col.time"))
9 cols <- setdiff(colnames(instance1), timecol)
10 ncols <- length(cols)
11 amat <- matrix(rep(0, ncols^2), ncols, ncols)
12 dimnames(amat) <- list(cols, cols)
13 amat <- rbind(class = rep(1, ncols), amat)
14 # apprendimento del CTNBC sui restanti elementi
15 m <- learn_ctbnc(monza100e, amat)
16 # inferenza della classe dell'istanza selezionata
17 results <- infer_ctbnc(m, instance1)

```

Sorgente A.12: Apprendimento di un classificatore CTNB (CTNBC) tramite la procedura generica e classificazione di un'istanza del dataset #2 tramite tale classificatore.

Come detto, utilizzando l'apposita funzione (si veda linea 8) di apprendimento dei classificatori CTNB (CTNBC) è possibile ridurre la quantità di linee di codice e di operazioni necessarie, come mostra il seguente sorgente.

```

1 options(width = 190)
2 options(ctbn.verbose = FALSE)
3 # selezione di una istanza (la prima) dal dataset
4 instance1 <- monza100e[[1]]
5 monza100e[[1]] <- NULL
6 instance1[, class := NULL]
7 # apprendimento del CTNBC sui restanti elementi
8 m <- learn_ctbnc(monza100e)
9 # inferenza della classe dell'istanza selezionata
10 results <- infer_ctbnc(m, instance1)

```

Sorgente A.13: Apprendimento di un classificatore CTNB (CTNBC) tramite la apposita funzione e classificazione di un'istanza del dataset #2 tramite tale classificatore.

L'output della funzione **infer_ctbnc** consiste in una lista etichettata composta due elementi: *logp*, contenente un vettore etichettato delle *log-probabilità* di ogni classe e *prob*, contenente un vettore etichettato delle *probabilità* assegnate ad ogni classe del modello CTNBC di riferimento.

```

$logp
      1
-863.3111
      2
-1308.3909
      3
-5862.1047
      4
-7147.8243
      5
-6520.6430
      6
-2781.4498

$prob
      1
1.000000e+00
      2
6.790600e-194
      3
0.000000e+00
      4
0.000000e+00
      5
0.000000e+00
      6
0.000000e+00

```

Si osservi che, questa modalità di output non è l'unica che la funzione **infer_ctbnc** possiede. Infatti, fornendole un ulteriore parametro booleano in input, essa restituisce il nome (i.e., classe 1) della classe assegnata all'istanza processata e il relativo valore di probabilità (100%, in questo caso). Nel caso si voglia operare manualmente tale operazione è sufficiente digitare il seguente comando:

```
max(results[["prob"]]).
```

A.1.7 Apprendimento strutturale

Lo scopo di questa sottosezione è presentare brevemente l'esecuzione del processo di `infer_ctbnc` tramite il pacchetto RCTBN.

Nello specifico, dato un ipotetico *training set*, si presenta l'utilizzo della funzione **greedy_search** per l'apprendimento del classificatore CTBN (CTBNC) che meglio rappresenta tali dati di addestramento, fissato il numero massimo di genitori di ogni nodo al valore 2. Tuttavia si osservi che la funzione in questione può apprendere una qualsiasi struttura CTBN. Tale funzione esegue l'algoritmo 3.1, presentato a pagina 38, per ogni nodo delle istanze del *training set*, utilizzando come punteggio la funzione $famscore_B$ presentata nella sezione 3.1 a

pagina 34 (si veda l'equazione 3.9).

La funzione **greedy_search** richiede che le siano fornite in input le seguenti informazioni, sotto forma di parametri:

- un *training set*
- *method*, l'euristica da utilizzare per la ricerca della struttura
- *fun*, la funzione di punteggio (i. e., *score bayesiano*) da utilizzare come criterio di selezione della struttura
- un punto di partenza, chiamato *start*, da cui iniziare la ricerca, inteso come una matrice, eventualmente vuota, che rappresenti la struttura iniziale
- la lista *excludelist*, eventualmente vuota, dei nodi da non sottoporre alla ricerca
- la lista *whitelist*, eventualmente vuota, dei nodi che devono sempre far parte dell'insieme dei nodi genitori di ogni nodo su cui si effettua la ricerca
- la lista *blacklist*, eventualmente vuota, dei nodi che non devono mai far parte dell'insieme dei nodi genitori di ogni nodo su cui si effettua la ricerca
- *k*, la cardinalità massima degli insiemi dei nodi genitori.

