UNIVERSITÀ DEGLI STUDI DI PALERMO

SCUOLA POLITECNICA
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Innovazione Industriale e Digitale
Ingegneria Chimica, Gestionale, Informatica e Meccanica

ESTRAZIONE DI CONOSCENZA MEDIANTE ALGORITMI DI APPRENDIMENTO ATTIVO PER L'INFERENZA REGOLARE

Tesi di laurea di:

Nicola Ciaco

Matricola:

0582164

Relatore:

Chiar.mo Prof. Salvatore Gaglio

Correlatori:

Ing. Marco Ortolani

Ing. Pietro Cottone

Controrelatore:

Chiar.mo Prof. Giuseppe Lo Re

Anno Accademico
2017/18

UNIVERSITÀ DEGLI STUDI DI PALERMO
SCUOLA POLITECNICA
Laurea Magistrale in Ingegneria Informatica

ESTRAZIONE DI CONOSCENZA MEDIANTE ALGORITMI DI APPRENDIMENTO ATTIVO PER L'INFERENZA REGOLARE

TESI DI LAUREA DI:
Dott. Nicola Ciaco

RELATORE:
Chiar.mo Prof. Salvatore Gaglio
CORRELATORI:
Ing. M. Ortolani, Ing. P. Cottone

Anno Accademico 2017/18

Sommario

Un sistema intelligente è caratterizzato dalla capacità di apprendere in modo automatico, la quale a sua volta presuppone la capacità sia di rappresentare la conoscenza nota a priori sia di *inferirne* di nuova. Allo stato dell'arte attuale, le tecniche più efficaci per l'*estrazione di conoscenza* sono state sviluppate nell'ambito di quella che è nota come Statistical Learning Theory (SLT). Per quanto tali metodi forniscano risultati apprezzabili in termini di accuratezza, essi presentano tuttavia anche dei non trascurabili svantaggi, quali ad esempio il fatto che il loro funzionamento si basa sull'inferire dei parametri ottimali per un modello che è visto necessariamente come una *black-box*, rendendo ardua o impossibilitando di fatto l'*interpretazione* dei campioni a partire dai quali il modello è stato costruito.
La presente tesi si colloca invece nel solco di un framework teorico alternativo, noto come Algorithmic Learning Theory (ALT), secondo cui i dati non sono considerati come campioni casuali da mappare in uno spazio vettoriale, bensì come specifiche istanze del modello nascosto che è oggetto di inferenza. In particolare, lo studio si è concentrato su una tecnica di estrazione di conoscenza che va sotto il nome di *Inferenza Grammaticale* (GI), un processo di apprendimento che si basa sull'*induzione* nel contesto dei linguaggi formali e del relativo formalismo grammaticale. In tale contesto, l'obiettivo è selezionare il migliore modello rappresentato mediante il tipico riconoscitore di una grammatica formale, ovvero l'automa a stati finiti, consistente con i campioni iniziali, che a loro volta sono interpretabili come stringhe generabili dalla grammatica. Nella fattispecie, la classe dei linguaggi oggetto di studio per la presente tesi è quella dei linguaggi regolari e si parla pertanto più specificamente di *Inferenza Induttiva Regolare* (IIR), i cui limiti teorici sono stati messi in luce dal lavoro di Gold [22]. Gli algoritmi della letteratura iniziale sull'argomento riconducevano l'apprendimento ad una ricerca euristica in un spazio rappresentato come un grafo contenente gli automi consistenti con i campioni forniti in cui il modello, inizialmente iperspecializzato, viene progressivamente generalizzato. Tale approccio

va sotto il nome di *passive learning* ed è caratterizzato da un'inevitabile esplosione combinatoria che ne rende la complessità ingestibile a meno di non sacrificare le garanzie teoriche di terminazione e ottimalità.
Nella presente tesi si è quindi scelto di seguire un approccio proposto nella letteratura più recente, duale rispetto al precedente, secondo cui un modello, inizialmente molto generale e quindi poco accurato, viene progressivamente specializzato per rappresentare con precisione i dati forniti. Questo paradigma, noto come *active learning*, presuppone l'esistenza di un *oracolo* che guida l'apprendimento rispondendo ad alcuni tipi di query sottopostegli attivamente dal sistema che cerca di inferire il modello. La base teorica del paradigma è stata fornita dagli studi di Angluin [2] che stabiliscono che l'oracolo debba appartenere alla classe dei cosiddetti Minimally Adequate Teacher (MAT) che, per garantire l'abilità di fornire risposte utili ai principali tipi di query considerate, richiede la conoscenza preliminare dell'automa oggetto d'inferenza. Questo è un requisito molto forte e fin troppo stringente, che limita l'utilizzo in contesti reali dell'inferenza induttiva regolare declinata nell'accezione dell'*active learning*.
Il cuore del lavoro svolto in questa tesi ha quindi riguardato la sostituzione di un tradizionale oracolo con una sua approssimazione mediante un classificatore statistico costruito a partire da esempi positivi e negativi del linguaggio target nell'ottica di permetterne un utilizzo nelle applicazioni reali. L'obiettivo è quello di tracciare un parallelo tra i modelli ottenibili seguendo il paradigma della teoria dell'apprendimento statistico e quelli formulabili algoritmicamente e di verificare sperimentalmente se l'uso combinato possa condurre al superamento dei limiti di entrambi. In altre parole, si vuole costruire per i campioni dati un modello strutturale, da usare in luogo dell'oracolo in modo da superarne i requisiti stringenti. Il modello statistico prescelto per affrontare il delineato problema di classificazione binaria è stato quello delle Support Vector Machines (SVM) perchè rappresentano un modello molto potente. Inoltre malgrado siano state proposte in letteratura delle loro applicazioni nel contesto dei linguaggi formali esse non sono state contestualizzate nell'ambito dell'inferenza induttiva regolare mediante *active learning* oppure erano limitate all'apprendimento di specifiche classi di linguaggi sottoinsiemi di quelli regolari.
Al fine di valutare la correttezza della soluzione ottenuta, durante il lavoro di tesi si è progettata un'implementazione in linguaggio C++11, integrando il codice nella preesistente libreria di passive learning Gi-learning [15]. L'implementazione è stata testata sperimentalmente inizialmente su una nota classe di automi che rappresentano linguaggi semplici, ovvero i Tomita [19, 51], ottenendo ottimi risultati paragonabili al caso ideale. Infine le performance e la precisione dei modelli ottenuti dall'approccio qui proposto sono state vagliate su data set di automi estratti casualmente, la cui complessità è paragonabile ai sistemi da apprendere nei casi pratici. I risultati ottenuti rivelano che all'aumentare della complessità del target si ha un progressivo decadimento del grado di approssimazione ed è quindi possibile concludere che un livello di fiducia alto sull'utilizzo dell'algoritmo è ottenibile solo fino ad un certo livello di complessità del target.

Indice

Elenco delle figure

Elenco delle tabelle

Acronimi

ObP	Observation Pack
ObPA	Observation Pack Approssimato
IIR	Inferenza Induttiva Regolare
GI	Inferenza Grammaticale
P	enunciati premessa
BK	conoscenze di background
H	ipotesi induttiva
MQ	Membership Query
EQ	Equivalence Query
$\mathcal{L}$	linguaggio target
MAT	Minimally Adeguate Teacher
FSA	Finite State Automata
DFA	Deterministic Finite Automata
FSM	Finite State Machine
NFA	Non-Deterministic Finite Automata
DT	Discrimination Tree
LCA	least common ancestor
SVM	Support Vector Machine
VC	Vapnik Chervonenkis
SRM	Structural Risk Minimization
OHE	One Hot Encoding

Introduzione

Un sistema intelligente è caratterizzato dalla capacità di apprendere in modo automatico, la quale a sua volta presuppone la capacità sia di rappresentare la conoscenza nota a priori sia di inferirne di nuova. Le tecniche di *estrazione di conoscenza* oggetto di studio in questa sede riguardano l'apprendimento induttivo di linguaggi formali. L'induzione è un procedimento che elabora informazioni parziali riguardanti le proprietà di un insieme, fornendo una generalizzazione, cioè estendendo l'insieme su cui le proprietà valgono. Questo procedimento in generale non è logicamente giustificato, e di conseguenza non è certo che l'informazione fornita in uscita sia vera. All'interno degli studi sull'induzione nasce l'inferenza grammaticale, che si occupa di studiare le modalità con cui può essere individuato un linguaggio formale, quando è conosciuto un insieme di stringhe che appartengono ad un linguaggio sconosciuto e, eventualmente, un insieme di stringhe che non appartengono al linguaggio. Supponiamo che una sorgente di informazioni fornisca delle stringhe binarie:

$$01, 0101, 010101, \ldots$$

Ci si può domandare se c'è una regola formale con la quale la sorgente genera le stringhe, se eventualmente questa regola sia individuabile guardando solo l'insieme delle stringhe, e quali siano i fattori che influenzano l'identificabilità della regola. L'inferenza grammaticale cerca di trovare risposte a queste domande, nell'ipotesi che esiste una regola con cui le stringhe sono state create, e che si tratti di una grammatica generativa di Chomsky.
Nella fattispecie, la classe dei linguaggi oggetto di studio per la presente tesi è quella dei linguaggi regolari e si parla pertanto più specificamente di *Inferenza Induttiva Regolare* (IIR), i cui limiti teorici sono stati messi in luce dal lavoro di Gold [22]. Gli algoritmi della letteratura iniziale sull'argomento riconducevano l'apprendimento ad una ricerca euristica in un spazio rappresentato come un grafo contenente gli automi consistenti con i campioni forniti in cui il modello,inizialmente iperspecializzato, viene progressivamente generalizzato. Tale approccio va sotto il nome di *passive learning* ed è caratterizzato da un'inevitabile esplosione combinatoria che ne rende la complessità ingestibile a meno di non sacrificare le garanzie teoriche di terminazione e ottimalità o di stringenti requisiti sui campioni in ingresso.
Nella presente tesi si è quindi scelto di seguire un approccio proposto nella letteratura più recente, duale rispetto al precedente, secondo cui un modello, inizialmente molto generale e quindi poco accurato, viene progressivamente specializzato per rappresentare con precisione i dati forniti. Questo paradigma, noto come *active learning*,

presuppone l'esistenza di un *oracolo* che guida l'apprendimento rispondendo ad alcuni tipi di query sottopostegli attivamente dal sistema che cerca di inferire il modello. La base teorica del paradigma è stata fornita dagli studi di Angluin [2] che stabiliscono che l'oracolo debba appartenere alla classe dei cosiddetti Minimally Adequate Teacher (MAT) che, per garantire l'abilità di fornire risposte utili ai principali tipi di query considerate, richiede la conoscenza preliminare dell'automa oggetto d'inferenza. Questo è un requisito molto forte e fin troppo stringente, che limita l'utilizzo in contesti reali dell'inferenza induttiva regolare declinata nell'accezione dell'*active learning*.

Il lavoro in questa tesi nasce con l'intenzione di indagare lo scenario nel quale il tradizionale oracolo è sostituito con una sua approssimazione mediante un classificatore statistico costruito a partire da esempi positivi e negativi del linguaggio target nell'ottica di permetterne un utilizzo nelle applicazioni reali. L'obiettivo è quello di tracciare un parallelo tra i modelli ottenibili seguendo il paradigma della teoria dell'apprendimento statistico, Statistical Learning Theory (SLT), e quelli formulabili algoritmicamente nell'ambito dell'inferenza grammaticale con *active learning* e di verificare sperimentalmente se l'uso combinato possa condurre al superamento dei limiti di entrambi ossia rispettivamente un modello poco significativo per i dati di partenza e il requisito di un oracolo onnisciente. In altre parole, si vuole costruire per i campioni dati un modello strutturale, da usare in luogo dell'oracolo in modo da superarne i requisiti stringenti. Il modello statistico prescelto per affrontare il delineato problema di classificazione binaria è stato quello delle Support Vector Machines (SVM) perchè rappresentano un modello molto potente. Inoltre malgrado siano state proposte in letteratura delle loro applicazioni nel contesto dei linguaggi formali esse non sono state contestualizzate nell'ambito dell'inferenza induttiva regolare mediante *active learning* oppure erano limitate all'apprendimento di specifiche classi di linguaggi sottoinsiemi di quelli regolari.

Al fine di valutare la correttezza della soluzione ottenuta, durante il lavoro di tesi si è progettata un'implementazione in linguaggio C++11, integrando il codice nella preesistente libreria di passive learning Gi-learning [15]. L'implementazione è stata testata sperimentalmente inizialmente su una nota classe di automi che rappresentano linguaggi semplici, ovvero i Tomita [19, 51], ottenendo ottimi risultati paragonabili al caso ideale. Infine le performance e la precisione dei modelli ottenuti dall'approccio qui proposto sono state vagliate su data set di automi estratti casualmente, la cui complessità è paragonabile ai sistemi da apprendere nei casi pratici. I risultati ottenuti rivelano che all'aumentare della complessità del target si ha un progressivo decadimento del grado di approssimazione ed è quindi possibile concludere che un livello di fiducia alto sull'utilizzo dell'algoritmo è ottenibile solo fino ad un certo livello di complessità del target.

Il lavoro qui esposto si divide in cinque parti. Nel primo capitolo si parlerà dell'inferenza induttiva, e riferendosi alla classificazione proposta in [34], si inquadrerà questo meccanismo nel complesso meccanismo dell'apprendimento. Inoltre, dopo avere messo a confronto l'induzione con la deduzione e l'abduzione verranno passati in rassegna le peculiarità del processo induttivo. Nel secondo capitolo si definirà l'in-

ferenza induttiva grammaticale e saranno scandagliati brevemente i risultati teorici e i limiti dell'**Inferenza Induttiva Regolare (*IIR*)**. Inoltre si descriverà brevemente il *Passive Learning* tecnica duale all' *Active Learning*. Infine sarà presentato brevemente il paradigma dell'*Active Learning* e sarà introdotto e approfondito L* [2] che può essere considerato il capostipite degli algoritmi di *active learning*. Nel terzo capitolo verrà esposto in maniera dettagliata la ratio che muove uno dei più efficienti algoritmi di *active learning*:l'**Observation Pack (*ObP*)** . Inoltre verranno riportate le scelte discostanti dal riferimento principale dell'algoritmo [26] e le motivazioni. Il quarto capitolo è il cuore della tesi dove sarà descritto in dettaglio il lavoro svolto per costruire l'oracolo approssimato e il suo utilizzo concreto all'interno del prescelto algoritmo di *active learning ObP*. Nel quinto capitolo si descriveranno criticamente i risultati sui test eseguiti sul programma al fine di valutarne le prestazioni sia in termini di accuracy del classificatore ottenuto che in termini di similarità tra il **Deterministic Finite Automata (*DFA*)** inferito e il *DFA* target. Inoltre si esaminerà come e quando è possibile variare alcuni parametri dell'algoritmo al fine di migliorarne le prestazioni e in quali contesti (complessità del linguaggio target,numero di esempi del linguaggio da apprendere) è possibile avere un livello di fiducia alto sull'utilizzo dell'algoritmo.

Capitolo 1

Inferenza Induttiva

Il metodo induttivo o induzione è un procedimento logico per cui dalla constatazione di fatti particolari si risale ad affermazioni o formulazioni generali. Si suole quindi indicare con il termine induzione il passaggio dal *particolare* al *generale*. Con il termine **Inferenza Induttiva** si indica un processo che partendo da degli esempi specifici congettura delle regole generali. L'inferenza induttiva gioca un ruolo fondamentale nel più vasto scenario dell'apprendimento ed in ogni contesto che si prefigge la scoperta di strutture universali. L'applicazione di questo metodo scientifico ,intrinseco agli esseri intelligenti, all'interno delle macchine ha portato alla nascita di diversi filoni di ricerca. Uno dei più rilevanti tratta dei complessi meccanismi che consentono ad un uomo di imparare un linguaggio.

1.1 Apprendimento

1.1.1 Definire l'apprendimento

Insieme alla capacità di pianificare cioè elaborare piani, la capacità di apprendere è ritenuta uno dei segni distintivi di un sistema intelligente. Un sistema si può considerare autonomo fintantochè le sue azioni sono determinate dalle esperienze pregresse e dalle percezioni correnti, invece che dal suo progettista (si pensi agli agenti stimolo-risposta). Senza la capacità di apprendimento un sistema non sarà in grado di operare con successo in qualsiasi ambiente ma solo in quelli previsti dal suo progettista. Nonostante la grande importanza dell'apprendimento, una conclusione largamente diffusa è che non sia possibile darne una definizione precisa: si procede invece analizzando gli effetti che l'apprendimento ha eventualmente prodotto.
Due concetti rivestono un ruolo importante nell'apprendimento:

- Il miglioramento delle capacità del sistema che apprende
- L'acquisizione di nuova conoscenza

Simon [47] approfondisce cosa significa migliorare le capacità di un sistema mediante l'apprendimento: *L'apprendimento identifica delle modifiche in un sistema che sono*

adattive, nel senso che consentono al sistema di svolgere lo stesso goal, o goals analoghi, in maniera migliore nel futuro. E' doveroso però osservare che esistono sistemi che migliorano nel tempo senza essere soggetti a nessun processo di apprendimento e che esistono degli scenari in cui non è facile calare la definizione data da Simon.

L'altro fattore che contraddistingue l'apprendimento è l'acquisizione di nuove conoscenze che presuppone a monte una rappresentazione della conoscenza in maniera descrittiva o iconica per potere rappresentare la nuova conoscenza avvenuta mediante l'apprendimento. In quest'ottica *l'apprendimento è creare e modificare rappresentazioni di ciò che è stato sperimentato.* Laddove con sperimentare si intende sia l'informazione proveniente dall'apparato sensoriale dell'agente sia ciò che il sistema recepisce mediante processi interni (ad esempio ripetere più volte una frase tra sè e sè ci consente d'impararla nonostante non è avvenuto nessuno stimolo dai nostri sensi).Da questo punto di vista apprendere significa costruire una rappresentazione della realtà anzichè un miglioramento delle capacità dell'agente, aspetto quest ultimo considerato come una conseguenza.

E' possibile quindi constatare il grado di apprendimento di un sistema misurando i miglioramenti nel portare a compimento un certo job dopo l'apprendimento. (Miglioramenti che implicitamente sono considerati una conseguenza della rappresentazione interna dell'agente della realtà esterna). Si fa presente che in questa caratterizzazione si assume che l'agente abbia un obbiettivo e che tale obbiettivo sia conosciuto all'osservatore che valuta l'apprendimento.

1.1.2 Una possibile schematizzazione

Esistono diverse possibilità di classificare i fattori che influenzano l'apprendimento. Michalski [36] propone una divisione basata sulle caratteristiche del sistema che apprende.
Michalski esegue una prima distinzione in base alla quantità di conoscenze iniziali di cui il sistema è dotato. Ai due estremi della classificazione troviamo le reti neurali artificiali e i sistemi esperti. Nel contesto dei sistemi dotati di scarse conoscenze iniziali le reti neurali sono uno strumento largamente usato: le connessioni dei neuroni che costituiscono il sistema, sono determinate in maniera essenziale dagli esempi presentati e solo marginalmente dai valori iniziali (di solito casuali) delle connessioni. Nella progettazione di un sistema esperto invece una grande quantità di informazione viene fornita al sistema. Un altro approccio, suggerito da Michalski, propone di suddividere i sistemi artificiali che apprendono in base al tipo di manipolazione eseguita dal **learner**(sistema che apprende) sull'informazione proveniente dall'esterno. In ogni processo di apprendimento il *learner* trasforma l'informazione fornita da un *teacher*, o più in generale da un **informant** sorgente di informazione, in una nuova forma che viene poi memorizzata per usi futuri. Questa trasformazione dell'informazione, che fa uso anche delle conoscenze già possedute dal *learner* viene chiamata **inferenza**. Il tipo di trasformazione eseguita determina la strategia di ap-

prendimento di cui il sistema fa uso. Si possono distinguere,seguendo esattamente Michalski in [34] , cinque diverse strategie:

- Apprendimento per *imitazione*
- Apprendimento per *istruzioni*
- Apprendimento per *deduzione*
- Apprendimento per *analogia*
- Apprendimento per *induzione*

Queste strategie sono elencate in ordine crescente di complessità del learning e decrescente di difficoltà del teaching. Michalski attua questa classificazione restrigendo l'ambito di applicabilità al ***concept learning*** una branca del machine learning. Un sistema intelligente deve essere abile nel classificare alcuni oggetti,eventi o comportamenti come equivalenti per raggiungere un determinato goal. Detto in maniera succinta un sistema intelligente deve essere capace di individuare i *concetti*.

Definizione 1.1. Un **concetto** è una classe di equivalenza per cui esiste un metodo operativo che permette di discriminare le istanze come appartenenti o non appartenti al concetto.

Dove le istanze sono le singole entità della classe di equivalenza (del concetto),cioè gli esempi presentati dall'informant. Un *learner* impara un concetto quando ,tramite una procedura effettiva, è in grado di distinguere le entità che appartengono al concetto da quelle che non appartengono.Adesso si prenderanno brevemente in esame le cinque strategie di apprendimento contestualizzandole nel *concept learning*:

Apprendimento per imitazione Questo è il caso estremo in cui il *learner* non deve effettuare alcuna inferenza sulle informazioni che gli provengono dall' *informant*. Infatti questo metodo è anche detto da Michalski impianto diretto di conoscenza (meglio conosciuto ancora come rote learning) proprio perchè il *learner* non deve fare altro che inidicizzare l'informazione per poterla poi recuperare. In questo caso l'*informant* fornirà una descrizione del concetto in input al *learner*. Questa strategia è usata quando uno specifico algoritmo per riconoscere un concetto è implementato su un calcolatore(oppure vi è a disposizione un database di fatti che permette di riconoscere il concetto) . Ad esempio nei primi programmi che giocavano a scacchi si salvavano i risultati dell'esplorazione del grafo di ricerca (in alcuni punti che rappresentano possibili situazioni in una partita) in un albero di gioco in modo che quando una situazione già memorizzata si fosse presentata in una partita reale si potesse risparmiare spazio e tempo di esecuzione.

Apprendimento per istruzioni In questo caso il *learner* acquisisce un concetto da un *teacher*, o da un'altra forma organizzata di informazione, come una publicazione o un libro, ma non copia direttamente in memoria l'informazione

acquisita. Nell'apprendimento per istruzioni le trasformazioni sull'informazione eseguite dal *learner* sono la selezione e la riformulazione a livello sintattico. Il processo di apprendimento può consistere nel selezionare i fatti più importanti e poi trasformarli in una forma più appropriata. Un programma che costruisce una database di fatti e regole sulla base di una conversazione con un utente è un esempio di sistema che apprende per istruzioni.

Apprendimento per deduzione Il *learner* acquisisce un concetto deducendolo dalle conoscenze fornite dall' *informant* insieme a quelle che il sistema già possedeva. Inoltre, questa strategia include ogni processo nel quale la conoscenza appresa è il risultato di una trasformazione che preserva la verità delle informazioni generate dall' *informant* e di ciò che viene inferito. All'interno dell' apprendimento dei concetti, l'apprendimento per deduzione tramite il processo inferenziale trasforma una definizione non adoperabile per discriminare il concetto, in una definizione operativa adatta a questo scopo. Ad esempio dal fatto che una giarra sia un oggetto stabile e trasportabile, si può dedurre che la brocca ha un fondo piatto e un manico.