Tuttavia, l'unico parametro obbligatorio è il *training set*, sul quale la procedura di ricerca euristica computa lo *score bayesiano* al variare dei nodi e del relativo insieme di genitori. I restanti parametri sono tutti dotati di un parametro di default.

Il sorgente che segue illustra come espletare il compito prefissatoci.

```

1 options(width = 190)
2 options(ctbn.verbose = FALSE)
3 # selezione di un training set
4 fakets = monza100e[seq(1, length(monza100e), 20)]
5 # creazione della struttura di un CTNBC
6 timecol = c(getOption("ctbn.col.time"))
7 classcol = c(getOption("ctbn.col.class"))
8 nodes <- colnames(monza100e[[1]])[- timecol]
9 nn <- length(nodes)
10 amat <- matrix(0, nn, nn, dimnames = list(nodes, nodes))
11 amat[classcol, ] <- 1
12 # ricerca della struttura del CTBNC partendo dal CTNBC
13 greedy_search(fakets, start = amat, whitelist = classcol,
    k = 2)

```

Sorgente A.14: Apprendimento della struttura di un CTBN (CTBNC) da una porzione del dataset #2.

L'output che tale funzione restituisce è composto dalla struttura che meglio rappresenta i dati di addestramento e il relativo modello CTBN computato in fase di ricerca.

A.2 GENERAZIONE DI DATASET


In questa sezione si illustra il processo di generazione di un dataset per le CTBN a partire da un modello di traffico TSIS, utilizzando l'estensione a tempo d'esecuzione sviluppata in questo lavoro di tesi, Sensors DLL.

A.2.1 Sensors DLL

Al fine di generare un dataset per le CTBN che rappresenti l'onda quadra dei sensori a spira magnetica posti su una rete stradale TSIS è necessario installare Sensors DLL in TShell. Il processo di installazione, come già ampiamente specificato, consiste nel collegamento delle funzioni della succitata RTE ai punti di chiamata predefiniti del modulo CORSIM. L'intero processo di installazione, propedeutico alla generazione dei dataset, è stato illustrato nella sottosezione 5.2.4 a pagina 59.

Si mostra il processo di generazione del dataset #2 (sezione 6.2 a pagina 78).

Come primo passo è chiaramente necessario aprire TShell e caricare il modello di traffico per i giorni lavorativi relativo a *Viale Cesare Battisti*. Di seguito si elencano quindi i passi necessari alla generazione del dataset #2:

- nella barra degli strumenti di TShell si clicchi l'icona relativa a Sensors DLL
- si clicchi sul pulsante relativo alle "proprietà d'esecuzione"  e successivamente sulla scheda "proprietà d'esecuzione multipla"
- si selezioni il file RNS fornito in allegato a questo lavoro di tesi (e illustrato nel sorgente 6.1 a pagina 86) come illustrato dalla figura A.1 nella pagina successiva
- si esegua la simulazione cliccando sull'apposito pulsante
- conclusa la simulazione, Sensors DLL salva, nella cartella dove il file TRF simulato risiede, i file CSV di output (riguardo il cui formato si è discusso nella sottosezione 5.3.2 a pagina 69).

Si osservi che per ogni esecuzione della simulazione viene generato il relativo file di output. Il nome di tali file CSV è una concatenazione delle seguenti informazioni: nome del file TRF (seguito dal

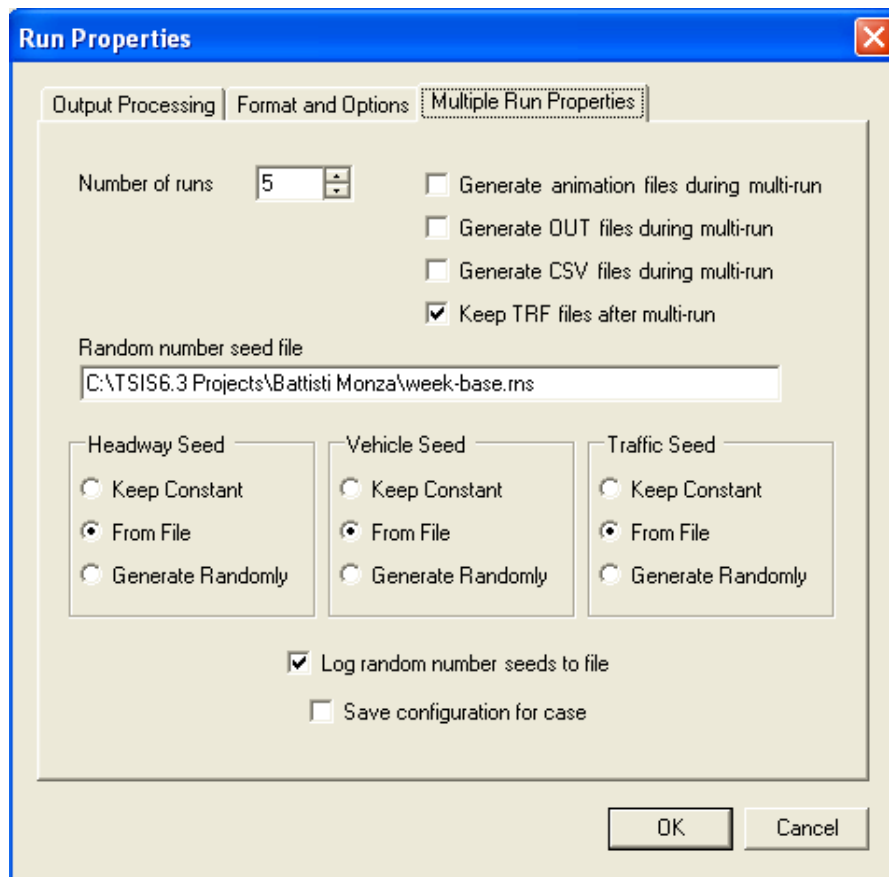


Figura A.1: Selezione del file RNS per l'impostazione dell'esecuzione multipla del dataset #2.

numero dell'esecuzione in caso di esecuzione multipla), data e ora dell'esecuzione, suffisso sensors.

A.2.2 Applicativi di supporto

Ottenuti i file di output di Sensors DLL è necessario trasformarli in un dataset utile al framework delle CTBN. A tale scopo sono stati sviluppati degli applicativi, brevemente discussi nella sezione 5.4 a pagina 71.

Il primo passo necessario consiste nella sostituzione dei *time period* con le rispettive classi (si veda a tal riguardo la tabella 6.7 a pagina 84). A tale scopo è stato sviluppato un programma in linguaggio R con interfaccia bash chiamato *sensors2dataset*.

Riguardo al modello in questione è quindi necessario eseguire il succitato programma nel seguente modo, ripetendo tale passo per tutti i 5 file di output generati (si ricorda che il modello per i giorni lavorativi viene eseguito infatti 5 volte, una per ogni giorno lavorati-

vo):

```
./sensors2dataset -v -t -r -b 2,3,5,8,10,12 -i ~/week-base-1.csv
```

Sorgente A.15: Esecuzione del programma bash per la sostituzione dei *time period* con le relative classi nel file CSV relativo al dataset #2.

Di seguito si spiegano brevemente le opzioni utilizzate:

- l'opzione *i* serve a specificare il percorso del file di input, cioè del file CSV generato da Sensors DLL
- l'opzione *t* comunica al programma che il file di input è dotato di intestazione
- l'opzione *r* comunica al programma di rimuovere la colonna relativa ai *time period*
- l'opzione *b* serve a definire i *time period* a partire dai quali una nuova classe deve essere creata e associata
- l'opzione *v* serve per eseguire il programma in modo verboso.

Si osservi tuttavia che `sensors2dataset` implementa molte altre opzioni la cui descrizione è stata tuttavia tralasciata.

Infine, si noti che `sensors2dataset` non sovrascrive il file di input. Infatti, esso genera automaticamente un nuovo file con le modifiche apportate, concatenando al nome del file di input il suffisso `dataset`.