Apprendimento per analogia Il *learner* acquisisce un nuovo concetto modificando la definizione di un concetto simile già noto. Anzichè formulare una descrizione del concetto ex novo, il sistema adatta una descrizione esistente modificandola appropriatamente per il nuovo scopo. Ad esempio se già si conosce una regola che definisce il concetto di arancia, per imparare il concetto di mandarino si possono notificare le differenze e le similitudini tra arancia e mandarino. L'apprendimento per analogia può essere visto come un incrocio tra l'apprendimento deduttivo e quello induttivo. Attraverso l'inferenza induttiva si possono determinare le caratteristiche generali o le trasformazioni che unificano i concetti confrontati. Poi, attraverso un'inferenza deduttiva si possono derivare le proprietà caratterizzanti possedute dal concetto che deve essere appreso

Apprendimento per induzione In questa strategia il *learner* acquisisce un concetto effettuando inferenza induttiva sui fatti forniti dall' *informant* o in base a delle osservazioni su tali fatti. Esistono due differenti forme di questa strategia:

1. **Apprendimento da esempi**
   Al *learner* partendo da degli esempi specifici (istanze del concetto) ed eventualmente dei controntroesempi induce una descrizione del concetto catturando la struttura generale . Si assume che il concetto esiste e che esiste anche un metodo effettivo per testare l'appartenenza di un'istanza ad un concetto. Il compito del *learner* è determinare una descrizione del concetto analizzando le singole istanze del concetto. Questa strategia è utilizzata nell' *IIR*
2. **Apprendimento per osservazione e scoperta**
   Il *learner* analizza le entità in input e determina che qualche sottoinsieme

di queste entità può essere raggruppato in un singolo concetto.Poichè , diversamente dall'apprendimento da esempi, non c'è un *teacher* che conosce in anticipo i concetti questa strategia è talvota menzionata come *unsupervisioned learning*. Un esempio è il *clustering* cioè il partizionamento di una collezione di oggetti all'interno di gruppi o classi che avviene in maniera gerarchica; l'eredità gioca un ruolo importante: se un entità è riconosciuta appartenere ad un determinato concetto erediterà da esso e dai concetti più in alto nella gerarchia tutte le proprietà . Ad esempio se si apprende che Freddy è un elefante, allora si può,senza vedere Freddy, dire che ha la proboscide e tutte le proprietà degli elefanti e più in generale anche degli erbivori e dei mammiferi.

1.2 Induzione

L'induzione è quel procedimento logico che permette di passare dal particolare all'universale. Questa definizione è troppo semplice e non spiega tutte le componenti in gioco nel processo induttivo. A tal fine si seguirà ancora [34]. Qui le principali componenti induttive sono più precisamente distinte e specificate nel contesto della manipolazione simbolica:

Dati i seguenti elementi di partenza

- Gli **enunciati premessa (*P*)** che comprendono fatti, generalizzazioni intermedie, specifiche osservazioni che forniscono informazioni su oggetti,fenomeni,processi eccetera. Costituiscono l'input del processo inferenziale.
- Le **conoscenze di background (*BK*)** che contengono concetti generali o specifici del dominio, che permettono di interpretare gli enunciati premessa e le regole rilevanti per l'inferenza. Ed includono concetti precedentemente imparati, vincoli del dominio, relazioni di causalità,goals dell'inferenza,e metodi per valutare la bontà di una congettura in base al goal (criterio di preferenza)

si determina alla fine dell'inferenza induttiva

- Una **ipotesi induttiva (*H*)** che implica gli enunciati premessa nel contesto delle conoscenze di background ed è l'ipotesi migliore in base al criterio di preferenza.

Si dice che H implica fortemente P nel contesto di BK se usando BK e l'inferenza deduttiva P è una conseguenza logica di H. Schematizzando si ottiene l'equazione

$$H \vee BK \implies P \tag{1.1}$$

che è vera con tutte le possibili *interpretazioni*. In contrasto H implica debolmente gli enunciati premessa nel contesto delle BK se usando le BK e l'inferenza deduttiva P è solo una conseguenza plausibile ma non una conseguenza logica. Michalski fornisce un esempio di quanto appena detto:

Enunciati premessa
Aristotele era greco
Socrate era greco
Platone era greco

Conoscenze di background
Socrate, Aristotele e Platone erano filosofi
Sono vissuti nell'antichità
I greci sono persone
I filosofi sono persone
Criterio di preferenza: Si preferiscono le ipotesi più corte e più utili per decidere la nazionalità dei filosofi

Le **ipotesi induttive** sono:

1. I filosofi che hanno vissuto nell'antichità erano greci
2. Tutti i filosofi sono greci
3. Tutte le persone sono greche

L'ipotesi da preferire,in base al criterio di preferenza, è la 2, perchè è più breve della 1 e più specifica della 3; consente a differenza della 1 di determinare la nazionalità di tutti i filosofi. Si può di mostrare che questa ipotesi induttiva è un'ipotesi forte,poichè P risulta essere una conseguenza logica di H e delle BK.
Supponiamo di aggiungere alla premessa gli enunciati Locke era inglese e Hume era inglese e di modificare le BK aggiungendo il fatto che sia Locke che Hume erano filosofi. In questo caso una ipotesi induttiva forte potrebbe essere che tutti i filosofi erano greci, con l'eccezione di Locke e Hume. Mentre una ipotesi induttiva debole potrebbe essere che alcuni filosofi erano greci. Dal fatto che Platone era un filosofo e sulla base di questa nuova ipotesi debole non consegue che Platone era greco, consegue solo che c'è la possibilità che Platone fosse greco.
Senza pretesa di esaustività si accenna ad altri tipi di inferenza presenti nel pensiero logico con lo scopo di fare emergere le peculiarità dell'inferenza induttiva. L'inferenza sta alla base dell'apprendimento. Seguendo [35] l'apprendimento si può sintetizzare in $apprendimento = inferenza + memorizzazione$ (definizione leggermente diversa da quella data in 1.1.1) quindi una completa teoria dell'apprendimento deve includere una completa teoria dell'inferenza [35]. Viene innanzitutto generalizzata l'equazione (1.1) valida solo per l'induzione ottenendo:

$$\mathrm{Q} \vee BK \models \mathrm{C} \tag{1.2}$$

detta **equazione fondamentale** per l'inferenza. Poi Michalsky effettua una prima suddivisione tra i metodi d'inferenza:

1. conclusivi
2. contingenti

Nel secondo caso nell' equazione (1.2) C è solo una plausibile,parziale,probabilistica conseguenza logica delle BK e di Q. Nell'inferenza conclusiva invece la conseguenza logica è garantita. Le proprietà dell'inferenza induttiva sono confrontate con quelle dell'inferenza deduttiva ed emerge che sono duali come si vede in figura 1.1 La

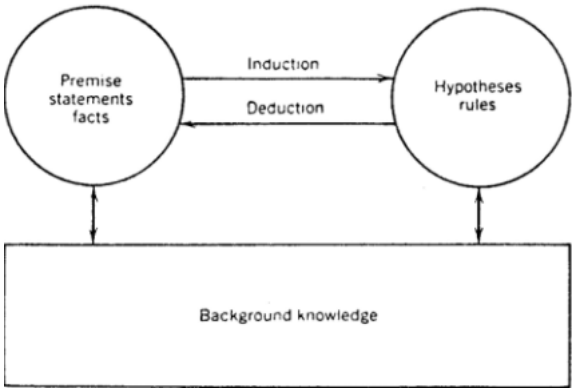


Figura 1.1: Relazione tra deduzione e induzione

relazione logica (1.2) succintamente cattura la relazione tra i due tipi d'inferenza. L'inferenza deduttiva deriva logicamente C date BK e Q. L'inferenza induttiva invece va ad ipotizzare Q date BK e C. La deduzione è il processo di determinare una conseguenza logica a partire da una conoscenza data, ed è *truth-preserving* (C deve essere vero se BK e Q sono vere). In contrasto l'induzione sta ipotizzando un Q che insieme con BK implica l'input C, ed è *false-preserving* (se C è falso allora anche Q deve essere falso. Cioè se l'input in ingresso è falso anche le ipotesi congetturate saranno false). La deduzione contingente invece suona come debole in quanto è debolmente *true-preserving* cioè produce conseguenze che possono essere vere in alcune situazioni e false in altre. Analogamente l'induzione contingente è debolmente *false-preserving*.
In [35] l'inferenza viene considerata come un processo che prende in Input un enunciato e tramite le BK già possedute (ed eventualmente la conoscenza dei criteri di preferenza per il goal che permette di restringere tutte le possibili ipotesi tra le quali scegliere) fornisce un enunciato in Output .In quest'ottica le proprietà dell'induzione sono messe in risalto dal confronto con quelle della deduzione e dell'abduzione fornendo per ciascuno di essi degli esempi chiarificatori.

1. DEDUZIONE tabella 1.1 Riferendosi all'equazione fondamentale (1.2) l'Input sta per Q e l'Output sta per C. L'Input consiste in un enunciato che afferma l'appartenenza di un elemento a ad X. Le BK sono costituite da un enunciato

che assegna una certa proprietà q agli elementi dell'insieme X, e da una regola logica detta *regola di specializzazione universale*. L'inferenza consiste solo nell'applicazione di tale regola che essendo una tautologia [1] fa si che il risultato dell'inferenza deduttiva assuma pure valore logico vero. Questo è un esempio di inferenza deduttiva conclusiva dato che l'Output è sempre una conseguenza logica dell'Input e delle *BK*

Input	$a \in X$	a è un elemento di X
BK	$\forall x \in X, q(x)$	Tutti gli elementi di X hanno la proprietà q.
	$\forall x \in X, q(x) \implies (a \in X \implies q(a))$	Se tutti gli elementi di X hanno la proprietà q, allora ogni elemento di X, e quindi anche a, deve avere la proprietà q
Output	$q(a)$	a ha la proprietà q

Tabella 1.1: Deduzione

2. INDUZIONE tabella 1.2 Riferendosi all'equazione fondamentale (1.2) l'Input è la conseguenza C e l'Output è Q (l'ipotesi).Si può dimostrare che l'Input è conseguenza logica dell'Output (l'ipotesi) e delle *BK* quindi dato che l'equazione fondamentale 1.2 è rispettata l'inferenza è conclusiva (forte). Inoltre il processo inferenziale è *false-preserving*, se l'Input fosse falso (a non ha la proprietà q) allora l'Output deve essere pure falso. Da rimarcare che l'Output dell'inferenza induttiva (sia che sia conclusiva che contingente) non ha un valore di verità sempre vero ma può essere vero o falso (anche se l'Input e le *BK* sono vere) da cui deriva il termine ipotesi per connotare l'Output.Essa si basa sull'assunzione che determinate regolarità osservate in un fenomeno continueranno a manifestarsi nella stessa forma anche in futuro e quindi generalizza ciò che è vero per alcune istanze ad un insieme più grande. Invece nell'inferenza deduttiva conclusiva è garantito logicamente che l'Output assuma valore di verità vero se l'Input e le *BK* sono pure vere perchè ciò che è vero in generale resta vero in un caso specifico contemplato dalla regola generale.

   Nell'esempio riportato le conoscenze di *BK* sono le stesse della deduzione. Tuttavia l'Output(l'ipotesi) è ottenuto tracciando all'indietro la *regola di specializzazione universale*. Quindi l'inferenza consiste nel supporre l'implicazione presente nella regola di specializzazione valida anche nel verso opposto, perciò si dice che l'induzione è una regola d'inferenza all'indietro a la deduzione una

[1] E' un enunciato che ha sempre valore logico vero

regola d'inferenza in avanti.

Input	$q(a)$	a ha la proprietà q
BK	$a \in X$	a è un elemento dell'insieme X.
	$\forall x \in X, q(x) \implies (a \in X \implies q(a))$	Se tutti gli elementi di X hanno la proprietà q, allora ogni elemento di X, e quindi anche a, deve avere la proprietà q
Output	$\forall x \in X, q(x)$	Tutti gli elementi di X hanno la proprietà q

Tabella 1.2: Induzione

3. ABDUZIONE tabella 1.3 In riferimento all'equazione (1.2) l'Output è Q e l'Input è C. Si può dimostrare che l'Input è conseguenza logica dell'Output (l'ipotesi) e delle BK quindi dato che l'equazione fondamentale 1.2 è rispettata l'inferenza è conclusiva (forte).Come nel caso dell'induzione l'inferenza abduttiva conclusiva è *false-preserving*. Come nell'induzione l'Output è solo un' ipotesi e quindi il suo valore di verità è incerto e c'è solo una probabilità che sia vero. L'abduzione, come l'induzione, non contiene in sé la sua validità logica e deve essere confermata per via empirica. Nell'abduzione come nell'induzione la regola implicativa di specializzazione universale viene tracciata all'indietro. Tuttavia c'è un'importante differenza infatti nell'induzione la regola implicativa nelle BK costituisce una tautologia mentre nel caso dell'abduzione rappresenta una verità solo nel dominio di conoscenza e non una verità universale.

   Nell'esempio specifico si assume che un elemento a gode della proprietà q. Le BK consistono in un unico enunciato, che esprime il fatto che tutti gli elementi di un certo insieme X hanno la proprietà q. L'inferenza abduttiva produce in Output un enunciato che asserisce l'appartenenza di a ad X. Intuitivamente tutti gli elementi che appartengono ad un insieme X hanno una proprietà; dall'input si ha che un elemento a ha quella proprietà; siccome tutti gli elementi appartenenti all'insieme X possiedono quella stessa proprietà si suppone che a appartiene all'insieme X

Input	$q(a)$	a ha la proprietà q
BK	$\forall x, x \in X \implies q(x)$	Se x è un elemento di X allora x ha la proprietà q.
Output	$a \in X$	a è un elemento di X

Tabella 1.3: Abduzione

1.2.1 Metodologia di ricerca induttiva

Si introduce brevemente,seguendo ancora [34] il *learning da esempi* induttivo di un concetto come un problema di ricerca in uno spazio. L'algoritmo inferenziale induttivo riceve in ingresso degli esempi (ed eventualmente anche controesempi) ,di membri del concetto target(specifiche istanze) , sottoinsieme dello **spazio delle istanze** che costituisce l'insieme di tutte le possibili istanze osservabili. Lo **spazio dei concetti** costituisce invece l'insieme di tutti i possibili concetti (tutte le possibili soluzioni). I concetti quasi sempre necessitano di una descrizione, un linguaggio che formalmente consente di definire operativamente un concetto e per questo si parla in maniera interscambiabile di **spazio delle descrizioni**. Un concetto è consistente se accetta alcuni esempi positivi e rifiuta tutti quelli negativi; è completo invece quando accetta tutti gli esempi positivi. La macchina inferenziale induttiva ha lo scopo di selezionare un'ipotesi dallo **spazio delle ipotesi** che sia consistente e completa con gli esempi visti. Lo spazio delle ipotesi è un sottoinsieme dello spazio dei concetti. All'aumentare degli esempi visti lo spazio delle ipotesi si riduce, tuttavia le ipotesi valide possono comunque essere numerose e spesso è necessario utilizzare dei criteri di preferenza per scegliere l'ipotesi corrente. E' necessario anche definire dei criteri di terminazione per sancire la ricerca conclusa. In sintesi il *concept learning induttivo* può essere descritto come una ricerca euristica nello spazio delle descrizioni della migliore ipotesi tra tutte quelle consistenti e complete rispetto agli esempi forniti.

Capitolo 2

Inferenza Grammaticale

L' **Inferenza Grammaticale (*GI*)** è considerata una branca del *machine learning* sebbene il primo algoritmo di *GI* sia più datato della nascita del concetto di machine learning. Più in dettaglio *GI* è un'istanza dell'inferenza induttiva e in particolare dell'apprendimento per induzione da esempi introdotti in 1.1.2, e può quindi essere descritto come il problema di congetturare un linguaggio target sconosciuto a partire da un *training set* che di solito comprende un insieme finito di stringhe S^+ appartenenti ad un **linguaggio target ($\mathcal{L}$)** ,definite su un alfabeto Σ, dette positive ed eventualmente anche un insieme finito di stringhe negative S^- che non appartengono ad $\mathcal{L}$.
Sebbene il nome *inferenza grammaticale* potrebbe suggerire che l'*output* di un algoritmo d'*inferenza grammaticale* sia una grammatica non è questo il caso — qualsiasi altra descrizione di un linguaggio come un automa, un'espressione regolare, ecc. può essere pure usata. In questa tesi l'attenzione è volta agli algoritmi di *GI* il cui *linguaggio target* appartiene alla classe più semplice della gerarchia di Chomsky: i linguaggi regolari. In quest ultimo caso si suole parlare di **inferenza grammaticale regolare** e talvolta di **inferenza induttiva regolare** (*IIR*).

2.1 Connotare l'IIR

Una caratterizzazione del *learning da esempi* induttivo ,seguendo [34], è stata già fornita in 1.2.1. In questa sede è opportuno puntualizzare ed approfondire alcuni dei punti di cui si compone e soprattutto contestualizzarla all'*IIR* dato che la classificazione in 1.2.1 fa riferimento all'apprendimento generico di un concetto e nell'*IIR* si specializza l'oggetto dell'inferenza che diventa un linguaggio regolare. I punti di cui si compone sono[1]:

[1]Da adesso in poi e per tutto il resto della tesi ci si discosterà dalla definizione di consistenza di Mychalski esposta in 1.2.1 a meno che non venga espressamente indicato. Con consistenza si intenderà che, dato un *DFA* A e un insieme di istanze $S = S^+ \cup S^- \ \forall x \in S^+ : \lambda^A(x) = 1 \wedge \forall x \in S^- : \lambda^A(x) = 0$

Spazio delle istanze L'algoritmo inferenziale induttivo riceve in ingresso degli esempi (ed eventualmente anche controesempi),di membri del concetto target(specifiche istanze) , sottoinsieme dello spazio delle istanze che costituisce l'insieme di tutte le possibili istanze osservabili. Nell'*IIR* le istanze sono delle stringhe definite sull'alfabeto Σ.

Spazio dei concetti E' l'insieme di tutte le possibili soluzioni e nell'*IIR* rappresenta l'insieme dei linguaggi regolari.

Spazio delle descrizioni Costituisce lo spazio che contiene le descrizioni operative degli elementi dello spazio dei concetti. Nell'*IIR* si possono usare le espressioni regolari, gli *NFA* o i *DFA* anche se di norma si usano i *DFA* perchè è garantita l'esistenza di un *DFA* canonico.

Spazio delle ipotesi Lo spazio delle ipotesi contiene quei concetti consistenti con gli esempi osservati ed è quindi un sottoinsieme dello spazio dei concetti.

Criteri di successo Sono i criteri che permettono di decretare che il processo d'inferenza è concluso. Esistono varie tecniche e varianti ma i due metodi principali sono:

- *Identification in the limit*
- *PAC-learning*

Definizione (Identification in the limit)**.** Un linguaggio $\mathcal{L}$ è identificato al limite da un algoritmo di inferenza se ad un certo punto del processo inferenziale l' ipotesi intermedia H generata è una descrizione di $\mathcal{L}$ e da quel punto in poi H non muta al variare delle istanze presentate in ingresso all'algoritmo.

Si è supposto che le istanze presentate in ingresso ad ogni iterazione dell'algoritmo di inferenza siano crescenti e includenti le istanze contenute alle iterazioni precedenti, e che le istanze possano divenire anche infinite. Allora si ha identificazione al limite [22] se a partitire da una determinata iterazione dell'algoritmo induttivo H resta invariata.
Nel *PAC-learning* [52] invece si richiede una identificazione solo parziale e probabilistica di $\mathcal{L}$. Sia $\mathcal{L}$ il target e H l'ipotesi induttiva generata da un algoritmo di *IIR* . Sia D una distribuzione di probabilità di tutte le stringhe su Σ^* allora si ha la seguente definizione:

Definizione (True error)**.** Il ***tasso di errore*** o ***true error*** $error_D(H)$ di H rispetto alla distribuzione D e ad $\mathcal{L}$ è

$$error_D(H) = \sum_{x \in \mathcal{L} \oplus L(H)} D(x)$$

Informalmente il *true error* è la probabilità che una stringa estratta casualmente dallo spazio delle istanze — in accordo alla distribuzione di probabilità D — appartenga allo spazio dove H ed $\mathcal{L}$ differiscono . L'algoritmo induttivo nel *PAC-learning* prende due parametri in ingresso l'accuratezza ϵ e la confidenza δ entro cui operare.

Definizione 2.1 (PAC learning)**.** Una classe di linguaggi $\mathbb{L}$ è **PAC-learnable** se esiste un algoritmo A tale che $\forall L \in \mathbb{L}$, per ogni distribuzione D su Σ^*,$\forall\epsilon(0 < \epsilon < 1)$,$\forall\delta(0 < \delta < 1), A$ su istanze fornite in accordo alla distribuzione D produce con probabilità $1 - \delta$ un'ipotesi H tale che $error_D(H) \leq \epsilon$.

E' possibile trovare una caratterizzazione alternativa ma analoga del problema induttivo rispetto a quella data in [4] in cui si effettua una classificazione anche in base a come sono presentate le istanze e al metodo impiegato nello spazio di ricerca:

Presentazione delle istanze Se l'insieme di istanze disponibili per l'algoritmo d'inferenza induttiva regolare, S , per apprendere una *DFA* target A sono tali che $\forall x \in S\, \lambda^A(x) = 1$ si parla di **presentazione positiva** cioè tutti gli esempi(istanze) sono accettanti nel *DFA* A. In questo caso si scrive $S = S^+$. Questo *setting* è denominato *text learning*[17, p. 217]. Si parla invece di **presentazione completa** quando $S = S^+ \cup S^-$ cioè esistono sia esempi accettanti che rigettanti in simboli $\exists x \in S : \lambda^A(x) = 1 \wedge \exists x \in S : \lambda^A(x) = 0$. Quest ultimo caso è menzionato come *informed learning* [17, p. 237] .

Si effettua un'ulteriore suddivisione del tipo di presentazione in base a come gli esempi vengono presentati all'algoritmo d'inferenza:

- **Presentazione Given-Data**
  Le istanze sono un insieme finito presentate totalmente fin dall'inizio del processo.
- **Presentazione completa**
  Le istanze (positive o complete) sono una sequenza infinita e sono presentate in successione (in maniera incrementale).
- Esiste un ***Oracolo*** che è in grado di rispondere a delle *membership query* ed *equivalence query* ed è la macchina inferenziale che direttamente interroga attivamente l'*Oracolo*. Questa modalità sarà approfondita nel capitolo 2.4.

Metodo inferenziale Gli algoritmi di *GI* e di *IIR* si possono ricondurre a dei problemi di ricerca in un grafo in cui ogni nodo è un'astrazione che rappresenta uno dei possibili linguaggi dello spazio dei concetti. $\mathcal{L}$,cioè il target, è da ricercare in questo spazio. Allora con algoritmo d'inferenza *astratto* si intende un algoritmo che ha valenza solo teorica perchè una effettiva realizzazione sarebbe troppo onerosa computazionalmente. Il più importante di questi algoritmi è l'algoritmo di *induzione per numerazione* per il quale esiste la seguente congettura largamente condivisa:

Teorema. *La classe dei linguaggi ricorsivi identificabili al limite da un algoritmo di inferenza grammaticale , è anche identificabile al limite da un algoritmo di induzione per numerazione*

L'induzione per numerazione cerca ad ogni iterazione dell'algoritmo tra le descrizioni di tutti i concetti consistenti con gli esempi in ingresso e ne sceglie uno secondo qualche criterio di preferenza. Tali descrizioni sono tipicamente un insieme di cardinalità infinita e ciò rende questo metodo impraticabile.
Per algoritmi induttivi *concreti* si intende invece quegli algoritmi efficientemente implementabili. Tra questi possiamo distinguere tra:

- **Algoritmi esaustivi**
  Sono algoritmi deterministici che sotto opportune condizioni degli esempi d'ingresso garantiscono di trovare il linguaggio target
- **Algoritmi euristici**
  Si tralascia di seguire dei percorsi di ricerca nel grafo per aumentare l'efficienza ,di solito a scapito della perdita della garanzia di successo. Tipicamente si usano dei meccanismi aleatori, come nel caso degli algoritmi genetici che manipolano delle informazioni simboliche attraverso dei processi aleatori numerici.