A questo punto è possibile scegliere una granularità temporale qualsiasi con cui tagliare il file CSV, tenendo presente che Sensors DLL utilizza per il monitoraggio del passaggio dei veicoli sui sensori la massima granularità temporale ottenibile da CORSIM, cioè 0.1 secondi.

Di conseguenza, ipotizzando di voler tagliare il corrente file CSV in più file da 100 secondi, si utilizza il programma bash appositamente sviluppato a tale fine, `splitfile`.

```
./splitfile.sh -i ~/week-base-1.csv_dataset.csv -n 1000
```

Sorgente A.16: Esecuzione del programma bash finalizzato alla generazione del dataset #2. Il file di input viene tagliato in più file, ognuno dei quali rappresentante un intervallo temporale di 100 secondi.

L'opzione *n* serve a specificare il numero di righe da cui ogni file deve essere composto. In questo caso quindi, poiché ogni riga rappresenta un intervallo temporale di 0.1 secondi e desideriamo ottenere dei file che rappresentino intervalli temporali minori o uguali di 100 secondi, impostiamo il suo valore pari a 1000.

Si osservi che `splitfile` salva i file generati in una nuova cartella la cui posizione corrisponde a quella del file di input.

Infine, è possibile ottimizzare il dataset appena generato rimuovendo le righe che corrispondono a intervalli temporali durante i quali non è avvenuta alcuna transizione di stato per nessun sensore. A tale scopo viene fornito un programma bash chiamato `killconsdup`. Esso permette l'ottimizzazione di un singolo file CSV secondo la logica descritta oppure l'ottimizzazione di tutti i file CSV presenti nella cartella di input fornita. Di seguito si illustra la seconda tipologia d'utilizzo di `killconsdup`.

```
./killconsdup.sh -i ~/1/
```

Sorgente A.17: Esecuzione del programma `bash` finalizzato alla rimozione delle osservazioni duplicate, cioè delle righe relative agli intervalli temporali durante i quali tutti i sensori sono rimasti nel loro precedente stato.

Si osservi che `killconsdup` non considera la prima riga dei file che processa, assumendo che essa contenga l'intestazione del file CSV. Inoltre esso non considera la prima e la seconda colonna dei file di input assumendo esse si riferiscano rispettivamente alla colonna del tempo e della classe. Infine, affinché tale programma funzioni correttamente è necessario che il separatore dei file CSV di input sia la virgola.

Eseguito il precedente comando, i file che compongono il dataset saranno sovrascritti con la loro rispettiva versione ottimizzata.

ACRONIMI

API	Application Programming Interface	56
BN	Bayesian Network	1
BNC	Bayesian Network Classifier	22
CIM	Conditional Intensity Matrix	10
CORSIM	Corridor microscopic simulation program	51
COM	Component Object Model	57
CPD	Conditional Probability Distribution	2
CPT	Conditional Probability Table	2
CRAN	The Comprehensive R Archive Network	41
CSV	Comma Separated Values	66
CTBN	Continuos time Bayesian Network	1
CTBNC	Continuos time Bayesian Network Classifier	21
CTNB	Continuos time Naive Bayes	21
CTNBC	Continuos time Naive Bayes Classifier	22
CTTANB	Continuos time Tree Augumented Naive Bayes	21
CTTANBC	Continuos time Tree Augumented Naive Bayes Classifier	22
DAG	Directed acyclic graph	1
DBN	Dynamic Bayesian Networks	33
DLL	Dynamic-link library	51
EM	Expectation Maximization	4
F_1	F-measure	92
FN	False Negative	91
FNR	False Negative Rate	91
FP	False Positive	91
FPR	False Positive Rate	91
FRESIM	Freeway Simulator	52
FWHA	Federal Highway Administration	50
GUI	Graphical User Interface	50
IDE	Integrated Development Environment	50
IC	Inductive Causation	6
IM	Intensity Matrix	7
MAP	Maximum a posteriori	28
MCMC	Markov Chain Monte Carlo	5
MLE	Maximum Likelihood Estimation	18
NETSIM	Network Simulator	51
NPV	Negative Predictive Value	92
PPV	Positive Predictive Value	91
PV	Process Variable	11

RNS	Random Number Seed	86
ROC	Receiver Operating Characteristic.....	95
RT	Record Type.....	51
RTE	Run-Time Extension.....	51
TAN	Tree Augmented Naive Bayes	22
TNO	TRAFED Native Object	51
TN	True Negative	90
TNR	True Negative Rate	91
TP	True Positive	90
TPR	True Positive Rate	91
TRAF	Traffic File.....	51
TRAFED	TRAF Editor.....	52
TRAFVU	TRAF Visualization Utility.....	52
TRF	Traffic File.....	50
TShell	TSIS Shell	52
TSIS	Traffic Software Integrated System.....	49
VBScript	Microsoft's Visual Basic Scripting Edition.....	52

INDICE ANALITICO

A

accuracy *vedi* metrica
 aciclicità 33, 36, 37
 apprendimento
 classificatore 24, 26, 116
 CTBNC 27, 116, 119
 CTNBC 25, 117, 120
 parametri *vedi* stima
 strutturale 33, 37, 39, 121

B

bayes, regola di 28
 bayesian network *vedi* rete bayesiana

C

chain rule 2
 classificazione 21
 CTBNC 28, 88
 CTNBC 119, 120
 multi-classe 88, 90, 96
 profili di traffico 72, 77, 86, 88
 supervisionata 28, 89, 93, 119
 complessità
 NP-completo 33, 36
 polinomiale 32, 33, 37
 conteggi immaginari 19, 20, 89, 114
 CORSIM 51, 54
 modello di simulazione 72, 78
 time interval 54–56
 time period 54, 56
 time step 54, 56
 cross-tabulazione 89
 cross-validazione 72, 88, 89
 fold 88, 89, 92
 k-fold 72

D

dataset 72, 77, 78, 86, 88
 generazione 66, 123, 124
 time interval 75, 84
 time period 75, 84, 124, 125
 dati
 addestramento, di .. *vedi* training set
 completi 13, 21, 28, 33
 definizione
 CTBN 12
 CTBNC 22
 CTNBC 23
 distribuzione
 Dirichlet, di 18, 36
 esponenziale 8, 16–18
 Gamma 18
 multinomiale 17, 18

F

flusso di evidenze 12, 28
 segmenti temporali 12
 F-measure *vedi* metrica
 fold *vedi* cross-validazione

G

grafo 33, 34, 36
 penalità, del 34

H

hill climbing 37–39

I

indipendenza condizionale 3, 4, 23, 28
 inferenza 21, 31, 120
 maximum a posteriori 28
 intervallo temporale *vedi* CORSIM

K

k-fold *vedi* cross-validazione
 k-learn 36

L

likelihood
 CTBN 15
 marginale 34–36
 temporale 28, 29
 transizione, di una 14

M

macro-average 88–90
 matrice
 confusione, di 89, 90, 96
 intensità, di 7, 10, 17
 metrica
 accuracy 92
 fall-out 91
 false alarm error 95
 false alarm rate 91
 false negative 91
 false negative rate 91
 false positive 91
 false positive rate 91, 95
 F-measure 92
 negative predictive value 92
 positive predictive value 91
 precision 91
 recall 91, 95
 ROC 95, 96
 sensitivity 91, 95
 specificity 91
 true negative 90
 true negative rate 91
 true positive 90
 true positive rate 91, 95
 micro-average 88–90, 92, 96

N

NP-completo *vedi* complessità

O

onda quadra 49, 69, 123
 ottimizzazione 33, 34, 36
 overfitting 88

P

pacchetto R *vedi* RCTBN
 parameter
 independence 18, 19, 34, 35
 modularity 34
 parametrizzazione
 mista 8, 9
 pura 8, 9
 passo temporale *vedi* CORSIM
 periodo temporale *vedi* CORSIM
 polinomiale *vedi* complessità
 precision *vedi* metrica
 priori coniugata 18, 19, 35
 processo di Markov 6, 7
 condizionale 10
 omogeneo 7
 proprietà di Markov 6
 pseudo-conteggi *vedi* conteggi immaginari