2.2 Limiti dell' IIR

Uno dei problemi dell'*GI* e di *IIR* è che nessun sottoinsieme finito di un insieme infinito ha informazioni sufficienti che gli consentono di inferire con assoluta certezza a quale insieme il sottoinsieme appartiene. Nella fattispecie ,dato un insieme finito di stringhe S, appartenente a un linguaggio infinito $\mathcal{L}$, non si può inferire $\mathcal{L}$ da S con assoluta certezza. Il motivo è che S è un sottoinsieme di molti altri, tipicamente infiniti, linguaggi diversi da $\mathcal{L}$. Questo problema è stato trattato da Gold in [22] nel paradigma dell'*identification in the limit* ottenendo i seguenti risultati:

Teorema 2.1. *La classe dei linguaggi superfiniti non può essere identificata al limite attraverso una presentazione positiva di esempi.*

Ne consegue che i linguaggi regolari di cui i linguaggi superfiniti sono un sottoinsieme non sono parimenti identificabili al limite solo tramite esempi positivi. Considerando anche gli esempi negativi il seguente teorema sempre in [22] stabilisce il limite alla classe di linguaggi inferibili:

Teorema 2.2. *Non è possibile inferire l'intera classe dei linguaggi ricorsivi attraverso una presentazione completa degli esempi.*

Invece i linguaggi primitivi ricorsivi e quindi anche i linguaggi regolari che sono un sottoinsieme sono identificabili al limite come attesta il seguente risultato:

Teorema 2.3. *La classe dei linguaggi primitivi ricorsivi è identificabile al limite mediante una presentazione completa degli esempi*

Un altro risultato implicito che si evince dal lavoro di Gold è che non possiamo mai essere sicuri di apprendere il linguaggio corretto a meno che il numero di esempi non sia infinito. Con una presentazione finita l'unica cosa di cui possiamo essere certi è della consistenza della soluzione inferita con il *training set* (la presentazione). Se il *training set* è grande aumenta la fiducia nell'approssimazione della soluzione inferita con il linguaggio target sconosciuto ma senza mai esserne certi e questa assunzione è nota come **Inductive Learning Hypothesis** [1]. Con il *PAC-learning* si ottiene pure un'approssimazione del linguaggio target ma è possibile quantificare quanto vicini saranno la soluzione ottenuta ed il target e con quale probabilità.

Un altro problema che va affrontato è che tipicamente il numero di ipotesi consistenti con gli esempi a disposizione è infinito, quindi è necessario stabilire un **inductive preference bias** [1] cioè un criterio di preferenza da usare per selezionare quale delle congetture consistenti rispetto ai dati scegliere. Nel caso dell'*IIR* l'*inductive preference bias* che si adotta è il principio del ***Rasoio di Occam*** che consiste nello scegliere sempre la soluzione più semplice che nell'*IIR* significa scegliere il *DFA* minimo[2]. Tale scelta da maggiori garanzie che il processo di generalizzazione abbia avuto luogo scongiurando il rischio di *overfitting* cioè di un *DFA* sovradattato agli esempi dati.
Il seguente risultato negativo è relativo a quest ultimo requisito di trovare il *DFA* minimo:

Teorema 2.4. *Trovare il più piccolo automa finito deterministico consistente rispetto a un insieme di esempi completo è un problema NP-hard [21].*

2.3 Passive learning

Gli studi su *GI* si possono suddivedere in due filoni, uno volto ad indagare i limiti teorici dell'apprendimento di linguaggi in determinati paradigmi ,come in 2.2, e l'altro volto a superare sotto opportune assunzioni i limiti teorici emersi rendendo gli algoritmi utilizzabili in termini di efficienza computazionale. Quest ultimo tema sarà affrontato adesso volgendo in particolare l'attenzione sugli algoritmi di **passive learning** e sulla strategia che li governa. La trattazione non sarà volutamente esaustiva dato che l'oggetto di studio di questo lavoro è l'*active learning* che è una tecnica duale rispetto al *passive learning*.

2.3.1 Ricerca nel reticolo

I risultati negativi del teorema 2.4 sull'apprendimento del *DFA* minimo possono essere superati delineando il problema di ricerca del *DFA* minimo come un problema

[2]Si è soliti scegliere quasi sempre i *DFA* come descrizione di un linguaggio regolare proprio perchè assicurano l'esistenza e l'unicità di un *DFA* minimo

di ricerca in un uno spazio degli stati in cui ogni stato è la modellazione di un *DFA*. La strategia più semplice sarebbe quella enumerativa che ha come grafo di ricerca tutto lo spazio delle descrizioni e preleva da questo spazio l' ipotesi minima. Ma tale tecnica è impraticabile perchè tale spazio ha cardinalità infinita (basta pensare che solo i *DFA* consistenti e completi con un dato *training set* sono di per sè infiniti). Allora si definisce come grafo di ricerca il cosidetto **reticolo booleano** indicato con $Lattice(PTA(S^+))$che ha come nodo radice $PTA(S^+)$. La radice può essere modellata come un insieme che comprende tutti gli stati dell'automa $PTA(S^+)$, e l'operatore che consente di generare i nodi figli della radice (e applicando ricorsivamente l'operatore anche ai figli ottenuti consente di derivare anche gli altri nodi) è una relazione d'ordine parziale $\approx$ (cioè binaria transitiva,simmetrica e riflessiva) che applicata al $PTA(S^+)$ consente di derivare gli automi quoziente $PTA(S^+)/\approx$ che non sono altro che una fusione di alcuni degli stati del *DFA* di partenza in base alla relazione d'ordine parziale definita. Inoltre l'automa quoziente $A/\approx$ ottenuto applicando la relazione $\approx$ sul nodo che è un automa A è tale che $L(A) \subseteq L(A/\approx)$ quindi tutti gli automi ottenuti nel reticolo sono consistenti con S^+ ed estendendo il linguaggio da cui sono originati effettuano il processo induttivo generalizzando. Questa tecnica è detta di *state-merging* in quanto ad ogni passo effettua una **fusione** degli stati dell'automa cui la relazione è applicata per ottenere un nuovo nodo e tutto il processo è noto in letteratura come **passive learning**.
Un aspetto che finora è stato trascurato riguarda se il *DFA* minimo è sempre contenuto nel *lattice* altrimenti la procedura di ricerca potrebbe essere infruttuosa. Il seguente teorema [42] da una risposta affermativa se alcune condizioni sono verificate:

Teorema 2.5. *Se l'insieme di istanze positive S^+ è strutturalmente completo rispetto al DFA minimo target M allora $M \in Lattice(PTA(S^+))$.*

laddove

Definizione 2.2 (Insieme di esempi positivi strutturalmente completo)**.** Un insieme di esempi positivi S^+ è detto **strutturalmente completo** rispetto a un automa M accettante L se

- ogni stato di $\mathbb{F}_{\mathbb{A}}^{M}$ è impiegato da almeno una stringa di S^+
- ogni transizione di M è utilizzata nell'accettazione di almeno un esempio di S^+

Tuttavia la cardinalità del $Lattice(PTA(S^+))$ se $\|PTA(S^+)\| = n$ è :

$$C_n = \sum_{i=0}^{n-1} \binom{n-1}{i} C_i \quad \text{con } C_0 = 1$$

e rende impraticabile qualsiasi algoritmo di ricerca per enumerazione. Anche una ricerca in ampiezza è impraticabile. Inoltre per avere la garanzia dell'automa minimo nel *lattice* è necessario fare delle assunzioni, analogamente per effettuare una ricerca efficiente nel *lattice* del *DFA* minimo sarà necessario imporre dei vincoli.

2.3.2 Algoritmi concreti

Qui si presentano le peculiarità dei due principali algoritmi di *passive learning* in uno scenario di *informed learning*. La trattazione ivi esposta è qualitativa e di alto livello ed i dettagli implementativi sono tralasciati.

Red-Blue Framework

Questo framework riduce significativamente il numero di possibili *merge* tra stati senza ridurre il numero di possibili soluzioni. Gli algoritmi con questo *setting* mantengono un cuore di stati *red* e una frontiera di stati *blue* che sono gli stati immediatamente raggiungibili (figli diretti) a partire dagli stati *red*. Un algoritmo di *red-blue state-merging* esegue *merges* solo tra stati *blue* e stati *red* e se non sono possibili *red-blue merges* l'algoritmo ***promuove*** uno stato *blue* a *red*. Gli stati *red* inoltre sono gli stati del *DFA* target che sono già stati identificati. L'idea è stata descritta per la prima volta in [32] in cui una prima versione dell'algoritmo *EDSM*— che valuta il *merge* tra tutte le coppie di stati dell' ipotesi corrente [32, p. 6] — è migliorata nel *Blue-fringe Algorithm* che secondo il principio di blanda località summenzionato valuta il merge solo tra stati di colore *red* e *blue*.
Sebbene la nomenclatura *red-blue framework* non venga ufficialmente introdotta in [32] con questo nome, è adottata in molti autorevoli testi e conferenze di riferimento su *GI* come [17] e [45, p. 70].

RPNI

L'algoritmo *REGULAR POSITIVE AND NEGATIVE INFERENCE* (RPNI) [40] è uno dei più noti algoritmi di *state-merging* che consente di trovare un *DFA* — in uno scenario di *informed learning* — consistente e completo con gli esempi $S = S^+ \cup S^-$ in tempo polinomiale. E' un algoritmo esaustivo che supera il problema della dimensionalità del reticolo booleano eseguendo una ricerca in profondità con backtracking a partire dal $PTA(S^+)$ guidata dagli esempi negativi.
Se S è caratteristico per $\mathcal{L}$ allora RPNI assicura che venga trovato il *DFA* minimo:

Definizione 2.3 (Insieme caratteristico)**.** Un insieme di esempi completo $S = S^+ \cup S^-$ è **caratteristico** per un linguaggio L se:

- S^+ è *strutturalmente completo* (definizione 2.2) per il *DFA* che accetta L
- S^- impedisce il *merge* di due stati p, q di un $DFA \in Lattice(PTA(S^+))$tali che $p \not\equiv q$ durante un'esplorazione del lattice.

Se viene a mancare la condizione in 2.3 di insieme caratteristico non è assicurata la terminazione con il *DFA* minimo ma è comunque garantito un *DFA* consistente rispetto agli *esempi nel training set* grazie alla proprietà d'inclusione descritta in 2.3.1.
Il funzionamento a grandi linee di RPNI è:

1. Costruire $PTA(S^+)$, numerare gli stati in base all'ordine lessicografico delle stringhe di S^+ e inizializzare l'insieme degli stati *red* con lo stato corrispondente a ϵ e gli stati *blue* con i suoi figli diretti.

2. Per ogni stato *blue* si effettua sul *DFA* corrente (inizialmente è $PTA(S^+)$)il *merge* con ogni stato *red* estraendo gli stati in ordine lessicografico ottenendo un *DFA* temporaneo A.

   (a) Se $A[S^-] \notin \mathbb{F}_{\mathbb{A}}^A$ il *merge* è valido, i restanti *merge* dello stato *blue* corrente con gli altri eventuali stati *red* non vengono valutati, ed A è il nuovo automa da considerare.

   (b) Se per il corrente stato *blue* non si trova nessun *merge* compatibile (con nessuno dei nodi *red*), si esegue la sua *promozione* a nodo *red* e gli stati che sono figli diretti di questo nuovo nodo *red* sono aggiunti ai nodi *blue*

3. Torna al punto 2 fin quando l'insieme degli stati *blue* non è vuoto.

Questa procedura può generare condizioni di non-determinismo cioè gli automi intermedi e finale prodotti possono essere degli NFA, per ovviare a questo inconveniente si può rendere il *merge* una procedura ricorsiva che oltre a fondere gli stati rossi e blu effettua eventualmente delle ulteriori fusioni se la condizione di determinismo dell'automa creato dal corrente *merge* non è posseduta. Vedasi [17] per una versione di RPNI che genera sempre *DFA*. Infine il costo computazionale dell'algoritmo nella sua versione originale [40] è $\mathcal{O}((\|S^+\| + \|S^-\|)\|S^+\|^2)$

EDSM

EVIDENCE DRIVE STATE MERGING è un algoritmo di *state-merging euristico* che come RPNI non garantisce di trovare un *DFA* canonico ma solo una soluzione ottima a meno che il *training set* non sia caratteristico 2.3. L'algoritmo nella sua versione base è presentato in [32] ed esistono delle versioni più efficienti come *Blue-fringe EDSM* sempre in [32] che usando il *red-blue framework* limitano notevolemente il numero dei *merge* e ne incrementano le *performances*, ed è questo ultimo che viene brevemente spiegato qui. Il miglioramento rispetto a RPNI è che EDSM non è *greedy* nell'esecuzione dei *merges* ma valuta tutti i merges possibili tra stati *red* e *blue* — invece RPNI selezionerebbe il primo possibile e ignorerebbe gli altri *merges* — e si sceglie il *merge* migliore in base ad un'euristica. La migliore euristica emersa nella competizione Abbandingo è quella di Price che valuta un *merge* non possibile se almeno una stringa accettante e almeno una rigettante rispettivamente di S^+ e S^- terminano nello stesso stato , ed ha invece un punteggio tanto più alto quante più stringhe di S^+ o di S^- (ma non di entrambi gli insiemi, quindi l'*or* è esclusivo) terminano in uno stesso stato del *DFA mergiato* che si sta valutando. Si sceglierà il *DFA* col punteggio maggiore.
Un'altra differenza sostanziale rispetto a RPNI in EDSM è la priorità data alle *promozioni* a scapito di possibili fusioni: il *merge* viene effettuato soltanto se tutti i

possibili *merges* tra tutte le coppie di stati *red* e *blue* è possibile secondo l'euristica, altrimenti si privilegia la *promozione* dello stato *blue* a *red*. EDSM costituisce lo stato dell'arte per ciò che riguarda gli algoritmi di *passive learning*.

2.4 Active Learning

L' *Active Learning* è un caso speciale di *semi-supervisionato machine learning*[3] in cui un algoritmo di *learning* può interagire con l'utente o qualche sorgente d'informazione per ottenere informazioni significative come ad esempio l'etichetta di un'istanza. Nel contesto dell' *IIR* l' *active learning* non esplora il reticolo costruito a partire dal PTA o dall' APTA come fanno gli algoritmi presentati in 2.3.2 ma si basa su una stretta interazione tra il ***learner*** e il ***teacher*** detto anche ***Oracolo*** o ***informant***. L' *active learning* è una tecnica duale rispetto al *passive learning*: il *passive learning* è un approccio *top-down* che esplora lo spazio di ricerca a partire dal nodo iniziale (PTA o APTA costruito dagli esempi iniziali) e tramite dei *merges* genera nuovi nodi e nel caso degenere ha come ultimo nodo l'automa universale, invece l'*active learning* effettua una ricerca *bottom-up* che inizia dall'automa universale e mediante lo ***split*** degli stati raffina l'ipotesi. Inoltre è il *learner* che sceglie attivamente gli esempi e il *teacher* può selezionare attentamente i controesempi significativi: proprio per queste ragioni spesso il numero di esempi per apprendere un concetto e in generale il tempo di esecuzione del processo di apprendimento è minore negli algoritmi di *active learning* che in quelli di *passive learning*. L'*active learning* nasce per ragioni teoriche come ad esempio per dimostrare che non è possibile apprendere in maniera efficiente alcune classi di linguaggi con una presentazione given-data: nell' *active learning* nel cui contesto è il *learner* che seleziona gli esempi è sufficiente dimostrare che il numero di query non può essere polinomiale . Ma è anche applicabile in numerosi contesti pratici come la Robotica in cui un agente può costruire una mappa usando l'interazione tra i sensori e l'ambiente come *Oracolo* o nella modellazione dell'acquisizione dei linguaggi naturali dove attribuire la figura del *teacher* al genitore risulta naturale. Qui si esaminerà l' *active learning* nell' *IIR*. L* è senz altro il più noto algoritmo di *active learning* applicato ai linguaggi regolari. Il più recente e performante *Observation Pack* sarà introdotto nel capitolo 3

2.4.1 Active learning nell' Inferenza Induttiva Regolare

Il paradigma dell' *active learning* si basa sull'esistenza di un *Oracolo* che conosce $\mathcal{L}$ e può rispondere solo a certi tipi di interrogativi sottopostigli dal *learner*. L' *Oracolo* può trovarsi in una situazione in cui più risposte valide sono possibili e in questo caso si deve assumere che non viene rispettata nessuna distribuzione di probabilità nelle risposte date ma che queste sono casuali pertanto nell'analisi dell'algoritmo si deve assumere il caso peggiore cioè un *Oracolo* avverso (nell' algoritmo adoperato

[3]Nel semi-supervisionato learning una piccola quantità di dati è etichettata e la restante, la maggioranza, è senza etichetta.

nell' *IIR*, il table-filling descritto nella sottosezione 2.5.3, l'*Oracolo* non garantisce di ritornare la ***witness*** cioè il controesempio più breve). I principali tipi di interrogativi possibili a cui si può sottoporre un *Oracolo* sono:

- **Membership Query (*MQ*)** Una membership query è effettuata proponendo una stringa all'*Oracolo*, che risponde YES se la stringa appartiene a $\mathcal{L}$ e NO se la stringa non appartiene:

$$MQ : \Sigma^* \rightarrow \{\text{YES,NO}\}$$

- **Equivalence Query (*EQ*)** (forte) Un'equivalence query (forte) è effettuata proponendo un *DFA* ipotesi H all'*Oracolo* che risponde YES se il DFA ipotesi H è equivalente al *DFA* target altrimenti ritorna una stringa(*witness*) appartenente alla differenza simmetrica tra $\mathcal{L}$ e $L(H)$:

$$EQ : DFA \rightarrow \{\text{YES}\} \cup \Sigma^*$$

- ***WEQ*** (debole) Un'equivalence query (debole) è effettuata proponendo un *DFA* ipotesi H all'*Oracolo* che risponde YES se il DFA ipotesi H è equivalente al DFA target altrimenti ritorna NO :

$$WEQ : DFA \rightarrow \{\text{YES,NO}\}$$

- ***SSQ*** Una subset query è effettuata proponendo un DFA ipotesi H all'*Oracolo* che risponde YES se $L(H)$ è un sottoinsieme di $\mathcal{L}$ altrimenti ritorna una stringa appartenente a $L(H)$ che non appartiene ad $\mathcal{L}$:

$$SSQ : DFA \rightarrow \{\text{YES}\} \cup \Sigma^*$$

I seguenti risultati e definizioni sono in [3].Si da la seguente definizione preliminare:

Definizione 2.4. Chiamiamo ρ un'esecuzione del *learner* A. Chiamiamo $\langle r_1, r_2, \ldots r_m \rangle$ la sequenza di risposte alle query $\langle q_1, q_2, \ldots q_m \rangle$ che l'*Oracolo* fa durante l'esecuzione ρ . Si dice che **A** è **polinomialmente limitato** se esiste un polinomio a due variabili p() che dato qualsiasi formalismo L descrivente $\mathcal{L}$ e in qualsiasi esecuzione ρ, e a qualsiasi *query point*(indica il momento prima che avvenga una query ed è definito come un numero intero che specifica il numero di query avvenute fino a quel momento) k dell'esecuzione, denotando il tempo di esecuzione prima di quel punto con t_k, si ha:

- $k \leq p(\|L\|, \max\{|r_i| : i < k\})$
- $|q_k| \leq p(\|L\|, \max\{|r_i| : i < k\})$
- $t_k \in \mathcal{O}(p(\|L\|, \max\{|r_i| : i < k\}))$

Informalmente significa che in qualsiasi *query point* k di qualunque esecuzione, al momento precedente l'effettuazione della query q_k, si ha che il numero di query fatte, il tempo di esecuzione e la dimensione della prossima query (q_k) sono tutte limitate da un polinomio p dipendente dalla dimensione del target e dalla lunghezza del più lungo controesempio ritornato dall'*Oracolo* fino a quel punto
La seguente definizione stabilisce quando una classe di linguaggi è efficientemente *identificabile in the limit* da un algoritmo di *learning*.

Definizione 2.5. Una classe di linguaggi $\mathfrak{L}$ è **polinomialmente *identificabile in the limit con query*** fissati i tipi di query possibili se esiste un **polinomialmente limitato** *learner* **A** che dato il formalismo descrivente qualsiasi linguaggio $L \in \mathfrak{L}$, identifica L in the limit, cioè ritorna L' equivalente ad L e termina.

Adesso ci si chiede se la classe dei linguaggi regolari è polinomialmente identificabile in the limit tramite qualche algoritmo di apprendimento secondo la definizione 2.5. E' importante sottolineare che la risposta a questa domanda dipende anche dalla classe cui appartiene l'*Oracolo* cioè dal tipo di interrogativi che è possibile rivolgergli. A tal proposito si hanno i seguenti risultati :

Teorema 2.6. *La classe dei linguaggi regolari non è polinomialmente identificabile in the limit da un numero polinomiale di MQ, WEQ e SSQ*

Quindi come conseguenza del teorema 2.6 la classe dei linguaggi regolari non è polinomialmente identificabile in the limit neanche sottoponendo all'*Oracolo* esclusivamente *MQ* .
Un'ulteriore risultato è il seguente:

Teorema 2.7. *La classe dei linguaggi regolari (usando i DFA come formalismo descrittivo) non è polinomialmente identificabile in the limit da un numero polinomiale di EQ (forti)*

Si rimanda alla sezione 2.5 per le condizioni di polinomiale identificabilità in the limit dei linguaggi regolari

2.5 L*

L* è il più noto algoritmo di *active learning* nell'ambito dell' *IIR* e garantisce di emettere in output il DFA minimo (o uno ad esso isomorfo) accettante $\mathcal{L}$. Detto n il numero degli stati del DFA target minimo ed m la lunghezza del più lungo controesempio ritornato dall' *Oracolo* durante l'inferenza, il costo computazionale di L* sarà limitato da una funzione polinomiale di n ed m. In L* il *teacher* appartiene alla classe dei **Minimally Adeguate Teacher (*MAT*)** in grado di rispondere ad *EQ* e *MQ*. Questi risultati ,che consentono di dire che i linguaggi regolari sono polinomialmente identificabili in the limit (definizione 2.5), sono stati conseguiti da Dana Angluin [2] e succintamente riportati nel seguente teorema:

Teorema 2.8. *Dato un MAT presentante un linguaggio regolare sconosciuto U, il Learner L* termina restituendo in output un automa finito isomorfo al DFA minimo accettante il linguaggio target U. Inoltre, se n è il numero di stati del DFA minimo accettante U e m è un limite superiore della lunghezza di ogni controesempio ritornato dal Teacher, allora il costo totale di esecuzione di L* è limitato da un polinomio in n ed m*

L* viene presentato all'interno del *red–blue framework* (introdotto in 2.3.2) che in algoritmi come *EDSM* (vedasi sottosezione 2.3.2) consente di diminuire i *merges*. In L* l'adozione di questo *framework* malgrado non comporti un vantaggio computazionale consente un'esposizione più chiara.