 punteggio *vedi* score bayesiano

R

R (linguaggio) 40
 closure 40
 estensione 41
 programmazione funzionale 40
 scoping lessicale 40
 RCTBN 43
 recall *vedi* metrica
 regolarizzazione bayesiana 16, 114
 rete bayesiana 2, 4, 11
 ricerca della struttura 36, 122

ricerca euristica 33, 37, 122
 risultati
 accuracy 93, 94
 ROC 97–103
 sperimentazione 92, 95
 ROC *vedi* metrica

S

score bayesiano 33, 34, 36, 39, 122
 sensori 49, 53, 55, 61, 66, 68
 smoothing *vedi* regolarizzazione bayesiana
 sperimentazione 88, 89
 statistiche sufficienti 13–15, 18, 19, 110, 111
 stima
 bayesiana 18, 113, 114
 maximum a posteriori *vedi* inferenza
 maximum-likelihood 16, 18, 113
 parametri 16–18, 113
 structure modularity 34, 36

T

test set 88, 90, 119
 traiettoria 11, 13, 77, 87
 training set 26, 33, 34, 88, 89, 116, 119, 122
 transizione
 intensità di 8, 10
 modello di 7–9, 13

V

validation set *vedi* test set

BIBLIOGRAFIA

Chickering, David Maxwell

- 2013 «A Transformational Characterization of Equivalent Bayesian Network Structures», *CoRR*, p. 87-98. (Citato a p. 6.)

Codecasa, D e F Stella

- 2013 «Conditional Log-Likelihood for Continuous Time Bayesian Network Classifiers», in *New Frontiers in Mining Complex Patterns*, Springer. (Citato a p. 97.)

Kuhn, Max, Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt e Tony Cooper

- 2013 *caret: Classification and Regression Training*, R package version 5.17-7. (Citato a p. 92.)

R Core Team

- 2013 *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. (Citato alle p. ii, 40.)

Wickham, Hadley e Winston Chang

- 2013 *devtools: Tools to make developing R code easier*, R package version 1.3.99. (Citato alle p. 42, 106.)

Stella, F e Y Amer

- 2012 «Continuous time Bayesian network classifiers.» *Journal of biomedical informatics*, 45, 6 (dic. 2012), p. 1108-19, ISSN: 1532-0480, DOI: 10.1016/j.jbi.2012.07.002. (Citato alle p. xi, 10, 12, 22, 25, 28, 31, 33.)

Angulo, Eusebio, Francisco P Romero, Ricardo García, Jesús Serrano-Guerrero e José A Olivas

- 2011 «An adaptive approach to enhanced traffic signal optimization by using soft-computing techniques», *Expert Systems with Applications*, 38, 3, p. 2235-2247. (Citato a p. 48.)

Korb, K.B. e A.E. Nicholson

- 2011 *Bayesian Artificial Intelligence*, Chapman & Hall / CRC Computer Science and Data Analysis, CRC Press, ISBN: 9781439815915. (Citato alle p. 1, 3.)

Wickham, Hadley

- 2009 *ggplot2: elegant graphics for data analysis*, Springer New York, ISBN: 978-0-387-98140-6. (Citato a p. ii.)

Gershenson, Carlos

- 2008 «Self-Organizing traffic lights», *Arxiv preprint nlin/0411066*.
(Citato a p. 48.)

Manning, Christopher D., Prabhakar Raghavan e Hinrich Schütze

- 2008 *Introduction to Information Retrieval*, Cambridge University Press, New York, NY, USA, ISBN: 0521865719, 9780521865715.
(Citato a p. 88.)

Jensen, F.V. e T.D. Nielsen

- 2007 *Bayesian Networks and Decision Graphs*, Information Science and Statistics, Springer, ISBN: 9780387682815. (Citato a p. 5.)

Mantecca, P, M Gualtieri, M Andrioletti, R Bacchetta, C Vismara, G Vailati e MC Camatini

- 2007 «Tire debris organic extract affects *Xenopus* development», *Environment international*, 33, 5, p. 642-648, ISSN: 0160-4120.
(Citato a p. 48.)

Nodelman, Uri D.

- 2007 *Continuous Time Bayesian Networks*, tesi di dott., Stanford University. (Citato alle p. xi, 9, 11, 12, 16, 18.)

Fawcett, Tom

- 2006 «An introduction to ROC analysis», *Pattern Recognition Letters*, 27, 8 (giu. 2006), p. 861-874, ISSN: 01678655, DOI: 10.1016/j.patrec.2005.10.010. (Citato a p. 96.)

Felici, G, G Rinaldi, A Sforza e K Truemper

- 2006 «A logic programming based approach for on-line traffic control», *Transportation Research Part C: Emerging Technologies*, 14, 3 (giu. 2006), p. 175-189, ISSN: 0968-090X, DOI: 10.1016/j.trc.2006.05.007. (Citato a p. 48.)

Oliveira, S. e D. Stewart

- 2006 *Writing Scientific Software: A Guide for Good Style*, Cambridge University Press, ISBN: 9781139458627. (Citato a p. 40.)

Gualtieri, M, L Rigamonti, V Galeotti e MC Camatini

- 2005 «Toxicity of tire debris extracts on human lung cell line A549», *Toxicology in Vitro*, 19, 7 (ott. 2005), p. 1001-1008, ISSN: 0887-2333, DOI: 10.1016/j.tiv.2005.06.038. (Citato a p. 48.)

Papageorgiou, M., C. Diakaki, V. Dinopoulou, A. Kotsialos e Y. Wang

- 2005 «Review of road traffic control strategies», *Proceedings of the IEEE*, 91, 12 (dic. 2005), p. 2043-2067, ISSN: 0018-9219, DOI: 10.1109/JPROC.2003.819606. (Citato a p. 48.)

Certet

- 2004 «Sei felice di essere un'automobilista». (Citato a p. 48.)

- Chickering, David Maxwell, David Heckerman e Christopher Meek
 2004 «Large-sample learning of Bayesian networks is NP-hard», *Journal of Machine Learning Research*, 5, p. 1287-1330. (Citato a p. 33.)
- Russell, Stuart J. e Peter Norvig
 2003 *Artificial Intelligence: A Modern Approach*, Pearson Education, ISBN: 0137903952. (Citato alle p. 2, 3, 37, 38.)
- Nodelman, Uri D., CR Shelton e Daphne Koller
 2002 «Learning Continuous Time Bayesian networks», *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, X, arXiv: /arxiv.org/abs/1212.2498 [http:]. (Citato alle p. 15, 33, 37.)
- Steck, Harald and Jaakkola, Tommi S
 2002 «On the Dirichlet Prior and Bayesian Regularization», *Advances in Neural Information Processing Systems*, September, p. 713-720. (Citato a p. 19.)
- Hastie, Trevor, Robert Tibshirani e Jerome Friedman
 2001 *The Elements of Statistical Learning*, Springer Series in Statistics, Springer New York Inc., New York, NY, USA. (Citato a p. 88.)
- Provost, Foster e Tom Fawcett
 2001 «Robust classification for imprecise environments», *Machine Learning*, 42, 3, p. 203-231. (Citato a p. 95.)
- Lando, David
 1998 «On Cox Processes and Credit Risky Securities», 120, p. 99-120. (Citato a p. x.)
- MacKay, D. J. C.
 1998 «Introduction to Monte Carlo methods», in *Proceedings of the NATO Advanced Study Institute on Learning in graphical models*, Kluwer Academic Publishers, Norwell, MA, USA, p. 175-204. (Citato a p. 5.)
- Norris, James R.
 1998 *Markov chains*, Cambridge series in statistical and probabilistic mathematics, Cambridge University Press, p. I-XVI, 1-237, ISBN: 978-0-521-48181-6. (Citato a p. 7.)
- Park, Byungkyu e CJ Messer
 1998 «A Genetic Algorithm-Based Signal Optimization Program for Oversaturated Intersections», *Texas Transportation Institute*, p. 1-8. (Citato a p. 48.)

Friedman, N, D Geiger e M Goldszmidt

- 1997 «Bayesian network classifiers», *Machine learning*, 163, p. 131-163. (Citato alle p. 21, 22, 29.)

Provost, Foster e Tom Fawcett

- 1997 «Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions», in *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, AAAI Press, p. 43-48. (Citato a p. 95.)