2.5.1 Tabella di Osservazione

Una **tabella di osservazione** è una struttura dati che rappresenta il DFA ipotesi congetturato al passo corrente. Al suo interno sono codificati gli esiti delle *MQ* richieste al *teacher*.

Definizione (Tabella di Osservazione)**.** La *tabella di osservazione* è una tripla $\langle STA, EXP, OT\rangle$, dove:

- STA = RED $\cup$ BLUE. STA è un insieme finito di stringhe definite su Σ che rappresentano gli stati. STA è **prefix–closed**
  RED $\in \Sigma^*$ è un insieme finito di stati
  BLUE = $\{ua \notin \text{RED} : u \in \text{RED}\}$ è l'insieme dei successori degli stati RED che non sono RED. Rappresentano le transizioni.
- EXP $\in \Sigma^*$ è l'insieme degli esperimenti. E' **suffix–closed**
- OT : STA $\times$ EXP $\rightarrow$ {0,1,*} è una funziona così definita:

$$OT[u][e] = \begin{cases} 1 & \text{se } ue \in \mathcal{L} \\ 0 & \text{se } ue \notin \mathcal{L} \\ * & \text{altrimenti} \end{cases}$$

Dalla tabella di osservazione si costruisce una nuova ipotesi e la si sottopone al *teacher*. Se l'ipotesi non è equivalente al *DFA target* il *teacher* torna un controesempio che sarà usato dal *learner* per *splittare* gli stati e modificare la tabella di osservazione per ottenere una nuova ipotesi cofacente al controesempio. Le *MQ* permettono di riempire i buchi generati dall'introduzione di nuovi prefissi dal controesempio. A partire dalla tabella di osservazione è possibile costruire un DFA ipotesi sole se questa gode di tre proprietà:

Completezza

La completezza garantisce che non ci siano comportamenti parzialmente (o totalmente) sconosciuti per prefissi presenti all'interno della tabella.

Definizione (Tabella completa)**.** Una tabella è completa se non ha *buchi*. Un *buco* in una tabella di osservazione è una coppia (u,e) tale che $OT[u][e] = *$.

L'eventuale incompletezza può essere eliminata mediante *MQ* al *teacher*.

Chiusura

La chiusura (algoritmo 4) assicura che ogni possibile stato raggiunto con una transizione sia presente tra gli stati finali dell'automa. Dato un elemento $s \in$ STA e gli n esperimenti $e \in$ EXP si indica con row(s) la riga in OT indicizzata da s cioè $row(s) = OT[s][e_1] \cdot OT[s][e_2] \cdot \ldots OT[s][e_n]$. Gli stati dell'automa sono un sottoinsieme degli stati RED, quando una transizione da uno stato RED porta ad uno stato BLUE si deve trovare uno stato RED equivalente a quello BLUE (vedasi algoritmo 1) (almeno secondo i suffissi trovati fino a quel momento) in modo che la transizione arrivi in questo stato RED(che è presente nell'automa ipotesi perchè gli stati RED trovati fanno parte del *DFA* ipotesi a differenza di quelli BLUE).

Definizione (Tabella chiusa)**.** Una tabella è **chiusa** se $\forall u \in$ BLUE, $\exists s$ $\in$ RED : row(u) = row(s)

Se la tabella di osservazione non fosse chiusa è possibile renderla tale mediante una (o più) **promozione**, cioè l'inserimento di **u** responsabile della non chiusura nei RED e $u \cdot \Sigma$ nei BLUE

Consistenza

La consistenza (algoritmo 5) impedisce situazioni di indeterminismo nel DFA, nella fattispecie che da uno stato dell'ipotesi per uno stesso simbolo dell'alfabeto si giunga in stati di arrivo diversi. Questa situazione è resa possibile dal fatto che l'algoritmo non impedisce di avere due stati RED s_1 ed s_2 tali che $\text{row}(s_1) = \text{row}(s_2)$.

Definizione (Tabella consistente)**.** Una tabella di osservazione è **consistente** se $\forall s_1, s_2 \in \text{RED} : \text{row}(s_1) = \text{row}(s_2) \implies \forall a \in \Sigma, \text{row}(s_1 a) = \text{row}(s_2 a)$

La definizione sopra significa che affinchè vi sia consistenza ogni coppia di stati equivalenti RED cioè di stati in RED con righe uguali deve restare equivalente in STA aggiungendo qualsiasi simbolo dell'alfabeto. Se la tabella di osservazione non fosse consistente è possibile renderla tale ampliando l'insieme EXP con la stringa ottenuta dalla concatenazione del simbolo dell'alfabeto e dall'esperimento che hanno generato l'inconsistenza. Ciò assicura che i due stati s_1 ed s_2 che prima erano equivalenti (e che quindi rappresentavano un unico stato nell'ipotesi) adesso non lo sono più perchè $\text{row}(s_1) \neq \text{row}(s_2)$ e quindi sarà aggiunto un nuovo stato all'insieme RED cioè un nuovo stato all'ipotesi.

2.5.2 L'algoritmo

Funzionamento

La *ratio* che ispira L* é il teorema *Myhill-Nerode* (sezione A.1). Nella tabella di osservazione le righe RED,in realtà un sottoinsieme delle stringhe RED, corrispondono

Algoritmo 1 LSTAR-BUILDAUTOMATON

Input: a closed and complete observation table $\langle$STA,EXP,OT$\rangle$
Output: DFA $\langle \Sigma, Q, q_\epsilon, F_A, F_R, \delta \rangle$

$Q \leftarrow \{q_u : u \in \text{RED} \wedge \forall v < u \, \text{row}(v) \neq \text{row}(u)\}$
▷ le stringhe più corte sono minori e per stringhe della stessa lunghezza si intendono minori quelle che lessicograficamente vengono prima
$F_{\mathbb{A}} \leftarrow \{q_u \in Q : \text{OT}[u][\epsilon] = 1\}$
$F_{\mathbb{R}} \leftarrow \{q_u \in Q : \text{OT}[u][\epsilon] = 0\}$
for $q_u \in Q$ **do**
    for $a \in \Sigma$ **do** $\delta(q_u, a) \leftarrow q_w \in Q : \text{row}(ua) = \text{row}(w)$
end for
return $\langle \Sigma, Q, q_\epsilon, F_A, F_R, \delta \rangle$

Algoritmo 2 LSTAR

Input: –
Output: DFA $\mathcal{A}$

LSTAR-INITIALISE
repeat
    while $\langle$STA,EXP,OT$\rangle$ *is not closed or not consistent* **do**
        if $\langle$STA,EXP,OT$\rangle$ *is not closed* **then**
            $\langle$STA,EXP,OT$\rangle \leftarrow$ LSTAR-CLOSE($\langle$STA,EXP,OT$\rangle$)
        if $\langle$STA,EXP,OT$\rangle$ *is not consistent* **then**
            $\langle$STA,EXP,OT$\rangle \leftarrow$ LSTAR-CONSISTENT($\langle$STA,EXP,OT$\rangle$)
    end while
    Answer $\leftarrow$ EQ($\langle$STA,EXP,OT$\rangle$)
    if Answer $\neq$ YES **then**
        $\langle$STA,EXP,OT$\rangle \leftarrow$ LSTAR-USEEQ($\langle$STA,EXP,OT$\rangle$, Answer)
until Answer = YES
return LSTAR-BUILDAUTOMATON($\langle$STA,EXP,OT$\rangle$)

Algoritmo 3 LSTAR-INITIALISE

Input: –
Output: $\langle$STA,EXP,OT$\rangle$

RED $\leftarrow \{q_\epsilon\}$
BLUE $\leftarrow \{q_a : a \in \Sigma\}$
EXP $\leftarrow \{\epsilon\}$
OT$[\epsilon][\epsilon] \leftarrow$ MQ(ϵ)
for $a \in \Sigma$ **do** OT$[a][\epsilon] \leftarrow$ MQ(a)
return $\langle$STA,EXP,OT$\rangle$

Algoritmo 4 LSTAR-CLOSE

Input: ⟨STA,EXP,OT⟩
Output: ⟨STA,EXP,OT⟩ updated
1: **for** $s \in$ BLUE *such that* $\forall u \in$ RED $\text{row}(s) \neq \text{row}(u)$ **do**
▷ Non per $\forall s$ ma per uno solo quindi la tabella in output può ancora essere non chiusa
2: RED $\leftarrow$ RED $\cup \{s\}$
3: BLUE $\leftarrow$ BLUE $\setminus \{s\}$
4: **for** $a \in \Sigma$ **do** BLUE $\leftarrow$ BLUE $\cup \{s \cdot a\}$
5: **for** $u, e \in \Sigma^*$ *such that* OT$[u][e]$ *is a hole* **do** OT$[u][e] \leftarrow$ MQ(ue)
6: **end for**
7: **return** ⟨STA,EXP,OT⟩

Algoritmo 5 LSTAR-CONSISTENT

Input: ⟨STA,EXP,OT⟩
Output: ⟨STA,EXP,OT⟩ updated
1: find $s_1, s_2 \in$ RED, $a \in \Sigma$ and $e \in$ EXP *such that* $\text{row}(s_1) = \text{row}(s_2)$ and
2: OT$[s_1 \cdot a][e] \neq$ OT$[s_2 \cdot a][e]$
▷ se $s_1 a$ ed $s_2 a$ differiscono per più di un esperimento basta considerarne uno
3: EXP $\leftarrow$ EXP $\cup \{a \cdot e\}$
4: **for** $u, e \in \Sigma^*$ *such that* OT$[u][e]$ *is a hole* **do** OT$[u][e] \leftarrow$ MQ(ue)
5: **return** ⟨STA,EXP,OT⟩

Algoritmo 6 LSTAR-USEEQ

Input: ⟨STA,EXP,OT⟩ , Answer
Output: ⟨STA,EXP,OT⟩ updated
1: **for** $p \in$ PREF(Answer) **do** ▷ Anche Answer fa parte dei prefissi
2: RED $\leftarrow$ RED $\cup \{p\}$ ▷ Se un pref. è già in OT renderlo RED se non lo è
3: **for** $a \in \Sigma : pa \notin$ PREF(Answer) **do** BLUE $\leftarrow$ BLUE $\cup \{pa\}$
4: **end for**
5: **for** $u, e \in \Sigma^*$ *such that* OT$[u][e]$ *is a hole* **do** OT$[u][e] \leftarrow$ MQ(ue)
6: **return** ⟨STA,EXP,OT⟩

agli stati del DFA ipotesi e le colonne, l'insieme EXP, corrispondono alle stringhe rappresentanti i suffissi che distinguono coppie di stati distinti dell'ipotesi. I singoli stati sono etichettati dalle stringhe che portano dallo stato iniziale allo stato stesso. Lo stato iniziale è etichettato dalla stringa ϵ. Per ogni stato q l'etichetta della colonna (un esperimento) indica lo stato che potrebbe essere raggiunto a partire dallo stato q dopo la lettura della stringa corrispondente all'etichetta dell'esperimento. Due stati sono considerati equivalenti se hanno le righe uguali nella tabella.
L'algoritmo inizia costruendo una tabella di osservazione corrispondente all'automa universale (algoritmo 3) Una volta resa la tabella completa,chiusa e consistente viene estratta l'ipotesi corrispondente[4] e viene effettuta un'*EQ* dell'ipotesi al *teacher*. In caso di risposta affermativa cioè di equivalenza il processo termina in quanto il *learner* ha identificato un automa uguale o equivalente al *target* (L* inferisce il DFA minimo, invece il *target* potrebbe non essere un DFA minimo). In caso contrario sarà ritornato un controesempio che sarà usato per modificare la tabella di osservazione e quindi formulare una nuova ipotesi. L'algoritmo 2 chiarifica e approfondisce i passaggi summenzionati.

Correttezza

Per verificare che L* è corretto è sufficiente dimostare che termina dato che la terminazione con un *MAT* assicura l'equivalenza dell'ipotesi col target. A tal fine si riporta preliminarmente il seguente teorema ([2]):

Teorema 2.9. *Se una tabella di osservazione è completa,chiusa e consistente, allora l'ipotesi H (quella inferita dall'algoritmo 1) è consistente con la funzione OT. Qualsiasi altra ipotesi consistente con OT ma non equivalente ad H deve avere più stati.*

Nel teorema 2.9 con ipotesi consistente con la funzione OT si intende che $\forall u \in$ STA e $\forall e \in \text{EXP}, ue \in \mathcal{L} \iff \text{OT}[u][e] = 1$
Il risultato del teorema 2.9 è che qualunque DFA consistente con OT o è isomorfico all'ipotesi inferita da L* o contiene almeno uno o più stati. Quindi ogni ipotesi H fatta da L* è sempre il minimo DFA consistente con OT.
Un altro risultato che ci torna utile è:

Lemma 1. *Detto n il numero di differenti valori di row(s) per $\forall s \in$ RED in una tabella di osservazione. Qualsiasi H consistente con OT deve avere almeno n stati*

Si indica con n il numero di stati del DFA minimo accettante $\mathcal{L}$. E' facile dimostrare che il numero di valori distinti di $row(s)$ per $s \in$ RED è incrementato monotonicamente fino ad un massimo di n durante l'esecuzione di L*. Infatti sia la chiusura che la consistenza introducono un nuovo stato RED. Se la tabella fosse già chiusa e consistente il controesempio t tornato dal *teacher* comunque garantisce che un

[4]La prima ipotesi fatta dal *learner* non è sempre l'automa universale, ma dipende dall'esito delle *MQ*

nuovo stato RED venga aggiunto alla tabella di osservazione: detto T_u il DFA *target* minimo (ovviamente consistente con OT) — il controesempio t ci permette di dedurre che H e T_u non sono equivalenti — , allora dal teorema 2.9 sappiamo che H ha al massimo $n-1$ stati. Inoltre L* classificherà il controesempio t allo stesso modo di T_u e quindi il nuovo DFA ipotesi *H'* che si otterrà non sarà equivalente ad H (per via di t) e inoltre sarà consistente con OT, quindi dal teorema 2.9 si deduce che *H'* deve avere almeno n stati (almeno 1 stato in più di H).
Questo dimostra che ad ogni passo del ciclo più esterno di L* almeno uno stato RED distinto deve essere aggiunto sempre alla tabella di osservazione. Quando ci saranno n stati RED distinti L* troverà il DFA *target* minimo consistente con OT infatti il DFA *target* minimo è sempre consistente con OT[5] e l'ipotesi creata da L* è sempre il DFA minimo per il teorema 2.9 e dal lemma 1 si ha che l'ipotesi creata da L* deve avere almeno n stati. Essendo il DFA ipotesi un DFA ipotesi minimo e con n stati e consistente con T non può essere che uguale o isomorfo col DFA *target minimo*. Quindi L* dovrà costruire nel caso peggiore $n-1$ ipotesi errate prima di trovare l'ipotesi corretta e quindi il numero di *EQ* è al massimo n (perchè l'ipotesi corretta comunque va sottoposta al *teacher*)

Complessità computazionale

Come detto in precedenza la complessità computazionale di L* è limitata da un polinomio dipendente dal numero di stati del DFA minimo identificante $\mathcal{L}$ e dalla lunghezza del controesempio più lungo ritornato dal *teacher*.In [2] si trova una dimostrazione dettagliata di quanto detto sopra. In questa sede si analizzano dei parametri oggettivi nella valutazione del costo computazionale cioè il numero di *MQ* e di *EQ*. In quest'analisi si terrà conto anche di k un ulteriore parametro che rappresenta la dimensione dell'alfabeto. Il numero di *EQ* sarà limitato da n (sottosezione correttezza 2.5.2) come dimostrato in precedenza. Il numero di *MQ* è invece limitato dalla dimensione della tabella di osservazione. Il numero di elementi in EXP non può eccedere n , in quanto l'insieme EXP viene incrementato di un elemento quando la tabella di osservazione è inconsistente (algoritmo 5) e l'inconsistenza può presentarsi al più $n-1$ volte cioè ogni volta che viene aggiunto un nuovo nodo RED distinto (con n nodi RED distinti L* termina quindi se ne aggiungono al più $n-1$) (la dimensione di EXP è al più n e non $n-1$ perchè EXP inizialmente contiene λ).EXP rappresenta il numero di colonne della tabella di osservazione, adesso si calcola il numero di righe della stessa. Si indica con m la lunghezza del controesempio più lungo ritornato dal *teacher*. Il numero di stati RED non può eccedere $n+m(n-1)$ perchè gli stati RED sono aggiunti quando si scopre che la tabella non è chiusa e quando il *teacher* torna un controesempio. La non chiusura può accadere al più $n-1$ volte ed ogni volta aggiunge uno stato RED, e ci possono essere massimo $n-1$ con-

[5]Ma se sia il *target* che H sono sempre consistenti con OT come si fa a trovare un controesempio? La risposta è semplice: da OT viene creata un' ipotesi H consistente all'OT ma nell'ipotesi possono essere *parsate* anche altre stringhe non contemplate in OT da cui può derivare la non equivalenza con il *target*

troesempi ognuno dei quali può causare l'aggiunta di al più m stati RED (i prefissi aggiunti ad ogni iterazione sono al più pari alla lunghezza del controesempio e nel caso peggiore in cui tutti i controesempi sono lunghi m il numero dei prefissi aggiunti ad ogni iterazione è m). Il numero degli stati BLUE è al più $k(n+m(n-1))$ perchè i BLUE, che rappresentano le transizioni, sono ottenuti concatenando tutti i simboli dell'alfabeto agli stati RED. Quindi la dimensione della tabella sarà righe $*$ colonne cioè (RED + BLUE) $*$ EXP quindi si ha:

$$(k+1)(n+m(n-1))\,n = \mathcal{O}(kmn^2)$$

che è il numero di *MQ* totali.

2.5.3 Il teacher

Il teacher di L* essendo un *MAT* è chiamato a rispondere a due tipi di query: *MQ* ed *EQ*. Si suppone che esso abbia a disposizione il DFA che identifica $\mathcal{L}$ quindi è immediato rispondere a una *MQ*. Il *teacher* deve anche vagliare l'equivalenza del target con l'ipotesi fornitagli dal *learner* e in caso di inequivalenza deve tornare un controesempio. A tal fine il *table-filling algorithm* [39] risulta essere un buon algoritmo (il più performante con complessità quasi lineare atto solo a testare l'equivalenza e tornare una witness quindi senza consentire anche la minimizzazione è [24])

Table-filling

Il *table-filliling* [39] è un algoritmo in grado di individuare ricorsivamente tutti gli stati tra loro distinti, alla fine dell'esecuzione le coppie di stati non marcati come tali saranno coppie di stati equivalenti. Per questo motivo l'algoritmo di table-filling è utilizzato anche nella minimizzazione di DFA dove gli stati trovati equivalenti saranno fusi in un unico stato.
Per stati distinti si intende stati per cui esiste almeno una stringa che partendo da quei due stati (e non dallo stato iniziale) giunge in una coppia di stati di arrivo composta da uno stato accettante e da uno stato rigettante.
Inizialmente si distinguono le coppie di stati che non sono equivalenti cioè gli stati che vengono distinti dalla stringa vuota cioè quelle coppie di stati formate da uno stato accettante e da uno rigettante. Al passo successivo si procede esaminando tutte le coppie di stati che momentaneamente l'algoritmo considera equivalenti (che non ha marcato come distinti nei passi precedenti): se per un simbolo dell'alfabeto s da quella coppia di stati di partenza si arriva a una coppia di stati distinti (già marcati dall'algoritmo nei passi precedenti) anche la coppia di stati di partenza va marcata come distinti perchè se gli stati di arrivo sono distinti vuol dire che esiste un suffisso w che li distingue quindi gli stati di partenza saranno distinti dalla stringa sw. Questa procedura va ripetuta ed ha termine quando al passo corrente l'algoritmo non ha trovato nessuna nuova coppia di stati distinti.Inoltre come attesta il seguente teorema:

Teorema. *Se due stati non sono marcati come distinti dall'algoritmo di table-filling, allora questi stati sono equivalenti. [25]*

si ha che quando vengono identificati gli stati equivalenti è possibile passare alla minimizzazione tramite il merge di questi stati.
Per testare l'equivalenza di due DFA si manda in esecuzione il *table-filling* sul DFA costituito dall'unione dei due DFA di cui verificare l'equivalenza. Se al termine dell'esecuzione i due stati iniziali risultano equivalenti i due DFA di partenza saranno equivalenti perchè non esiste nessuna stringa che distingue i due stati iniziali e quindi i due linguaggi dei due DFA sono identici. Per ottimizzare l'esecuzione è possibile interrompere l'esecuzione non appena viene individuato che i due stati iniziali sono distinti.
Per abilitare il *teacher* a ritornare, in caso di inequivalenza, una *witness* è necessario modificare leggermente il *table-filling*. Anzichè limitarsi nel marcare le coppie di stati non equivalenti è necessario anche memorizzare il simbolo dell'alfabeto che ha causato l'inequivalenza. Inoltre bisogna marcare con la stringa vuota o con un marcatore speciale le coppie di stati distinti in fase d'inizializzazione . Quando l'algoritmo termina è possibile creare il controesempio, partendo dalla coppia di stati iniziali, percorrendo la struttura dati usata durante il *table-filling* con l'ausilio delle funzioni di transizione dei due DFA e del marcatore memorizzato fino a quando non viene trovata una coppia di stati contrassegnata con la stringa vuota (cioè uno stato accettante e l'altro no). E' garantito che un controesempio venga sempre individuato ma non vi è la garanzia che ad essere scovato sia quello di lunghezza minima.

Una versione più efficiente

Il *table-filling* ha una complessità polinomiale rispetto ad n cioè alla somma del numero degli stati dei DFA di cui si vuole testare l'equivalenza. Si avranno $n * \frac{n-1}{2}$ coppie distinte di stati che verranno tutte considerate ad ogni visita della tabella (la struttura dati mantenuta dall'algoritmo) quindi $\mathcal{O}(n^2)$. Nel caso peggiore ad ogni iterazione una sola coppia verrà scoperta essere distinta e le coppie sono tutte distinte il numero di iterazioni sono al più $\mathcal{O}(n^2)$. Moltiplicando le due complessità si ottiene $\mathcal{O}(n^4)$ che è il costo computazionale nel caso peggiore.
E' possibile migliorare la complessità computazionale a $\mathcal{O}(n^2)$ memorizzando per ogni coppia di stati (i, j) una lista di dipendenza costituita da tutte quelle coppie (x,y) che tramite un singolo simbolo dell'alfabeto k arrivano in (i, j) cioè $\hat{\delta}(x, k) = i$ e $\hat{\delta}(y, k) = j$. Si memorizzano in una coda tutte le coppie di stati inizialmente distinte. Si estrae una coppia dalla coda (che quindi è distinta) e tutte le coppie che da essa dipendono sono marcate come distinte nella tabella e sono aggiunte in fondo alla coda. L'algoritmo ripete questi passi finchè la coda è vuota.