Duffie, Darrell, Mark Schroder e Costis Skiadas

- 1996 «Recursive valuation of defaultable securities and the timing of resolution of uncertainty», *The Annals of Applied Probability*, 6, 4 (nov. 1996), p. 1075-1090, DOI: 10.1214/aoap/1035463324. (Citato a p. x.)

Gilks, WR, S Richardson e DJ Spiegelhalter

- 1996 «Markov chain Monte Carlo in practice». (Citato a p. 5.)

Heckerman, David

- 1996 «A Tutorial on Learning With Bayesian Networks», *Innovations in Bayesian Networks*, Studies in Computational Intelligence, 1995, November, a cura di Dawn E Holmes e Lakhmi C Jain, p. 33-82, ISSN: 1860949X, DOI: 10.1007/978-3-540-85066-3. (Citato alle p. 5, 18, 19.)

Thorpe, T.L. e C.W. Anderson

- 1996 «Traffic light control using sarsa with three state representations», in *IBM Corporation*, Citeseer. (Citato a p. 48.)

Heckerman, David, Dan Geiger e David M. Chickering

- 1995 «Learning Bayesian networks: The combination of knowledge and statistical data», *Machine Learning*, 20, 3 (set. 1995), p. 197-243, ISSN: 0885-6125, DOI: 10.1007/BF00994016. (Citato alle p. 6, 19.)

Chickering, David Maxwell

- 1994 *Learning Bayesian networks is NP-hard*, rapp. tecn., Microsoft Research. (Citato alle p. 33, 36.)

Langley, Pat, Wayne Iba e Kevin Thompson

- 1992 «An Analysis of Bayesian Classifiers», in *AAAI*, p. 223-228, ISBN: 0-262-51063-4. (Citato alle p. 21, 23.)

Verma, Thomas S. e Judea Pearl

- 1991 «Equivalence and synthesis of causal models», in *Uncertainty in Artificial Intelligence*, North Holland, p. 255-268. (Citato a p. 6.)

Shachter, Ross D. e Mark A. Peot

- 1990 «Simulation Approaches to General Probabilistic Inference on Belief Networks», in *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, UAI '89, North-Holland Publishing Co., Amsterdam, The Netherlands, p. 221-234, ISBN: 0-444-88738-5. (Citato a p. 5.)

Dean, Thomas e Keiji Kanazawa

- 1989 «A model for reasoning about persistence and causation», *Computational Intelligence*, 5, 2, p. 142-150, ISSN: 1467-8640, DOI: 10.1111/j.1467-8640.1989.tb00324.x. (Citato a p. xi.)

Pearl, Judea

- 1988 *Probabilistic reasoning in intelligent systems: networks of plausible inference*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, ISBN: 0-934613-73-7. (Citato alle p. x, 5.)

Geman, Stuart e Donald Geman

- 1984 «Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images», *IEEE Trans. Pattern Anal. Mach. Intell.*, 6, 6, p. 721-741, ISSN: 0162-8828, DOI: 10.1109/TPAMI.1984.4767596. (Citato a p. 5.)

Loève, Michel

- 1978 *Probability theory. II Edition*. Fourth, Graduate Texts in Mathematics, Vol. 46, Springer-Verlag, New York, p. xvi+413, ISBN: 0-387-90262-7. (Citato alle p. 6, 7.)

Dempster, A P, N M Laird e D B Rubin

- 1977 «Maximum likelihood from incomplete data via the EM algorithm», *Journal of the Royal Statistical Society Series B Methodological*, Series B, 39, 1, p. 1-38, ISSN: 00359246, DOI: 10.2307/2984875. (Citato a p. 4.)

Duda, R. O. e P. E. Hart

- 1973 *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York. (Citato a p. 21.)

Gihman, Iosif I. e Anatolij V. Skorohod

- 1973 *The theory of stochastic processes II*, New York: Springer-Verlag. (Citato a p. 9.)

DICHIARAZIONE

Dichiaro che il presente elaborato e i contributi ad esso correlati sono frutto del mio impegno e studio. Dichiaro inoltre di aver esplicitamente citato tutte le fonti di informazioni utilizzate.

Milano, settembre 2013

Leonardo Di Donato