Capitolo 3

Observation Pack

Esistono molte varianti dell'algoritmo L* originariamente presentato da Angluin. L'algoritmo *ObP* presentato da Falk Howar in [26] –presentato per le Mealy Machines ma comunque applicabile ai DFA – si basa su alcune di queste varianti. Tecnicamente l'*ObP* combina l'idea di usare un **Discrimination Tree (*DT*)** per i linguaggi regolari [28] con una versione localizzata della tabella di osservazione [43]. Inoltre viene utilizzata una tecnica più efficiente di gestione del controesempio rispetto ad L*. Quindi l'*ObP* è in stretta correlazione con L*, e molti dei concetti introdotti nella sezione 2.5 rimangono validi. Anche *ObP*, come L* è un algoritmo di *active learning* nell'ambito dell'*IIR* che garantisce di emettere in output il *DFA* minimo (o uno ad esso isomorfo) accettante $\mathcal{L}$. Anche in questo caso detto n il numero degli stati del DFA target minimo ed m la lunghezza del più lungo controesempio ritornato dal *teacher* durante l'inferenza, il numero di *MQ* sarà limitato da una funzione polinomiale di n ed m e il numero di *EQ* linearmente da n. Nonostante L* sia il più noto algoritmo di *active learning* l'*ObP* ottiene prestazioni migliori dato che permette di diminuire il numero delle *MQ*. Nell' *ObP* il teacher appartiene ancora alla classe dei *MAT*, in grado di rispondere ad *EQ* e *MQ*.

3.1 Fondamenti teorici

Si parla di *active learning* perchè il *learner* attivamente può interrogare il *teacher* sull'appartenenza o meno di alcune stringhe ad $\mathcal{L}$, in contrapposizione al *passive learning* in cui le stringhe sono date a priori come in *EDSM* ad esempio. Il *learner* da un certo punto di vista si trova ad affrontare un problema di *classificazione* cioè deve assegnare alcune stringhe ad un determinato stato di *H*. L'ipotesi *H* ottenuta induttivamente sarà consistente con l'etichettatura degli esempi sottoposti al *teacher* fino al momento della creazione di *H* ma produrrà una generalizzazione dato che in *H* è possibile effettuare il *parsing* di stringhe mai sottoposte al *teacher*. Detto T il *DFA* target,la *classificazione* delle stringhe e la costruzione dell'ipotesi si basa sui risultati del teorema Myhill-Nerode (teorema A.1) che consente di:

1. Trovare un insieme $Sp \subset \Sigma^*$ di **prefissi**, detti **short prefix o access sequence**, in cui ogni short prefix è una stringa reppresentativa per una classe di equivalenza sulla relazione d'equivalenza $\simeq_{\lambda^T}$

2. Trovare un insieme $V \subset \Sigma^*$ di **suffissi** che è sufficiente a realizzare la relazione di Nerode su Sp, cioè tale che $s \not\simeq_{\lambda^T} s'$ implica $\lambda^T(sv) \neq \lambda^T(s'v)$ per $s, s' \in Sp$ e qualche $v \in V$

Il teorema di Myhill-Nerode asserisce che un linguaggio $L(T) = \mathcal{L}$ è regolare se e solo se $\simeq_{\lambda^T}$ identifica un numero finito di classi d'equivalenza. $\simeq_{\lambda^T}$ agisce su due stringhe x e y che sono in relazione se non esiste nessuna stringa z tale che xz e yz, esattamente una delle due appartiene ad $\mathcal{L}$: quindi nella classe d'equivalenza ci saranno stringhe non distinguibili da nessuna altra stringa e che quindi rappresentano un unico stato dell'automa T. L'*ObP* trova dei prefissi in cui ogni singolo prefisso è una stringa. contenuta in una specifica classe d'equivalenza (e vi è un prefisso rappresentativo, detto short prefix, per ogni classe d'equivalenza). Essendo $L(T) = \mathcal{L}$ un linguaggio regolare, il numero di classi d'equivalenza è finito e sarà certamente possibile trovare un insieme di short prefix Sp . L'esistenza dell'insieme di suffissi V in 2 è garantita dal fatto che se esistono almeno due classi di equivalenza deve esistere almeno un suffisso che distingue le due classi d'equivalenza (che altrimenti sarebbero un'unica classe d'equivalenza). Quindi $|V|$ è limitato dall'indice di $\simeq_{\lambda^T}$ cioè dal numero di stati del *DFA* target.

3.2 Costruzione dell'ipotesi

L'*ObP* mantiene due strutture dati rappresentative dell'ipotesi: il DT e un insieme di componenti.

3.2.1 Tabella di osservazione localizzata

A differenza di L* in cui vi è un'unica tabella d'osservazione rappresentativa dell'ipotesi, in *ObP* vi è una tabella di osservazione di dimensioni ridotte per ogni stato trovato fino a quel momento denotata come **componente**:

Definizione (Componente)**.** Un componente C è una quadrupla $\langle U, u_0, V, OT \rangle$ dove:

$U \subset \Sigma^*$ è un insieme finito di prefissi
$u_0 \in U$ è l'unico short prefix del componente
$V \subset \Sigma^*$ è un insieme di suffissi $v_1, \ldots, v_k$
$OT : U \times V \to \{0, 1, *\}$ è una funzione così definita:

$$OT[u][v] = \begin{cases} 1 & \text{se } uv \in \mathcal{L} \\ 0 & \text{se } uv \notin \mathcal{L} \\ * & \text{altrimenti} \end{cases}$$

Sia $u \in U$ e sia $|V| = n$, si indica con row(u) la riga in OT indicizzata da u cioè $row(u) = OT[u][v_1] \cdot OT[u][v_2] \cdot \ldots OT[u][v_n]$. Il componente è individuato dall'access sequence e quindi spesso lo si indica con C_{u_0}

Un componente approssima[1] la relazione di Nerode. Ogni componente rappresenta una classe di equivalenza: tutti i prefissi del componente fanno parte della stessa classe di equivalenza cioè sono equivalenti (secondo OT) in base ai suffissi di quel componente. I valori di output di OT sono ricavati tramite delle *MQ*. L'access sequence non è altro che un prefisso, possibilmente il più breve ma non necessariamente, rappresentativo del componente. Nell'ipotesi H ad ogni componente corrisponde uno stato. I prefissi di un componente sono tutte quelle stringhe che terminano nello stato dell'ipotesi rappresentato da quel componente. L'insieme di tutti i componenti è indicato con C_I, invece con Sp(C_I) o semplicemente Sp si indica l'insieme delle access sequences di tutte le componenti. **Sp è prefix-closed**.

Completezza

La completezza garantisce che non ci siano comportamenti parzialmente (o totalmente) sconosciuti per prefissi presenti all'interno di una componente.

Definizione (Componente completo)**.** Un componente è completo se non ha *buchi*. Un *buco* in un componente è una coppia (u,v) tale che $OT[u][v] = *$.

C_I è completo se tutti i componenti sono completi. L'eventuale incompletezza può essere eliminata mediante *MQ* al *teacher*.

Chiusura

La proprietà di chiusura di C_I garantisce che tutti i prefissi di un componente sono equivalenti secondo la relazione di Nerode approssimata.

Definizione (Componente chiuso)**.** Un componente $\langle U, u_0, V, OT \rangle$ è chiuso se row(u) = row(u_0) per $\forall u \in U$.

C_I è chiuso se tutti i componenti sono chiusi.

La proprietà di chiusura su C_I consente l'applicazione del teorema di Myhill-Nerode sulla relazione $\simeq_{OT}$. La relazione di Nerode è soltanto approssimata in quanto la funzione OT non è definita su Σ^* ma su un dominio più ristretto. Tuttavia, se le proprietà di chiusura e completezza su C_I sono mantenute il teorema Myhill-Nerode è ancora valido e come visto in A.1 assicura la costruzione di un *DFA* la cui *funzione di output* è quella su cui si basa la relazione di Nerode utilizzata che nella fattispecie

[1] Si parla di approssimazione perchè può accadere che alcuni prefissi che attualmente fanno parte dello stesso componente in futuro facciano parte di componenti diverse. Ciò è dovuto all'aggiunta di nuovi suffissi non ancora esaminati fino a quel momento. In ultima analisi ciò è dovuto al fatto che OT ha un dominio ristretto ad $U \times V$ anzichè Σ^*

è OT. Un ipotesi H consistente con OT può essere costruita come nell'algoritmo 7. L'eventuale non chiusura di un componente può essere eliminata con uno *split* , che consiste nel dividere un componente in due componenti oppure uno stato in due stati (un componente corrisponde ad uno stato).

Algoritmo 7 OBP-BUILDAUTOMATON

Input: a closed and complete components set C_I
Output: DFA $\langle \Sigma, Q, q_\epsilon, F_A, F_R, \delta \rangle$
1: $Q \leftarrow \{q_u : \forall u \in \mathrm{Sp}(C_I)\}$
2: $q_\epsilon \leftarrow$ Component with short prefix ϵ
▷ Stato iniziale corrispondente al componente con short prefix ϵ
3: $F_{\mathbb{A}} \leftarrow \{q_u \in Q : \mathrm{OT}[u][\epsilon] = 1\}$
4: $F_{\mathbb{R}} \leftarrow \{q_u \in Q : \mathrm{OT}[u][\epsilon] = 0\}$
5: **for** $q_u \in Q$ **do**
6: **for** $a \in \Sigma$ **do** $\delta(q_u, a) \leftarrow q_w \in Q : \mathrm{row}(ua) = \mathrm{row}(w)$
▷ Da q_u per a si va nello stato q_w se ua è nel componente con short prefix w
7: **end for**
8: **return** $\langle \Sigma, Q, q_\epsilon, F_A, F_R, \delta \rangle$

3.2.2 Discrimination tree

Definizione (Discrimination tree)**.** Un *discrimination tree* è un albero binario con radice definito come $DT = \langle N, n_0, E, \tau, L \rangle$ dove:

N è un insieme finito di nodi
$n_0 \in N$ è la radice dell'albero
$E \subseteq N \times N \times \mathbb{B}$ è l'insieme finito di archi[2]
$\tau : N \to \Sigma^*$ assegna le etichette ai nodi
$L \subseteq N$ è l'insieme di foglie

Un DT è in stretta connessione con l'ipotesi come mostrato in figura 3.1. C'è una corrispondenza biunivoca tra foglie nel DT e stati in H. Come si può vedere in figura 3.1 alcune delle transizioni in H sono disegnate in *bold* perchè corrispondono ai prefissi che fungono da short prefix: nella fattispecie si ha $Sp = \{\epsilon, a, ab, b\}$. Le foglie sono etichettate con gli short prefix contenuti in Sp. I nodi interni invece sono etichettati con i suffissi $\in V$ di C_I che consentono di *discriminare* le foglie cioè stati diversi del *DFA*. Il DT assicura l'esistenza di un discriminatore per ogni coppia di foglie (e quindi di short prefix e di stati) diverse, il **least common ancestor (*LCA*)**. L' LCA di due foglie a e b è incontrato nei rispettivi percorsi dai nodi verso la radice e rappresenta ,nei percorsi dalla radice verso a e b, il nodo in

[2]Si specifica per ogni arco se il secondo nodo è figlio destro o sinistro del primo nodo (rispettivamente 1 e 0)

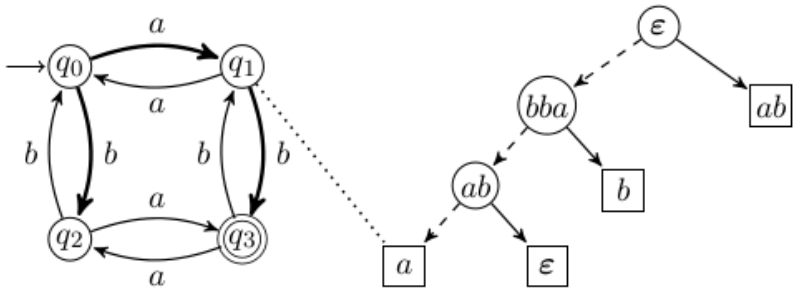


Figura 3.1: DFA ipotesi e possibile Discrimination Tree corrispondente

cui questi percorsi divergono come evidenziato in figura 3.2.
Un'operazione fondamentale è il *sift* di una stringa $x \in \Sigma^*$ nel DT che consente di *affondare* x all'interno dell'albero. Sia q un nodo interno etichettato con v ed A il *DFA* target, il sifting di x procede,partendo dalla radice, nel sottoalbero sinistro o nel sottoalbero destro di q a seconda del valore di λ^A(xv) (se è 0 si va nel sottoalbero sinistro). Questa procedura è ripetuta finchè una foglia è raggiunta. Ad esempio in figura 3.1 il *sift* della stringa aba —,che ha access sequence b, e che in H termina nello stato q_2 — tramite la valutazione nel *DFA* target A di $\lambda^A(aba \cdot \epsilon)$ che da 0 (per questo si va a sinistra) e di $\lambda^A(aba \cdot bba)$ che produce 1 (per questo si va a destra) arriva alla foglia con etichetta b (non casualmente, infatti b e aba sono transizioni che nell'ipotesi giungono nello stesso stato e quindi sono nella stessa classe di equivalenza).Il *sift* è descritto in dettaglio nell'algoritmo 12 e nella sottosezione 3.4.1.

3.2.3 Observation Pack

Definizione (Observation Pack)**.** Un Observation Pack è una tupla $\langle C_I, DT \rangle$

Da cui deriva il nome dell'algoritmo. Un Observation Pack è chiuso e completo se C_I è chiuso e completo.

3.3 Gestione del controesempio

3.3.1 Classificazione

Un controesempio è una stringa w $\in \mathcal{L} \oplus L(H)$ che viene ritornato dal *teacher* quando H e il target differiscono. w va sfruttato dal *learner* in qualche modo per produrre una nuova ipotesi H. Esistono essenzialmente due modi per farlo:

1. *Metodi Suffix-based.* Aggiungono uno o più suffissi del controesempio all'insieme di suffissi dell'algoritmo provocando la non chiusura.

2. *Metodi Prefix-based.* Aggiungono uno o più prefissi di un controesempio all'insieme di prefissi dell'algoritmo causando l'inconsistenza delle osservazioni, cioè una situazione di indeterminismo in H.

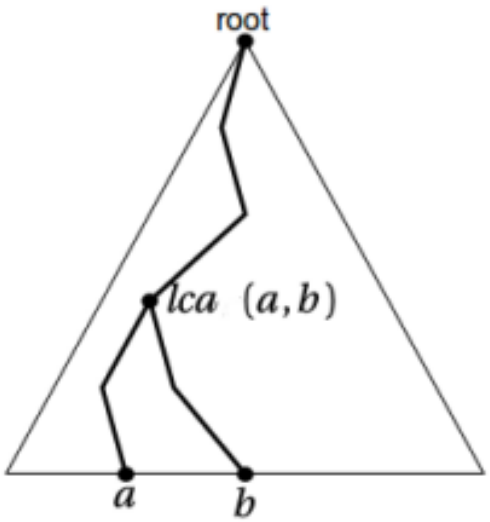


Figura 3.2: LCA di due nodi in un discrimination tree

ObP usa la prima strategia.Inoltre un'ulteriore classificazione sui metodi di gestione del controesempio viene effettuata in base a (1) se tutti o solo qualche suffisso (prefisso nel caso dei metodi *prefix-based*) del controesempio sono impiegati e (2) questi suffissi (prefissi) sono applicati a tutti o solo a qualche prefisso (nell'accezione di prefisso rappresentante uno stato dell'ipotesi). Esistono diverse versioni di *ObP* (e in generale anche per i metodi *suffix-based*) in base a quest'ulteriore classificazione:

- **AllGlobally**.
  Si aggiungono tutti i suffissi del controesempio all'insieme di suffissi V di ogni componente in C_I.
- **OneGlobally**
  Si individua e si aggiunge un singolo suffisso del controesempio e lo si aggiunge all'insieme di suffissi V di ogni componente in C_I.
- **OneLocally**
  Si individua e si aggiunge un singolo suffisso del controesempio ad un ben preciso componente.

In tutti e tre i casi l'aggiunta del suffisso (o dei suffissi) porterà alla non chiusura di almeno un componente e al conseguente *split* che produrrà un nuovo stato nell'ipotesi. In AllGlobally e OneGlobally è possibile far diventare l'insieme V globale dato che è uguale per ogni componente. Nella strategia OneLocally l'insieme V di suffissi differirà da componente a componente. Su come sia possibile individuare un singolo suffisso dal controesempio e l'esatto componente a cui bisogna aggiungere questo suffisso si rimanda al teorema 3.1. Una classificazione simile è possibile per i metodi *prefix-based*. La politica di gestione del controesempio è una differenza rilevante tra L* e *ObP*. L* gestisce il controesempio in maniera poco sofisticata: usa

un metodo *prefix-based* in cui tutti i prefissi del controesempio sono aggiunti alla tabella di osservazione:quindi è una strategia AllGlobally dato che tutti i prefissi del controesempio sono usati in combinazione con un insieme di suffissi globale. Questa strategia causa l'inconsistenza di alcuni prefissi e per risolverla si aggiunge un suffisso che a sua volta causa uno *split* dato che alcuni prefissi (prefissi RED) che prima erano equivalenti adesso non lo sono più e quindi devono corrispondere a due stati diversi. Lo svantaggio principale di questa strategia è che vengono aggiunti dei prefissi improduttivi che fanno aumentare il numero di *MQ*. Invece con una strategia che aggiunge un solo suffisso (o un solo prefisso) è necessario fare delle *MQ* per trovare il suffisso in questione dal controesempio ma il numero complessivo di *MQ* risulterà sempre minore complessivamente ad una strategia AllGlobally come quella usata da L* (qui si fa riferimento alla versione originale di L* in [2]. Esiste la variante OneLocally di L* in [28] che determina un unico prefisso del controesempio che genera inconsistenza, che viene risolta con l'aggiunta di un suffisso che causa uno *split* nella loro versione del discrimination tree).

3.3.2 Decomposizione del controesempio

Il seguente risultato fondamentale [43] garantisce che dato qualsiasi controesempio esiste sempre un suffisso che discrimina due prefissi nello stesso componente e ne causa di conseguenza lo *split*:

Teorema 3.1 (Decomposizione del controesempio)**.** *Sia H un'ipotesi, A il target, e $w \in \Sigma^+$ un controesempio ,cioè $\lambda^A(w) \neq \lambda^H(w)$. Allora esiste una decomposizione $\langle u, a, v\rangle \in \Sigma^* \times \Sigma \times \Sigma^*$ tale che $w = u{\cdot}a{\cdot}v$ e $\lambda^A(\lfloor u \rfloor_H a \cdot v) \neq \lambda^A(\lfloor ua \rfloor_H \cdot v)$.*

Innanzitutto bisogna precisare che il controesempio non può essere mai ϵ perchè $\lambda^A(\epsilon) = \lambda^H(\epsilon)$ che segue da $\epsilon \in Sp$ e dall'invariante (I2) (vedasi sottosezione 3.4.2) e ciò giustifica l'assunzione $w \in \Sigma^+$ che si fa nel teorema 3.1. Si osservi che u e $\lfloor u \rfloor_H$ sono stringhe che terminano nello stesso stato di H, lo stesso dicasi allora per ua e $\lfloor u \rfloor_H \cdot a$. Inoltre ua e $\lfloor ua \rfloor_H$ terminano nello stesso stato di H da cui segue transitivamente che pure $\lfloor u \rfloor_H \cdot a$ e $\lfloor ua \rfloor_H$ terminano nello stesso stato dell'ipotesi H (e quindi appartengono anche allo stesso componente dato che c'è corrispondenza biunivoca tra stati dell'ipotesi e componenti). Quanto detto è schematizzato nella figura 3.3. Ma il teorema 3.1 afferma che $\lambda^A(\lfloor u \rfloor_H a{\cdot}v) \neq \lambda^A(\lfloor ua \rfloor_H{\cdot}v)$ il che significa che $\lfloor u \rfloor_H \cdot a$ e $\lfloor ua \rfloor_H$ — che sono due stringhe che in H terminano nello stesso stato — nel target A terminano in due stati diversi. Quindi si è trovato il suffisso v che discrimina i due prefissi $\lfloor u \rfloor_H \cdot a$ e $\lfloor ua \rfloor_H$ e in più si sa che i due prefissi sono anche nello stesso componente.

Decomposizione del controesempio riformulata

In [50] è descritto un framework che permette di riformulare il teorema 3.1 in modo da facilitarne la comprensione della correttezza ed anche l'implementazione. Alcune funzioni si trovano nell'appendice B e qui sono date per scontate.

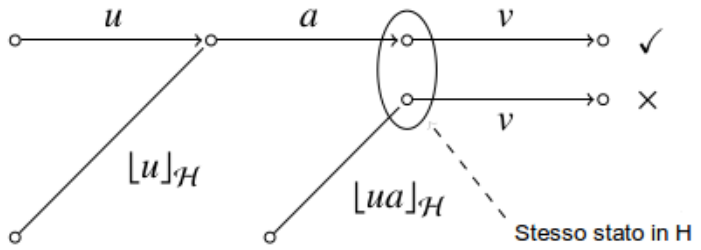


Figura 3.3: Sfruttare il controesempio

Sia dato un controesempio $w \in \Sigma^+$ tale che $|w| = m$. Dato che $\pi_H(w, m) \in Sp$ (si evince dalla definizione di w considerando che il controesempio è lungo m), dall'invariante (I2) (nella sottosezione 3.4.2) si ha $\lambda^A(\pi_H(w, m)) = \lambda^H(w)$ quindi $\alpha(m) = 1$. Inoltre dato che $\pi_H(w, 0) = w$ e w è un controesempio si ha $\lambda^A(\pi_H(w, 0)) \neq \lambda^H(w)$, quindi $\alpha(0) = 0$. Allora si può ricondurre il teorema 3.1 nel trovare un undice i tale che $\alpha(i) \neq \alpha(i + 1)$. Siccome $\alpha(0) = 0$ e $\alpha(1) = 1$ sicuramente ci sarà un indice i in cui il valore di α passa da 0 ad 1 e ciò dimostra la correttezza e l'esistenza del suffisso. Il teorema 3.1 può essere riformulato così:

Teorema 3.2 (Decomposizione del controesempio riformulata)**.** *Una politica di analisi del controesempio suffix-based può essere riformulata come il problema di, data una funzione* $\alpha : [0, m + 1) \to \mathbb{B}$ *con* $\alpha(0) = 0$ *e* $\alpha(1) = 1$*, trovare un indice i,* $0 \leq i < m$*, soddisfacente* $\alpha(i) \neq \alpha(i + 1)$

Una volta trovato un indice i siffatto la decomposizione è u=$w_{[0,i)}$, a=w_i, v=$w_{[i+1,m)}$

3.3.3 Metodi di decomposizione del controesempio

Esistono diverse strategie per la ricerca del suffisso (o per i metodi *prefix-based*) all'interno del controesempio. I parametri da tenere in considerazione nella valutazione di tali strategie sono il numero di *MQ* effettuate e la dimensione del suffisso trovato. Infatti per valutare $\alpha(i)$ è necessaria una *MQ* quindi l'impiego di un'euristica o di una politica che porta velocemente alla scoperta dell'indice i e quindi del suffisso permette di risparmiare il numero di *MQ* da effettuare al *teacher*. La dimensione dei suffissi[3] è anch esso un parametro molto importante in quanto il suffisso trovato verrà aggiunto all'insieme dei suffissi dei componenti o del componente rispettivamente in OneGlobally e OneLocally, e anche al DT (vedasi algoritmo 13). Quando si navigherà il DT , per esempio durante un sift, verranno fatte delle *MQ* in cui parte della stringa è composta dall'etichetta di un nodo interno costituita da un suffisso; quindi il costo della *MQ* crescerà linearmente con la dimensione del suffisso.
La strategia più semplice è quella che effettua una ricerca lineare in ordine discendente cioè partendo da $i = m - 1$ che termina quando un valore i tale che $\alpha(i) = 0$

[3]All'interno di un controesempio possono esistere più suffissi discriminatori, cioè molteplici indici i che permettono una decomposizione

è incontrato. Questo metodo assicura di trovare il suffisso più breve ma nel caso peggiore richiede m-1 *MQ*. Anche se un costo lineare alla dimensione del controesempio sembra accettabile negli scenari reali non sempre lo è. In un contesto reale spesso capita di non avere la possibilità di effettuare un *EQ* che va approssimata con un certo numero di *MQ* nell'ambito del *PAC-learning*. In questo scenario il controesempio tornato è una stringa di lunghezza non ottimale che può avere una lunghezza significativa. Esistono dei metodi che permettono di diminuire il numero di *MQ* anche se spesso bisogna rinunciare alla lunghezza minima del controesempio ed accontentarsi di una lunghezza ottima.

Algoritmo 8: Binary-Search

Input: A counterexample w with $|w| = m$
Output: Index i : $\alpha(i) \neq \alpha(i+1)$

```
low ← 0
high ← m
while high − low > 1 do
    mid ← ⌊(low+high)/2⌋
    if α(mid) = 0 then
        low ← mid
    else
        high ← mid
    end if
end while
return low
```

Algoritmo 9: Exponential-Search

Input: A counterexample w with $|w| = m$
Output: Index i : $\alpha(i) \neq \alpha(i+1)$

```
low←0, high←m, ofs←1, found←false
while high − ofs > 0 and ¬found do
    if α(high − ofs) = 0 then
        low ← high − ofs
        found ← true
    else
        high ← high − ofs
        ofs ← 2 · ofs
    end if
end while
return Binary-Search(α, low, high)
```

Binary search

Il metodo descritto nell'algoritmo 8 è suggerito in [43]. Esso impiega la ricerca binaria per trovare una decomposizione valida. Il numero di *MQ* necessarie è $\lceil log_2(m) \rceil$ sempre, cioè non esiste un caso migliore ma il numero di *MQ* è fisso perchè è necessario testare il valore di $\alpha()$ per due indici i contigui e ciò avviene solo alla fine della ricerca. La dimensione del suffisso trovata può anche essere molto più grande di quella minima.

Exponential Search

Il metodo descritto qui e tutti quelli a seguire sono descritti in [50]. La ricerca binaria ha lo svantaggio evidente che può essere tornato un controesempio relativamente lungo: se il primo valore per *mid* testato è tale che $\alpha(mid) = 1$ il suffisso risultante sarà di lunghezza almeno $\lceil m/2 \rceil$[4]. Con *exponential search* si testano $\alpha(m - 2^0), \alpha(m - 2^1), \alpha(m - 2^2)$ eccetera, finchè non si trova un intervallo [l,h) per cui $\alpha(l) = 0$ e $\alpha(h) = 1$ ed allora si chiamerà il metodo che usa la ricerca binaria descritto nell' algoritmo 8 sui due indici l ed h. Nel caso peggiore(*exponential search*

[4]La funzione $\alpha()$ non è necessariamente monotona quindi anche se $\alpha(mid) = 1$ ci può essere un indice $i > m$ per il quale $\alpha(i) = 0$ e quindi esserci un suffisso più breve

non riesce a restringere l'intervallo cioè l resta 0 ed h resta m) questo metodo richiede $2\lfloor log_2(m) \rfloor$ *MQ* , $\lfloor log_2(m) \rfloor$ per *exponential search* e altrettante per la ricerca binaria.Nella maggior parte dei casi l'algoritmo termina molto prima e nel caso migliore ($\alpha(m-1) = 0$) si effettua una singola *MQ*. Per come funziona quest algoritmo favorisce il ritrovamento di suffissi più brevi rispetto alla ricerca binaria.
Exponential Search è presentato in dettaglio nell'algoritmo 9.

Partition Search

Exponential search può terminare velocemente e individuare suffissi molto brevi ma nel caso che anche poche posizioni di $\alpha(m - 2^i) = 1$ può essere svantaggiosa per via della rapida (anche per i piccolo),esponenziale, crescita dell'intervallo. Un approccio più bilanciato è quello di patizionare α in $\lceil log_2(m) \rceil$ intervalli, ognuno di lunghezza $s = \lfloor \frac{m}{log_2 m} \rfloor$. Poi i valori testati saranno $\alpha(m - s), \alpha(m - 2s)$ eccetera, finchè un intervallo [l,h) soddisfacente $\alpha(l) = 0$ e $\alpha(h) = 1$ è trovato. Quest intervallo sarà poi sottoposto al metodo di ricerca binaria per trovare un indice i in esso. In *exponential search* a causa del passo esponenziale questo intervallo poteva risultare molto grande, in *partition search* è di dimensione s. Quindi il numero di *MQ* da effettuare nel caso peggiore con *partition search* è $log_2(s) = log_2(\lfloor \frac{m}{log_2 m} \rfloor)$ per via della ricerca binaria da eseguire sull'intervallo trovato più $\lfloor log_2(m) \rfloor$ (il numero di partizioni) *MQ* per trovare l'intervallo. Quindi sono necessarie $\mathcal{O}(log_2(m))$ *MQ*. *Partition Search* è presentato in dettaglio nell'algoritmo 10. Si osservi come il costo della ricerca binaria sia fisso (cioè è presente anche nel caso migliore) e dipendente da m. Quindi ci si aspetta che questo metodo funzioni meglio per controesempi non troppo grandi (m piccolo).

Eager Search

Eager Search è una variante del metodo di ricerca binaria. Quest ultimo richiede sempre — cioè non c'è un caso migliore o medio ma il numero di *MQ* è sempre lo stesso — $log_2(m)$ *MQ*. Come accennato in precedenza il motivo è che il solo valore $\alpha(i)$ da solo non è sufficiente ma è necessaro testare anche $\alpha(i + 1)$[5]. La soluzione proposta in *Eager Search* è di testare ogni volta il valore di α sia per i che per i+1 e vagliare se differiscono o detto in maniera più succinta che il valore di β(vedasi B.1.2) sia uguale ad 1. Nel caso peggiore siccome ogni valutazione di β richiede 2 *MQ* e il numero di valutazioni da fare è lo stesso della ricerca binaria (nel caso peggiore) sono necessarie $2log_2(m)$ *MQ*. Tuttavia nel caso migliore solo 2 *MQ* sono sufficienti e la ricerca può terminare molto prima di quella binaria. Questa strategia è affetta dallo stesso problema della ricerca binaria per quanto riguarda la dimensione dei suffissi tuttavia è possibile utilizzare *Eager Search* in luogo della ricerca binaria sia in *exponential search* che in *partition search*.

[5]perchè dal teorema 3.2 si deve trovare un indice i per cui $\alpha(i) \neq \alpha(i + 1)$

Algoritmo 10: Partition-Search

Input: A counterexample w with $|w| = m$
Output: Index i : $\alpha(i) \neq \alpha(i+1)$

```
step ← ⌊m / log2(m)⌋ low ← 0, high ← m
found ← false
while high−step>low and ¬found do
    if α(high − step) = 0 then
        low ← high − step
        found ← true
        break
    else
        high ← high − step
    end if
end while
return Binary-Search(α, low, high)
```

Algoritmo 11: Eager-Search

Input: A counterexample w with $|w| = m$
Output: Index i : $\beta(i) = 1$

```
low ← 0, high ← m − 1
while high > low do
    mid ← ⌊(low+high)/2⌋
    if β(mid) = 1 then
        return mid
    else if β(mid) = 0 then
        low ← mid + 1
    else
        high ← mid − 1
    end if
end while
return low
```

3.4 L'algoritmo

3.4.1 Funzionamento

ObP nella fase d'inizializzazione crea il componente corrispondente allo stato iniziale $C_\epsilon = \langle \Sigma \cup \{\epsilon\}, \epsilon, \epsilon, \emptyset \rangle$ ed il DT è costituito solo dalla radice quindi $DT = \langle \{n_\epsilon\}, n_\epsilon, \emptyset, \tau(n_\epsilon) = \epsilon, \{n_\epsilon\} \rangle$ come si può vedere nell'algoritmo 15. Dopodichè si chiama la funzione closePack() (vedasi algoritmo 14) che ha lo scopo di rendere chiusi e completi C_I e modificare il DT in modo da rappresentare la stessa ipotesi rappresentata da C_I. Si completano i C_I e poi si ricerca un componente C_u (questo in generale e non solo nella fase di inizializzazione dove l'unico componente è C_ϵ) e un prefisso $u' \in C_u$ per cui row$(u) \neq$ row(u'). Si seleziona il suffisso v per cui $OT[u][v] \neq OT[u'][v]$ e si divide C_u chiamando la funzione *split* (algoritmo 13) che genererà un nuovo componente $C_{u'}$ e lo ritornerà a closePack(). In split() alcuni dei prefissi $U \in C_u$,che indichiamo con $x \in U$, ed esattamente quelli per cui $OT[x][v] \neq OT[u][v]$ vengono fatti migrare in $C_{u'}$ che nel frattempo è stato creato(cioè eliminati dal componente C_u ed immessi in $C_{u'}$). Si procede poi a modificare di conseguenza anche il DT: si individua il nodo foglia con etichetta u e lo si splitta nel senso che questo nodo foglia diventa un nodo interno con etichetta v (il suffisso, cioè il discriminatore) ed i suoi figli saranno due nodi foglia uno con etichetta u' e l'altro con etichetta u come si può apprezzare in figura[6] 3.5b (si stabilisce qual è il figlio sinistro e quale il destro in base al risultato della MQ $\lambda^A(uv)$ nel target A, se questa fa zero u è il figlio sinistro del nuovo nodo con etichetta v altrimenti è il figlio destro). Si puntualizza che lo *split* non necessita di MQ aggiuntive e che l'insieme V del nuovo componente creato con lo *splitting* è lo stesso dell'insieme V del componente da cui ha origine. A questo punto split() ritorna il componente in

[6]Per coerenza con il discorso fatto si consideri $v = v$, $b = u$,$b' = u'$

questione a closePack() che memorizza il componente trovato $C_{u'}$ in W[7]. Finchè W non è vuoto si deve estrarre ed eliminare da W un componente , nella fattispecie $C_{u'}$, e concatenare l'access sequence del componente u' ad ogni simbolo di Σ e per ogni stringa ottenuta effettuare il *sift* (algortimo 12). Questo serve per assicurare in ogni caso la possibilità di costruire una nuova ipotesi H in modo che per ogni nuovo componente aggiunto per ogni simbolo dell'alfabeto sia ben definito lo stato che viene raggiunto. Adesso si descrive in dettaglio come avviene il sift di una stringa x nel DT: si parte dalla radice e si effettua la query nel target A $\lambda^A(x \cdot \tau(n_0))$,se da esito 0 si procede verso il sottoalbero sinistro altrimenti verso quello destro. Si continua in questa maniera finchè due situazioni possono presentarsi:

- Un nodo foglia $n_{x'}$ è incontrato. Significa che la stringa x in H porta nello stato rappresentato dallo foglia in cui si è arrivati: detto x' lo short prefix di questa foglia si ha che $x \in [x']_{\simeq_{OT}}$. Quindi si dovrà anche provvedere ad aggiungere x all'insieme U del componente $C_{x'}$. L'assegnazione della stringa come nuovo prefisso a un componente non è necessariamente definitiva perchè successivamente può accadere che la scoperta di un nuovo suffisso renda non chiuso un componente e per assicurare la chiusura si debba effettuare uno split che divide il componente in due parti e causando così possibilmente anche la migrazione di alcuni prefissi nel nuovo componente.

- Un nodo interno n_z con un solo figlio e con etichetta z è incontrato ma $\lambda^A(xz)$ suggerisce di andare verso il percorso dove n_z non ha figli. In questo caso deve avvenire la creazione di una nuova foglia n_x con etichetta x come figlio del nodo n_z. Conseguentemente deve essere aggiunto C_x a C_I. L'insieme di suffissi V di C_x sarà inizializzato in OneLocally con i suffissi trovati lungo il percorso nel DT per arrivare alla nuova foglia e con OneGlobally e AllGlobally con i suffissi globali dell'algoritmo.

Nel caso in cui sift() torna un nuovo componente a closePack() quest ultimo va aggiunto in W. Si ripete il procedimento finchè W non è vuota. Inoltre l'algoritmo deve rendere C_I completo (la non completezza è generata dal *sifting* e anche dall'eventuale non chiusura tranne alla prima iterazione dove è causata dal suffisso trovato nella decomposizione del controesempio). Si procede poi alla generazione dell'ipotesi H come visto nell'algoritmo 7 e si effettua un EQ cui il *teacher* può rispondere dato che si assume che appartenga alla classe dei MAT. Se il target A e l'ipotesi H sono equivalenti l'algoritmo termina garantendo che H sia il DFA minimo e l'equivalenza di A e H altrimenti verrà tornato un controesempio che verrà sfruttato per trovare un suffisso discriminante due prefissi appartenenti allo stesso componente (e con uno dei prefissi ,diciamo x, : $x \in Sp$). Se si usa OneLocally si aggiunge il suffisso all'insieme V di C_x, con OneGlobally si aggiunge il suffisso all'insieme V che sarà globale per tutte le componenti. Con la strategia AllGlobally non è necessario effettuare la decomposizione del controesempio in quanto si aggiungeranno tutti i suffissi del

[7] (solo quando closePack() viene chiamato per la prima volta in assoluto ,immediatamente dopo la fase di inizializzazione, a W deve essere aggiunto anche C_ϵ (per esempio una coda)

controesempio all'insieme di suffissi V globale e comune a tutte le componenti ma è chiaramente una strategia controproducente. L'aggiunta del suffisso v garantisce la non chiusura ad ogni " generazione " e quindi l'aggiunta di almeno un nuovo componente, e quindi stato, mediante lo *split*.

Algoritmo 12 OBP-SIFT

Input: a $DT = \langle N, n_0, E, \tau, L \rangle$, C_I , new prefix $u \in \Sigma^*$
Output: A new component or "OK"

1: $n = n_0$
2: **while** $n \notin L$ **do**
3: $v \leftarrow \tau(n)$
4: $o \leftarrow MQ(uv)$ ▷ $MQ(uv) = \lambda^A(uv)$
5: **if** $\exists(n, n', o) \in E$ **then** ▷ Se n ha un figlio nella direzione indicata da o
6: $n \leftarrow n'$
7: **else** ▷ Il nodo n non ha figli nella direzione indicata da o
8: Create new node n_u
9: $N \leftarrow N \cup \{n_u\}, \quad E \leftarrow E \cup \{(n, n_u, o)\}$
10: $\tau(n_u) = u$
11: Create component C_u
▷ Aggiungi a C_u lo short prefix u e i suffissi secondo la strategia usata
12: **return** C_u
13: **end while**
▷ Se si è qui significa che si è arrivati a una foglia
14: $u' = \tau(n)$ ▷ Si prende lo short prefix del nodo n
15: add u to $C_{u'}$
▷ aggiungere il prefisso al componente solo se il prefisso non è già presente
16: **return** "OK'

Inoltre a differenza di L* in *ObP* è possibile sfruttare più volte lo stesso controesempio. In L* tutti i prefissi del controesempio compreso il controesempio stesso vengono aggiunti alla tabella di osservazione e la nuova ipotesi generata è consistente con essi e quindi alla successiva " generazione " il controesempio precedente non è più riutilizzabile perchè difatti non è più in $\mathcal{L} \oplus \mathrm{L}(H)$. In *ObP* invece viene aggiunto un solo suffisso del controesempio e quindi potrebbe accadere che alla successiva " generazione " sia ancora un controesempio utilizzabile per trovare un altro suffisso che produca la non chiusura. Ciò può fare risparmiare molte *EQ* al costo di una *MQ* aggiuntiva per ogni " generazione " (perchè è necessario testare se il controesempio è ancora tale).

3.4.2 Correttezza

La correttezza dell' *ObP* scaturisce dal mantenimento di tre ***invarianti d'apprendimento*** :

Algoritmo 13 OBP-SPLIT

Input: a $DT = \langle N, n_0, E, \tau, L\rangle$, C_I , a component $C_{u_0} = \langle U, u_0, V, OT\rangle$, a prefix $u \in U$, a suffix $v \in V : OT[u_0][v]) \neq OT[u][v]$
Output: A new component

Create component $C_u = \langle \emptyset, u, V, \emptyset\rangle$
for $u' \in U$ **do**
    if $OT[u_0][v] \neq OT[u'][v]$ **then**
        Transfer u' from C_{u_0} to C_u ▷ Trasferire significa eliminare da C_{u_0}
end for
▷ Adesso a seguire le modifiche da apportare al discrimination tree
Let $n \in L$ where $\tau(n) = u_0$ ▷ Seleziona la foglia con etichetta u_0
$\tau(n) = v$ ▷ Modifica l'etichetta del nodo con quella del suffisso discriminante
Create new node n_u
Create new node n_{u_0}
$N \leftarrow N \cup \{n_u, n_{u_0}\}$
$\tau(n_u) = u$
$\tau(n_{u_0}) = u_0$
$E \leftarrow E \cup \{(n, n_u, OT[u][v])\} \cup \{(n, n_{u_0}, OT[u_0][v])\}$
return C_u

Algoritmo 14 OBP-CLOSEPACK

Input: a observation pack $\langle DT, C_I\rangle$
Output: ipotesi H

$W \leftarrow \emptyset$ (only in first call ever $W \leftarrow \{C_\epsilon\}$)
while C_I is unclosed or incomplete **do**
    Complete C_I
    Let $C_{u_0} = \langle U, u_0, V, OT\rangle$ with $u \in U, v \in V : OT[u_0][v] \neq OT[u][v]$
    $C_u = \text{OBP-SPLIT}(DT, C_I, C_{u_0}, u, v)$
    $W \leftarrow W \cup \{C_u\}$
    while $W \neq \emptyset$ **do**
        $C_u \leftarrow poll(W)$
    ▷ Estrai un componente secondo una qualche politica ad esempio FIFO
        for $a \in \Sigma$ **do**
            $C = \text{OBP-SIFT}(DT, C_I, ua)$
            if $C \neq$ "OK" **then**
                $W \leftarrow W \cup C$
        end for
    end while
end while
$H = \text{OBP-BUILDAUTOMATON}(C_I)$
return H

Algoritmo 15 OBSERVATION PACK

Input: alfabeto Σ
Output: minimal DFA H : L(H)=$\mathcal{L}$

1: $C_\epsilon = \langle \Sigma \cup \{\epsilon\}, \epsilon, \epsilon, \emptyset \rangle$
2: $C_I = C_\epsilon$
3: $DT = \langle \{n_\epsilon\}, n_\epsilon, \emptyset, \tau(n_\epsilon) = \epsilon, \{n_\epsilon\} \rangle$
4: **loop**
5: H = OBP-CLOSEPACK(DT, C_I)
6: w = EQ(H)
7: **if** w = "OK" **then**
8: **return** H
9: **end if**
10: **if** AllGlobally **then**
11: Add all suffixes of w to V of all components in C_I
12: **continue**
13: **end if**
14: Split w with a decomposition method in uav : $\lambda^A(\underbrace{\lfloor u \rfloor_H a}_{p} v) \neq \lambda^A(\underbrace{\lfloor ua \rfloor_H}_{u'} v)$ tale che $p \in C_{u'}$
15: **if** OneGlobally **then**
16: Add v to V of all components $\in C_I$
17: **end if**
18: **if** OneLocally **then**
19: Add v to V of $C_{u'}$
20: **end if**
21: **end loop**

(I1) $u \neq u' \in Sp$ corrispondono a stati differenti nel *DFA* target A cioè $A[u] \neq A[u']$.

(I2) Ogni stato q in H è accettante se e solo se "parsando" $\lfloor q \rfloor_H$ nel target A si termina pure in uno stato accettante. In simboli: $\forall q \in Q^H : q \in F^H_{\mathbb{A}} \Leftrightarrow f_{Sp}(q) \in F^A_{\mathbb{A}}$

(I3) Transizioni in H puntano allo stato corretto nel target, se quest ultimo è già stato scoperto dal *learner*: $\forall q \in Q^H, a \in \Sigma$ si ha $\delta^A(f_{Sp}(q), a) \in A[Sp] \implies f_{Sp}(\delta^H(q, a)) = \delta^A(f_{Sp}(q), a)$

E' evidente che la scoperta di nuovi short prefix in Sp mantenendo la condizione (I1) porterà alla scoperta di tutti gli stati nel target A (dato che sono finiti per il teorema di Myhill-Nerode descritto in A.1). Le condizioni (I2) ed (I3) garantiscono che f_{Sp} è un isomorfismo .In L* vi è una violazione della condizione (I1) perchè può accadere che più di uno short prefix (più di uno stato RED) rappresentino lo stesso stato nel target. Il requisito di consistenza tuttavia garantisce che uno qualsiasi di questi può essere scelto come rappresentativo dello stato target.

Si dimostra che le tre *invarianti* sono possedute dall'algoritmo *ObP*:

(I1) Siano u e $u' \in Sp$ e $u \neq u'$, allora essendo in due componenti diversi esiste un suffisso v che distingue u e u' cioè tale che $\lambda^A(uv) = OT[u][v] \neq \lambda^A(u'v) = OT[u'][v]$. Quindi si ha anche $\lambda^A_{A[u]}(v) \neq \lambda^A_{A[u']}(v)$ e quindi $A[u] \neq A[u']$

(I2) Sia $u = \lfloor q \rfloor_H$ l'access sequence di uno stato $q \in Q^H$. Si ha allora che $q \in F^H_{\mathbb{A}} \iff$[8] $OT[u][\epsilon] = 1$. Dato che $OT[u][\epsilon] = \lambda^A(u \cdot \epsilon) = \lambda^A_{A[u]}(\epsilon)$ si può concludere che $q \in F^H_{\mathbb{A}} : u = \lfloor q \rfloor_H \Leftrightarrow A[u]$[9] $\in F^A_{\mathbb{A}}$

(I3) Sia dato uno stato $q \in Q^H$ tale che $u = \lfloor q \rfloor_H$, il successore di q per il simbolo $a \in \Sigma$ è determinato con il *sifting* di ua nel DT. Sia lo stato con access sequence u' il risultato di questa operazione di *sifting*, si definisce allora $\delta^H(q, a) = H[u']$. Se però nel target A questa transizione è errata cioè $\delta^A(A[u], a) \neq A[u']$, si sa per certo che il successore reale sul simbolo a cioè $A[ua]$ non è ancora stato scoperto, quindi (I3) è preservata: si noti che *siftando* ua nell'albero, arrivando ad u' si escludono definitivamente gli altri stati scoperti fino a quel punto $A[Sp \backslash \{u'\}]$ come possibili successori sul simbolo a di $A[u]$.

Ciò dimostra la correttezza dell' *ObP* ma tramite quanto detto sopra si dimostra solo che H ed A sono isomorfi senza assumere la canonicità per H. Infatti in generale A non è un *DFA* minimo e secondo quanto detto finora non si può dedurre che

[8] Nel caso della strategia OneGlobally è lampante capire che in ogni componente ci sarà sempre il suffisso ϵ essendo questo già presente in fase d'inizializzazione. Utilizzando OneLocally accade lo stesso perchè i suffissi da aggiungere ad un nuovo stato (rappresentato da una access sequence e da un componente) sono quelli incontrati durante il *sifting* dell'access sequence nel discrimination tree e quindi si passa sempre dalla radice che contiene il suffisso ϵ

[9] è uguale a $f_{Sp}(q)$

H sia minimo ma solo isomorfo ad A. Inoltre a causa del maccanismo di gestione del controesempio non vi è la garanzia che l'insieme di suffissi sia *suffix-closed* dato che viene aggiunto solo un suffisso del controesempio. Quanto detto causa anche la generazione di ipotesi intermedie non minime. Tuttavia applicando l'euristica di utilizzare più volte lo stesso controesempio finchè da esso non è più possibile estrarre un suffisso discriminante è garantito che l'ipotesi finale è un *DFA* canonico [49, p. 25]. Esiste un'ottimizzazione che tramite il concetto di *semantic suffix closdness* ,sempre in [49] , assicura che anche le ipotesi intermedie prodotte siano canoniche ed il meccanismo che permette di scoprire dei nuovi suffissi da quelli esistenti in modo da ottenere la *semantic suffix closdness* permette di risparmiare anche delle *EQ* e *MQ* perchè permette di scoprire nuovi suffissi discriminanti e nuovi stati senza costi aggiuntivi in termini di *EQ* ed *MQ* (vi è però il tempo di esecuzione aggiuntivo dell'algoritmo per garantire la *semantic suffix closdness*).

3.4.3 Complessità computazionale

Il tempo di esecuzione degli algoritmi di *active learning* è quasi sempre un polinomio di grado piccolo dipendente dalla dimensione dell'alfabeto k, il numero di stati del *DFA* canonico equivalente ad $\mathcal{L}$ che qui si indica con n , e la lunghezza del controesempio m con cui si indica la lunghezza del più lungo controesempio tornato dal *teacher*. Il tempo di esecuzione è quasi sempre dominato dal tempo speso nell'effettuare *MQ* ed *EQ* [10] e quindi contare quest ultime può essere sufficiente per analizzare le prestazioni dell'algoritmo. In quest'ottica in [26, p. 20] si trova un'analisi dell' *ObP* nelle sue varie forme per le Mealy machines confrontando i risultati con L*. Questi risultati vengono qui applicati ai *DFA* e riportati nella tabella 3.1. Dato che per tutti gli algoritmi il numero di *EQ* è limitato da $\mathcal{O}(n)$ l'analisi delle *EQ* viene omessa. Infine si noti come l'analisi venga effettuata nel caso peggiore.
Per tutti gli approcci il numero di suffissi necessari per distinguere tutti gli stati è limitato da n dato che ogni aggiunta di un suffisso determina almeno uno *split* e l'individuazione di almeno un nuovo stato (stati che sono al massimo n). Il numero di prefissi in tutte le componenti per *ObP* è al più $n \cdot k$ perchè al più si hanno n componenti (n stati), se da ogni stato si hanno k transizioni il numero di transizioni totali sarà $n \cdot k$ (i prefissi di uno stato sono le transizioni che finiscono in quello stato). La dimensione della tabella di osservazione o meglio la somma delle dimensioni di tutte le tabelle di osservazione di ogni componente è allora $nk \cdot n$ (numero prefissi $\cdot$ numero suffissi). Per AllGlobally si aggiungono tutti gli m suffissi di ogni controesempio, e lo si fa n volte quindi si avranno $n \cdot m$ suffissi con una tabella di dimensione $nk \cdot nm$. Per OneLocally e OneGlobally vi è da aggiungere il numero di query fatte durante la ricerca del suffisso che nel caso della ricerca binaria è $log_2(m)$ effettuate n volte. Per la dimensione della tabella di L* si faccia riferimento alla sottosezione 2.5.2.

[10] Talvolta si conta anche il numero di simboli contenuto in tutte le stringhe per cui si effettua una *MQ* per pesare il costo di una *MQ* dato che è lineare con la lunghezza

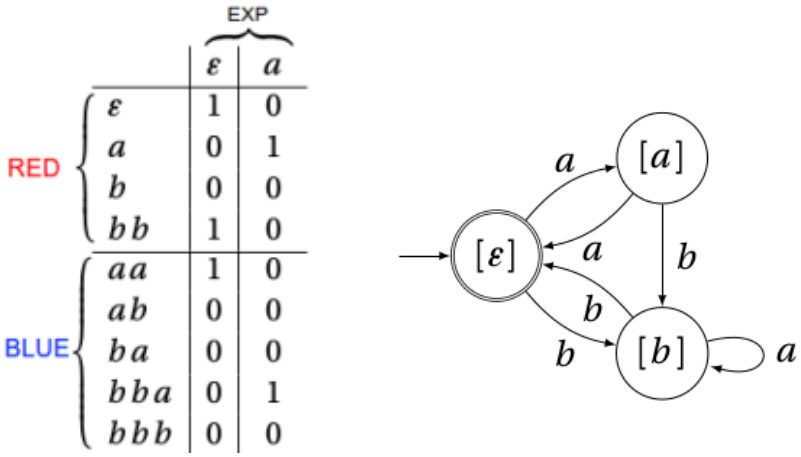
Algoritmo	Dimensione tabella	Membership queries
L*	$(k+1)\cdot(n+m(n-1))\cdot n$	$\mathcal{O}(n^2km)$
AllGlobally	$nk\cdot nm$	$\mathcal{O}(n^2km)$
OneLocally	$nk\cdot n$	$\mathcal{O}(n^2k)$
OneGlobally	$nk\cdot n$	$\mathcal{O}(n^2k)$

Tabella 3.1: Complessità delle membership queries per le differenti varianti di ObP

Tuttavia i risultati evidenziati nella tabella 3.1 ed anche quelli omessi per le *EQ* costituiscono solo un limite superiore, il lavoro sperimentale in [26, p. 22-23] mette meglio in evidenza le differenze tra i vari algoritmi. OneLocally è l'algoritmo con il minore numero di *MQ* dato che il suffisso discriminante è aggiunto solo ad un componente e quindi sarà necessario completare solo la tebella di osservazione di questo componente ,e nonostante OneLocally e OneGlobally abbiano lo stesso ordine di grandezza per il numero di *MQ* fatte al crescere dei parametri la differenza in termini di *MQ* è notevole. In OneLocally tuttavia come detto in [26, p. 23] il numero delle *EQ* aumenta drasticamente rispetto a OneGlobally anche se in numero pur sempre limitato dal numero di stati del *DFA* minimo equivalente ad $\mathcal{L}$. L* effettua un numero di *EQ* quasi sempre minore (seppur di poco) di OneGlobally ma un numero di *MQ* sempre molto maggiore sia a OneLocally che a OneGlobally. AllGlobally consente di effettuare un numero di *EQ* quasi sempre leggermente inferiore anche ad L* ma il numero di *MQ* è sempre molto elevato anche rispetto ad L*.

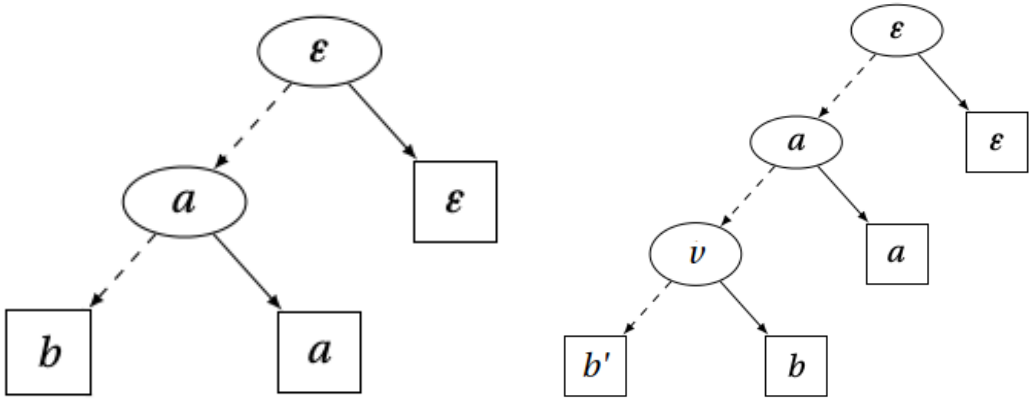
3.4.4 Discrimination tree vs Tabella di Osservazione localizzata

In figura 3.4a vi è la rappresentazione della tabella di osservazione per l'algoritmo L* per il *DFA* in figura 3.4b. I prefissi RED distinti tra di loro $\{\epsilon, a, b\}$ (cioè con row() distinte tra di loro) sono gli short prefix delle classi di equivalenza che hanno una corrispondenza con gli stati del *DFA*. I prefissi BLUE sono invece delle stringhe appartenenti ad una delle classi di equivalenza (un prefisso BLUE x termina nello stato rappresentato da uno stato RED r ed appartiene alla sua classe di equivalenza se row(r)=row(x)). Il *DT* è una rappresentazione della tabella di osservazione scevra da ridondanze come si può apprezzare in figura 3.5a. Infatti a parte lo short prefix superfluo bb, nella tabella di osservazione di L* non tutti i suffissi sono necessari per distinguere le varie classi di equivalenza (gli stati RED). Ad esempio il singolo suffisso ϵ da solo è sufficiente per distinguere $[\epsilon]$ dalle altre classi di equivalenza. Lo scopo di un *DT* è di eliminare queste ridondanze ed organizzare le osservazioni delle passate *MQ* in una maniera efficiente in modo da consentire efficientemente il *sifting* di una stringa. Nella tabella 3.2 vi è l'insieme di componenti di *ObP* corrispondente



(a) (b)

Figura 3.4: DFA ipotesi e corrispondente tabella di osservazione in L*



(a) (b)

Figura 3.5: Discrimination Tree e split di uno stato

alla tabella di osservazione per OneGlobally anche se è solo un esempio e molti dei prefissi presenti in L* potrebbero non essere presenti in *ObP*.

C_ϵ	ϵ	a
ϵ	1	0
aa	1	0
bb	1	0

C_a	ϵ	a
a	0	1
bba	0	1

C_b	ϵ	a
b	0	0
ab	0	0
ba	0	0
bbb	0	0

Tabella 3.2: Insieme di componenti

Sia il DT che C_I sono rappresentativi dell'ipotesi e quindi vi deve essere in ogni momento coerenza anche tra DT e C_I oltre che con l'ipotesi. Infatti per ogni foglia nell'albero c'è un corrispondente componente in C_I, l'etichetta della foglia corrisponde all'access sequence di quel componente. Entrambe le strutture dati sono rappresentative dell'ipotesi. A questo punto ci si potrebbe chiedere perchè utilizzare sia un DT che C_I per rappresentare l'ipotesi. Le considerazioni da fare differiscono tra OneLocally e OneGlobally. Quando si fa il sift di una stringa che termina in un nodo foglia si aggiungerà quella stringa al componente corrispondente a quel nodo foglia come detto nella sottosezione 3.4.1 .Tuttavia in OneGlobally l'insieme V di suffissi è globale e non coincide con i suffissi (che costituiscono un sottoinsieme di tutti i suffissi globali V),etichette dei nodi interni, incontrati durante la navigazione del DT per arrivare alla foglia summenzionata. Quindi vi è l'evenienza che tramite il *sift* di una stringa venga introdotta un'ulteriore non chiusura (ulteriore perchè sicuramente la non chiusura viene prodotta in prima istanza dal suffisso estrapolato dal controesempio) e conseguente *split*. Se non avessimo l'insieme di componenti sarebbe arduo esaminare la non chiusura nello scenario appena descritto da cui l'esigenza di C_I. Si noti inoltre che quanto appena descritto è il motivo per cui il numero di EQ non coincide ma è minore del numero degli stati del *DFA* minimo equivalente ad $\mathcal{L}$; infatti ad ogni " generazione " si possono avere più di uno *split* (ogni *split* consente di determinare un nuovo stato ed avvicinarci più velocemente alla soluzione) proprio per il motivo appena descritto. In OneLocally invece i suffissi di un componente C_x coincidono esattamente con i suffissi incontrati durante il *sifting* di x nel DT. Quando si effettua il sifting di un'altra stringa che tramite il DT si scopre essere in $[x]_{\simeq_{OT}}$, dato che giunge nel nodo foglia che ha $\tau(n_x) = x$, i suoi suffissi sono esattamente quelli incontrati durante la navigazione del DT e quindi non può essere introdotta non chiusura. Quindi in OneLocally si ha esattamente uno *split* ad ogni " generazione " ed il numero di EQ coincide con il numero di stati del DFA target minimizzato. In realtà non è così per via dell'ottimizzazione descritta in 3.4.1 che consente di riutilizzare più volte lo stesso controesempio e risparmiare delle EQ, ma comunque il numero di EQ in OneLocally sarà maggiore del numero di EQ in OneGlobally. Va detto che il controllo sulla non chiusura che si effettua in CLOSEPACK alla riga 14.2 può essere eliminato in OneLocally(il controllo va fatto solo la prima volta in assoluto che CLOSEPACK è chiamato, per le restanti generazioni la non chiusura si verificherà solo per via del suffisso del controesempio).

Riassumendo in OneLocally si potrebbe fare a meno dell'insieme di componenti ed utilizzare esclusivamente il DT. Tuttavia utilizzare anche l'insieme di componenti in ogni caso comporta dei vantaggi prestazionali per alcune operazioni da fare in *ObP*. La costruzione di H prevede di trovare lo short prefix associato ad una dato prefisso (ottenuto concatenando un altro short prefix con un simbolo di Σ), quest'operazione è più efficiente tramite C_I dato che nel DT comporta il *sifting* del prefisso e quindi un numero di MQ nel caso medio logaritmico da moltiplicare per la dimensione media dei suffissi incontrati. E ancora nella decomposizione del controesempio vi è l'esigenza di trovare dato un prefisso, lo si chiami p,il corrispondente short prefix di p ma può accadere che p non sia mai stato visto o che facendone il *sift* nel DT non termini in una foglia già esistente ma invece provochi la creazione di un nuovo nodo foglia. Quindi non riusciamo a trovare lo short prefix associato a p senza l'ausilio di C_I, anche se in realtà è pur sempre possibile effettuare il " parsing " della stringa x in H e tornare lo short prefix associato allo stato cui si arriva in H. In realtà lo sforzo computazionale richiesto per mantenere un'ipotesi anche tramite l'insieme di componenti potrebbe essere maggiore dei vantaggi prestazionali ottenuti rispetto ad un'implementazione che usa esclusivamente il DT per rappresentare H nel caso di OneLocally. Si ribadisce invece che nel caso di OneGlobally (e anche AllGlobally) ,per i motivi suddetti, C_I è necessario per testare la chiusura e garantire un corretto funzionamento.

3.4.5 Teacher

Tutte le considerazioni effettuate nella sottosezione 2.5.3 inerenti il *teacher* di L* rimangono valide anche per l'*ObP* e ad esse si rimanda.

3.5 Scelte Progettuali

Si è implementato in C++11 l'algoritmo *ObP* utilizzando la versione OneGlobally (si è implementato anche OneLocally ma la versione di *ObP* inclusa nella libreria Gi-learning è OneGlobally) ed interfacciandosi con la libreria preesistente Gi-learning. L'esposizione e lo pseudocodice riportato nella sezione3.4 per l'*ObP* fanno riferimento a Howar [26] tranne qualche piccola modifica. Diversamente alcune delle scelte progettuali per il codice si discostano in parte da quanto esposto in Howar. Qui saranno illustrate le differenze principali e le ragioni da cui tali scelte sono scaturite. In prima analisi viene effettuata una preliminare minimizzazione del *DFA* target al fine di permettere un'eventuale[11] velocizzazione del *testing* dell'equivalenza tra H ed il *DFA* target, dato che le *performances* dell'algoritmo d'equivalenza *table-filling* dipendono in maniera quadratica dal numero di stati.
E' stata inserita l'ottimizzazione che permette di sfruttare più volte lo stesso controesempio finchè non lo è più. Come detto ciò consente di ridurre il numero di

[11] perchè il *DFA* può essere già minimo

EQ. L'ottimizzazione è presente anche nell'implementazionde dell'*ObP* nella **LearnLib**[12].
Non è stata implementata invece l'ottimizzazione *semantic suffix closdness* (vedasi [49]) che consente di ottenere delle ipotesi intermedie minime (che consentono di velocizzare le *EQ*) e di ridurre eventualmente il numero di *EQ*. L'ottimizzazione è invece presente nell'implementazione dell'*ObP* nella **LearnLib**.
Si è scelto di completare le componenti nel momento stesso che l'incompletezza è introdotta dato che i punti dove avviene sono ben circostanziati:

- dopo l'individuazione del suffisso discriminante, nell'algoritmo e alla riga 15.16, e la sua aggiunta all'insieme globale di suffissi che avviene in una nuova funzione chiamata UPDATE-FROM-COUNTEREXAMPLE. Inoltre in UPDATE-FROM-COUNTEREXAMPLE dato che — in corrispondenza dei prefissi discriminati dal suffisso tornato dal metodo di decomposizione del controesempio — si conosce già l'esito della *MQ* ,perchè già effettuate nel metodo di decomposizione del controesempio, si risparmiano 2 *MQ*. Detto n il numero di stati del *DFA* target minimo ciò consente di risparmiare fino a un massimo di $2 \cdot n$ *MQ*.

- in OBP-SIFT(), l'algoritmo 12. Sia se durante il *sifting* il prefisso termini in un componente esistente oppure in uno nuovo è necessarrio effettuare delle *MQ* per completare il componente esistente oppure il nuovo componente rispettivamente sull'insieme di suffissi globale. In questo modo durante il *sifting* sarà possibile evitare di eseguire nuovamente le *MQ* effettuate durante la navigazione del *DT* con quel dato prefisso per tutti i suffissi incontrati durante il *sifting* nel *DT*. Resta inteso che utilizzando la strategia OneGlobally non tutti i suffissi contenuti nell'insieme di suffissi globale saranno incontrati e per i restanti alla fine sarà necessario effettuare una *MQ* esplicita per assicurare la chiusura del componente.

OBT-SPLIT non introduce incompletezza.
La summenzionata gestione della completezza oltre a consentire di risparmiare delle *MQ* implica anche una migliore efficienza dell'algoritmo dato che alla riga 14.2 l'algoritmo originale OBT-CLOSEPACK deve controllare se C_I è completo e ciò comporta un controllo su tutte le componenti, controllo non più necessario.
Un'altra modifica significativa rispetto all'algoritmo OBT-CLOSEPACK originale (algoritmo 14) sta nell'individuazione del suffisso che determina la non chiusura, alla riga 4. Infatti il metodo di decomposizione del controesempio consente di determinare quali sono i due prefissi ed il suffisso che determina la non chiusura. Quindi ogni volta che viene chiamato OBT-CLOSEPACK non è necessario spendere del tempo di esecuzione nel cercare dove non si verifica la non chiusura. Questo è vero ad ogni prima chiamata di OBT-CLOSEPACK, può poi accadere che venga generata ulteriore non chiusura all'interno dell'algoritmo OBT-CLOSEPACK stesso che

[12] `www.learnlib.de` in cui si può trovare sotto la nomenclatura *Discrimination Tree* un'implementazione in Java della'algoritmo *ObP* di Howar

in questo caso va individuata. C'è da precisare che quanto detto non è valido per la prima chiamata in assoluto di OBT-CLOSEPACK dato che la prima chiamata in assoluto non è preceduta da una chiamata al metodo di gestione del controesempio (che come detto permette di individuare esattamente in quale componente e per quale coppia di prefissi e quale suffisso accade la non chiusura).

Inoltre esistono adesso due versioni del metodo di decomposizione del controesempio BINARY-SEARCH, di cui una è uguale a quella descritta nell' algoritmo 8,che è usata dagli altri metodi di decomposizione del controesempio, e l'altra invece consente di scovare la decomposizione del controesempio *on the fly* cioè durante l'esecuzione dell'algoritmo stesso (cioè all'interno dell'algoritmo di decomposizione del controesempio e quindi non verrà tornato un indice ma la decomposizione stessa). Nell'appendice,ed esattamente nella sezione B.2 sono inseriti i dettagli implementativi più significativi.

Capitolo 4

Observation Pack Approssimato

L'obiettivo di questo lavoro è indagare la capacità di apprendere linguaggi regolari in uno scenario di *Active Learning*, nel caso in cui non si disponga di un *Oracolo* capace di rispondere nativamente nè ad *EQ* nè a *MQ*[1]. In letteratura esistono diversi algoritmi utilizzati per l'appredimento di linguaggi regolari. L' approssimazione dell' *Oracolo* è indipendente dallo specifico algoritmo d'*IIR*, tuttavia per renderlo concreto l'attenzione è stata focalizzata sull'*ObP*. Si è scelto di utilizzare **Support Vector Machine (*SVM*)** come classificatore atto a modellare un *Oracolo*, classificatore costruito a partire da alcuni esempi positivi e negativi del linguaggio da apprendere. Sarebbe opportuno preventivamente leggere le Appendici C e D dato che i concetti e le tecniche lì descritte sono propedeudiche all'algoritmo qui delineato.
Data la natura non esatta dell'algoritmo ivi realizzato questo viene indicato come **Observation Pack Approssimato (*ObPA*)**.
Corredata a questa tesi vi è anche l'implementazione dell' *ObP* in C++11 , codice che è stato integrato in Gi-learning [15] una libreria preesistente. Nelle applicazioni reali tuttavia è altamente improbabile la disponibilità di un *Oracolo* in grado di rispondere a delle *EQ* da cui l'esigenza di un *Oracolo approssimato* e dell'*ObPA* la cui relativa implementazione oggetto di tesi è stata parimenti integrata in Gi-learning [15].

4.1 Precedenti lavori in letteratura

Assumere che il *teacher* sia in possesso del linguaggio target $\mathcal{L}$ è uno scenario poco plausibile essendo esso stesso l'oggetto dell'inferenza. Quindi l'obiettivo delineato in questa tesi è indagare il comportamento di un algoritmo di *active learning* per inferire il *DFA* A tale che $L(A) = \mathcal{L}$ tramite un oracolo approssimato realizzato con un classificatore statistico e tale intento si discosta dai lavori preesistenti in letteratura. In realtà una prima versione di un algoritmo di *active learning* che realizza un Oracolo approssimato è già presente in Angluin [2] in cui si approssima

[1] L'*Oracolo* è approssimato tramite un classificatore che è in grado per sua natura di rispondere a una *membership query*. Qui si intende che non è in grado di rispondere a una *MQ* tramite il *DFA* target perchè quest ultimo è considerato incognito nel processo inferenziale

un *EQ* tramite un certo numero di *MQ* nella sua versione di L* approssimato. Più di recente, nella stessa direzione è andata la competizione Zulu [13]: il vincitore della competizione [26] definisce il nuovo tipo di query: l' *Identity Query*

Definizione (Identity Query)**.** Testa se due prefissi u, u' sono nella stessa classe di equivalenza del target A cioè se $u \not\simeq_{\lambda^A} u'$. In caso $u \not\simeq_{\lambda^A} u'$ ritorna un suffisso v per il quale $\lambda^A(uv) \neq \lambda^A(u'v)$. Altrimenti ritorna il successo.

e dimostra che i linguaggi regolari possono essere inferiti con un numero polinomiale di *MQ* ed *Identity Query*. Le *Identity Query* non sono meno realistiche delle *EQ* ma tramite esse è possibile definire un framework che consente di approssimare in maniera relativamente facile ed efficiente un'*Identity Query* tramite *MQ*. Anche gli altri algoritmi in letteratura sono focalizzati sulla ricerca di algoritmi che in maniera efficiente riescono ad approssimare le *EQ* tramite *MQ*. In questa sede si assume invece l'impossibilità di rispondere nativamente anche alle *MQ* oltre che alle *EQ*, cioè anche l'esito delle *MQ* non è noto ed andrà approssimato.
Per quanto concerne la selezione del classificatore statistico per approssimare l'Oracolo la scelta è ricaduta su *SVM* perchè costituiscono un modello molto potente. Inoltre sono poche le applicazioni delle *SVM* per l'apprendimento di linguaggi regolari come in [11][12] in cui si riesce a definire un kernel string utile per l'apprendimento dei linguaggi planari[2] oppure in [30] dove si delinea un lavoro teorico sui linguaggi regolari e si propone un kernel string senza concretizzarlo specificamente per l'apprendimento di linguaggi regolari nell'accezione dell'*active learning*. Altri tipi di classificatori come ad esempio le *Recurrent Neural Network* si sono rilevate particolarmente adatte allo scopo di modellare un *DFA* ma come si evince da [20], che definisce una panoramica su di esse a riguardo dell'impiego in *GI*, sono state già ampiamente dibattute in letteratura.

4.2 Introduzione ObPA

Una preliminare osservazione per evitare la creazione di ambiguità nel prosieguo consiste nel rimarcare che il codice prodotto è costituito da una duplice versione. Vi è infatti il codice riguardante la fase di *debug* riferito d'ora in avanti come *ideal version* e quello pronto per l'utilizzo reale riferito d'ora in avanti come *release version* Con l'intento di spiegare le differenze principali tra le due versioni e contemporaneamente introdurre l'*ObPA* vengono forniti gli pseudocodici 16 e 17 di alto livello:
La differenza principale tra le due versioni è che nella *ideal version* si è in possesso del *DFA* target e nella *release version* non si possiede in alcun modo questa conoscenza. Si generano 1500 campioni bilanciati campioni con un particolare algoritmo ,**random walk**. Dai campioni iniziali si estraggono il *training set* di 500 elementi e il *test set* composto dai rimanenti 1000. Si crea un oracolo approssimato che

[2]I linguaggi planari sono una classe di linguaggi che attraversano la tassonomia di Chomsky nel senso che apprendono solo in maniera parziale i linguaggi finiti,regolari,context-free,context-sensitive ecc. senza saturare nessuno di essi

Algoritmo 16 OBPA *ideal version*

Input: il *DFA* target A,l' alfabeto Σ
Output: il *DFA* inferito Ob_DFA
1: $samples \leftarrow A.random\_walk(750, 750)$
2: $training\_set \leftarrow pull\_out(samples, 500)$
3: $test\_set \leftarrow pull\_out(samples, 1000)$
4: $appr\_oracle \leftarrow$ **new** $appr\_oracle(|\Sigma|, \Sigma, training\_set, test\_set)$
▷ Sul training_set si addestra SVM e con il *test_set* si fa model evaluation
5: $Ob\_DFA \leftarrow OBSERVATION\_PACK(appr\_oracle).run()$
6: $statistical\_measure \leftarrow compare(Ob\_DFA, A)$
7: **return** Ob_DFA

Algoritmo 17 OBPA *release version*

Input: l' alfabeto Σ
Output: il *DFA* inferito Ob_DFA
1: $training\_set \leftarrow read\_file(path\_file)$ ▷ Nel file samples disponibili all'utente
2: $appr\_oracle \leftarrow$ **new** $appr\_oracle(|\Sigma|, \Sigma, training\_set)$
▷ Sul training_set si addestra SVM e non si effettua model evaluation
3: $Ob\_DFA \leftarrow OBSERVATION\_PACK(appr\_oracle).run()$
4: **return** Ob_DFA

provvede internamente a costruire un classificatore a partire dal *training set* e usa il *test set* per valutare il classificatore scelto. Si invoca l'*ObP* passandogli l'Oracolo approssimato; *ObP* funziona in maniera classica come descritto nel capitolo 3 ma al momento di invocare i metodi per effettuare le *MQ* ed *EQ* saranno chiamati i metodi dell'Oracolo approssimato che saranno realizzati mediante il classificatore precedentemente costruito. Viene inferito un *DFA* che viene confrontato con il *DFA* target tramite delle misure statistiche (cfr. App. D.5) . L'algoritmo *random walk* nonchè la scelta della dimensione del *training set* e del *test set* e come avviene il confronto tra il target e il *DFA* inferito sarà spiegato più approfonditamente nella sezione 5.2.
Nella *release version*, non essendo in possesso di un *DFA* target (come accade del resto in uno scenario reale) i campioni in possesso dell'utente vengono inseriti su un file e una volta caricati nel programma costituiscono essi stessi il *training set*. Come per la *ideal version* si costruisce un Oracolo approssimato ma non si effettua model evaluation (non c'è un *test set*) per ragioni che verranno chiarite in seguito. Alla stregua di quanto descritto per la *ideal version* si inferisce un *DFA* mediante l'algoritmo *ObP* ma non è possibile effettuare una comparazione di quest ultimo con il *DFA* target dato che è ignoto.

4.3 Costruzione del classificatore

Come capita spesso nell'addestramento di un classificatore sono molti gli *iperparametri* da potere regolare e le possibili strade perseguibili per ricercare una soluzione soddisfacente. Per questo motivo sono state realizzate tre diverse implementazioni nominate in seguito come *ObPA*1 *ObPA*2 e *ObPA*3. Per ognuna di esse vi è sia una *ideal version* che una *release version* e lo pseudocodice illustrato negli algoritmi 16 e 17 rimane valido, ma tra un'implementazione e l'altra cambiano le tecniche utilizzate per costruire il classificatore per l'oracolo approssimato a partire dal *training set*.

4.3.1 Libreria esterna per SVM

Come già evidenziato in precedenza il modello prescelto è *SVM* e la libreria esterna scelta è l'implementazione di Thorsten Joachims nella versione SVM^{light} (cfr. App. C.3). Qui si aggiungono ulteriori peculiarità riguardanti SVM^{light} in contrapposizione alla libreria GI-learning

- è in linguaggio C mentre la libreria GI-learning è in C++
- è concepita come un eseguibile pronto all'uso
- è una libreria povera in termini di tecniche di *data processing* e model selection

Quindi è stato necessario un notevole sforzo implementativo per aggiungere alcune funzionalità di cui si parlerà a breve (k-fold-validation, codifica, scaling ecc.) e per integrare la libreria *SVM* all'interno di GI-learning. Inoltre non è stato possibile utilizzare SVM^{light} come una *black-box* ma ispezionarla a basso livello per modificare alcuni aspetti e funzionalità per riadattarle agli scopi di *ObPA*.

4.3.2 ObPA1

L'algoritmo *ObPA*1 (relativamente all'addestramento del classificatore) viene presentato nell'algoritmo 18 corrispondente a una possibile realizzazione pratica di quello che negli algoritmi 16 e 17 è stato indicato con *appr_oracle*.
Nell'algoritmo 18 è stata presentata la *ideal version* di *ObPA*1 , la *release version* differisce solo per la riga 6 e per l'assenza del *test set* quindi per essa non si fa *model evaluation* come si approfondirà a breve.
I campioni devono essere necessariamente pre-processati perchè essendo categorici (l'alfabeto in generale è fatto da stringhe di caratteri alfanumerici) non sono direttamente utilizzabili da *SVM* e allora li si rende numerici con **integer encoding** (cfr. App. D.3.1).
SVM presuppone che i campioni siano della stessa lunghezza ed ecco giustificato il *padding*. Si seleziona la stringa più lunga nel *training set* e tutte le altre stringhe sono uniformate a questa lunghezza tramite il *padding*. Siccome *SVM* si basa su

Algoritmo 18 OBPA1

Input: a $TRS = training\_set$, a $TES = test\_set$, a Σ
Output: *app_oracle*
1: $encode(TRS, TES)$
2: $padding(TRS, TES)$
3: $scale(TRS, TES)$
4: **best_model** $\leftarrow grid\_search(TRS)$ ▷ Si fa model selection
5: $new\_best\_model \leftarrow$ retrain **best_model** found, on whole TRS
6: $statistical\_results \leftarrow model\_evaluation(new\_best\_model, TES)$
7: **return** app_oracle(new_best_model)

misure di similarità tra i campioni (cfr. App. C.2.4) non è stato ritenuto opportuno scegliere un valore fisso per il valore di padding che è allora diventato un parametro. Dato che i singoli attributi variano nello stesso range e come si dirà si è usato un *kernel* gaussiano che non ha problemi numeri lo *scaling* dei dati non è strettamente necessario ed allora se effettuarlo o meno è diventato un altro parametro. La tecnica usata per lo *scaling* è quella della **standardizzazione** (cfr. App. D.3.2) .
Terminata questa prima fase di elaborazione dei campioni grezzi il successivo step consiste nella scelta degli *iperparametri migliori* con la model selection. Inizialmente è stato necessario delineare i parametri. Si è scelto di usare *soft-margin* combinato con l'impiego di un *kernel* perchè è notoriamente la tecnica che da migliori risultati per *SVM*. In *ObPA*1 si è deciso di usare uno dei *kernel* noti in letteratura e la scelta è ricaduta sul *kernel* gaussiano (per i motivi ampiamente illustrati nell'App. D.4). L'intervallo di variazione del parametro γ del *kernel* gaussiano nonchè del parametro C(trade-off parameter) è quello indicato nella sezione D.4 in cui è anche spiegato che una ricerca come *grid search* per la selezione dei parametri migliori malgrado sia molto onerosa computazionalmente sia la più affidabile e per questo si è optato per essa per realizzare model selection. Per il parametro C è anche usato il valore di default della libreria *SVM*. Il padding è settato con 8 valori positivi diversi più due valori negativi e nel caso che il valore di padding positivo corrente non produca un classificatore migliore rispetto a un valore di padding precedente si passa direttamente all'ultimo valore di padding saltando quelli intermedi. In definitiva vi sono quattro parametri: lo scaling,il padding,C,γ che rendono molto grande lo spazio dei parametri.
Poi si è scelto di utilizzare la *5-fold-validation* per avere stime il più possibile unbiased. SVM^{light} mette a disposizione anche LOO ma ne calcola solo un'approssimazione e quindi non del tutto attendibile nonostante ciò *ObPA* è implementato in modo tale da potere scegliere dall'esterno la preferenza sul metodo di validazione incrociata (LOO o 5-fold-validation). Un altro criterio di stima dell'errore messo a disposizione da SVM^{light} è Xi-Alpha che viene calcolato in maniera molto efficiente ma come si evince da [18] 5-fold-validation è più attendibile.
Il modo classico di procedere per la model selection sarebbe quello di dividere i campioni disponibili (cioè quelli menzionati come TRS alla riga 4) in due parti un

training set MS e un *test set* ME . Su MS si fa *5-fold-validation* (per ogni combinazione di valori dei parametri dato che si usa *grid search*) e si usa la media dei cinque valori di *accuracy* ottenuti per fare model selection. Per il classificatore migliore trovato (e i suoi parametri) si rieffettua l'addestramento usando tutto l'insieme MS [3].
Per il classificatore costruito si fa model evaluation utilizzando gli *unseen* campioni del *test set* ME e infine si rieffettua l'addestramento su tutti i campioni iniziali (cioè i 500 campioni di TRS).
In *ObPA*1 (e anche nelle altre versioni) in *release version* si è fatta una scelta diversa. Con l'intento di ottimizzare le prestazioni e di usare il massimo numero di campioni per la scelta del modello si fa solo model selection tramite *5-fold-validation* su tutti i campioni (i 500 campioni di TRS) riaddestrando dopo la scelta dei migliori *iperparametri* su tutto l'insieme di campioni TRS e rinunciando a fare model evaluation. Quindi non verrà fornito un *feedback* sulle capacità di generalizzare del costruito classificatore statistico all'utente dato che quest ultimo potrà visionare solo l'accuracy della *5-fold-validation* che è indicativa ma non una stima precisa dell'errore di generalizzazione.
In *ideal version* si effettua la stessa cosa che in *release version*, ma essendo in possesso del *DFA* target è possibile produrre dei campioni aggiuntivi e ne vengono generati 1000 (senza sovrapposizione con quelli del *training set*) che fungeranno da *test set* per fare model evaluation sul migliore classificatore trovato (il classificatore viene scelto alla stregua della *release version*).
Due aspetti importanti relativi a quanto appena esposto sono:

1. I campioni del *test set* vanno elaborati con gli stessi valori trovati nella fase di preprocessing per il training set. Tali valori non vanno quindi ricalcolati *ex novo* sul *test set*.

2. Dal *test set* vanno scartate le eventuali stringhe di lunghezza maggiore della stringa di lunghezza massima tra quelle incontrate nel *training set*. Quindi il classificatore costruito ha il limite di funzionare solo su stringhe fino ad una certa lunghezza. Quindi se *ObP* dovesse formulare una *MQ* più lunga di quella di lunghezza massima incontrata nel *training set* il classificatore non è in grado di rispondere.

4.3.3 ObPA2

In *ObPA*2 si usa come codifica dei dati **OHE** (cfr. App D.3.1) la cui rappresentazione risultante è quella in cui ogni stringa rappresenta un vertice dell'ipercubo di lato unitario. Questa è una rappresentazione molto sparsa e ridondante, per cui incline all'*overfitting*, ma nella fattispecie si è ipotizzato che i molti gradi di libertà potessero tornare utili per la complessità di rappresentazione dei linguaggi regolari.
Rispetto a quanto esposto su OHE nell'Appendice D.3.1 occorre un bit in più per tenere in considerazione il valore per il padding. Ad esempio se l'alfabeto $|\Sigma| = 3$ in

[3] con 5-fold-validation il classificatore viene costruito su 4 "folds" e non su tutto l'insieme

OHE si ha:

1000 0100 0010 0001

laddove l'ultimo elemento è la codifica OHE per il valore di padding mentre i primi tre elementi servono a codificare le stringhe che costituiscono l'alfabeto Σ.
Tutte le considerazioni effettuate per *ObPA*1 restano valide per *ObPA*2. Le uniche differenze a parte quella suddetta sono che il valore di padding è unico e che per la codifica OHE non è necessario effettuare lo scaling quindi vengono a decadere due parametri velocizzando l'algoritmo. Il rovescio della medaglia è che il numero di attributi per stringa aumenta notevolmente e per stringhe o alfabeti molto lunghi questo metodo può allungare notevolmente l'onere computazionale per la convergenza dell'algoritmo della librearia SVM^{light}

4.3.4 ObPA3

Per superare il limite sulla lunghezza delle stringhe in *ObPA*1 e *ObPA*2 e provare ad ottenere dei risultati migliori è stata fornita una terza versione basata su un *kernel string* (cfr. App. C.2.4) . Alcuni *kernel* noti in letteratura sono stati oggetto di studio in precedenti lavori riuscendo ad apprendere alcuni specifici linguaggi regolari o classi di linguaggi che non comprendono l'intera classe dei linguaggi regolari [11][12]. Kontorovich e Nadler in [30] riescono ad ottenere dei risultati generali che adesso saranno presi in esame:
Detti C ,la classe dei concetti, ed X ,lo spazio delle istanze, due insiemi **numerabili**[4] si ha:

Definizione (Finitamente linearmente separabili)**.** Un concetto $c \in C$ è *finitamente linearmente separabile* se esiste una funzione $\phi : X \to \mathbb{B}^{\mathbb{N}}$ e un vettore dei pesi $w \in \mathbb{R}^{\mathbb{N}}$, entrambi con *supporto finito* , per $\forall x \in X$, tale che:

$$c = \{x \in X : (w \star \phi(x)) > 0\}$$

La classe dei concetti C è *finitamente linearmente seprabile* se ogni $c \in C$ è *finitamente lineramente separabile* sotto lo stessa funzione ϕ (w invece può variare da un concetto all'altro). Il concetto di supporto finito è molto importante perchè assicura che vi sia per ogni concetto c una dimensione finita $D(c)$ tale che c sia linearmente separabile in $\mathbb{B}^{D(c)}$, cioè in ultima analisi la funzione $\phi()$ mappa le stringhe del concetto in uno spazio di dimensione finita $D(c)$ in cui le stringhe sono lineramente separabili cioè esiste un iperpiano che separa le stringhe positive del concetto da quelle negative.

Teorema 4.1. *Ogni classe dei concetti numerabile C su uno spazio delle istanze X numerabile è finitamente linearmente separabile.*

[4]Un insieme è numerabile se ha cardinalità finita o cardinalità uguale a quella di $\mathbb{N}$ (è possibile mettere i suoi elementi in corrispondenza biunivoca con quelli di $\mathbb{N}$)

Questo teorema ,per la cui dimostrazione si rimanda al riferimento succitato, è strettamente correlato alla Structural Risk Minimization (cfr. App. C.1.3) dato che per la dimostrazione del teorema 4.1 vengono utilizzate due funzioni generiche $|\cdot|$ e $\|\cdot\|$ che indicano rispettivamente una misura della complessità di un'istanza e di un concetto (ad esempio nel caso di una stringa si può scegliere la sua lunghezza e per un *DFA* il numero di stati dell'automa canonico). Il trucco nella costruzione della funzione $\phi()$,che permette di separare le istanze di un generico concetto, è di arrivare alla lineare separabilità regolando le due funzioni suddette che difatti sono una misurà della capacità ,cioè della complessità, del concetto e delle stringhe.
Un primo risultato rilevante è il seguente teorema formulato nell'ambito del *PAC-learning* [52]:

Teorema 4.2. *Detta C una classe di concetti sulle istanze X, entrambe numerabili. Si definisce la famiglia di kernel:*

$$K_n(x,y) = \sum_{c:\|c\|\leq n} \lambda^c(x)\lambda^c(y) \qquad [5] \tag{4.1}$$

per $n \in \mathbb{N}$. Allora è possibile apprendere efficientemente C dagli esempi etichettati. Detto $c \in C$ uno specifico concetto, e detto $S \subset X$ un training set cioè una sequenza di m istanze casuali etichettate secondo c.
*Esiste un efficiente algoritmo **UniversalLearn**, che prende in ingresso S e restituisce un classificatore $\hat{f} : X \to \{-1, 1\}$ che con probabilità almeno $1-\delta$ ha un piccolo errore di generalizzazione limitato superiormente da:*

$$\frac{2}{m}(R(c)\log(8em)\log(32m) + \log(\frac{8m}{\delta})), \qquad per \quad m \geq \theta^{-1}(\|c\|)$$

Si chiarifica per evitare ambiguità che le istanze x, y di K_n appartengono al *training set* S.
In pratica le funzioni che definiscono il *kernel* K_i sono le funzioni che costituiscono le componenti della funzione $\phi()$ (nell'App. C.2.4 si spiega in che modo una funzione che trasforma i campioni nello spazio di Hilbert può essere composta da più funzioni) che realizzano la trasformazione dei campioni in uno spazio delle *features* (spazio di Hilbert). Queste funzioni sono tutti i concetti c fino alla dimensione $\|c\| = i$. L'algoritmo UniversalLearn consiste , per un dato i,nel trasformare e mappare nello spazio delle *features* i singoli campioni del *training set* S secondo la funzione $\phi()$ (che consiste nel calcolare ogni singola *feature* cioè attributo di un campione x del *training set* S secondo l'esito di $\lambda^c(x)$ con i concetti c che sono tutti quelli tali che $\|c\| \leq i$).
Quindi per un dato i si mappano nello spazio delle *features* tutti i campioni di S e per essi si calcola il **margine**(cfr. App. C.2.2) e si seleziona l'iperpiano (il vettore dei pesi) che rende il margine massimo.
Il calcolo del margine può avvenire in maniera molto naturale con *SVM*, infatti il teorema 4.2 definisce il *kernel* K_i come un prodotto scalare ed *SVM* ,una volta

[5] La funzione λ è stata definita per i *DFA* e qui la si estende a un concetto generico

trasformati i campioni secondo la funzione $\phi()$, calcola automaticamente il vettore dei pesi che massimizza il margine e nella forma duale figura il prodotto scalare dei campioni e nel caso di *SVM* con *kernel* figura il prodotto scalare dei campioni trasformati nel *feature space* tramite $\phi()$ (oppure viene valutato direttamente il *kernel* sui campioni a coppie con il *kernel trick* ma non è questo il caso) che è esattamente quello che definisce il *kernel* del teorema 4.2.
UniversalLearn ripete il calcolo descritto sopra per $\forall i : 1 < i < \theta(m)$ e alla fine seleziona il mapping $\phi()$ corrispondente al valore di i che permette di calcolare il margine massimo sui campioni.
La condizione $m \geq \theta^{-1}(\|c\|)$ è necessaria affinchè esista almeno un valore di i per cui sia garantita la lineare separabilità del *training set* S dopo la trasformazione $\phi()$.
É richiesto che θ sia una funzione monotonicamente crescente. Inoltre se θ cresce troppo velocemente $\theta(m)$ sarà troppo grande e l'algoritmo UniversalLearn diventa inefficiente, al contrario con una funzione θ che cresce lentamente $\theta(m)$ sarà troppo piccolo e la condizione del teorema 4.2 $m \geq \theta^{-1}(\|c\|)$ cioè $\theta(m) \geq (\|c\|)$ potrebbe non essere rispettata[6]. Quindi è necessario scegliere attentamente la funzione $\theta()$ e $ceil(\sqrt{\cdot})$ viene suggerita da Kontorovich come un equo compromesso.
UniversalLearn è intimamente collegato al Boosting (cfr. App. D.2.4) dato che il *kernel* K_n per un concetto c viene calcolato a partire da concetti di complessità (per i *DFA* il numero di stati) minore di c.

L'algoritmo UniversalLearn rende la classe dei concetti C linearmente separabile ma un suo utilizzo pratico è improponibile perchè l'esatta valutazione di questo kernel coinvolge la sommatoria in (4.1) che può contenere un numero super-esponenziale di termini (cioè i concetti tali che $\|c\| \leq n$). Viene in soccorso il seguente teorema:

Teorema 4.3. *Detta C una classe di concetti sulle istanze X, entrambe numerabili. Detta $S \subset X$ una sequenza di m istanze(training set casuale con istanze etichettate in accordo a qualche ignoto $c \in C$. Per qualsiasi $\delta > 0$, esiste un algoritmo randomizzato* ***ApproxUnivLearn*** *che da in uscita un classificatore $\hat{f} : X \to \{-1, 1\}$ tale che con probabilità almeno $1 - \delta$ si ha un errore di generalizzazione piccolo limitato superiormente da:*

$$\frac{2}{m}(4R(c)\log(8em)\log(32m)+\log(\frac{16m}{\delta})), per m \geq max\left\{\theta^{-1}(\|c\|), \frac{D(c)}{2}, \frac{R(c)^2}{8.4 \times 10^4}\right\}$$

dove $D(c) > 0$ e $R(c) < \infty$ sono costanti che dipendono solo da c.

e si dimostra che ha anche un tempo di esecuzione polinomiale.
ApproxUnivLearn come UniversalLearn esegue il ciclo principale $\theta(m)$ volte (al variare dell'indice n). Gli algoritmi sono identici , la differenza è che per approssimare

[6]Si usa il condizionale perchè il target cioè il concetto c è ignoto e quindi non si conosce la sua dimensione $\|c\|$ e per questo non è possibile impostare esattamente $\theta(\cdot)$

K_n vengono estratti casualmente in maniera uniforme T concetti $c : 1 \leq \|c\| \leq n$ con *sampling without replacing* [7]. T equivale a :

$$T = \frac{1}{2\epsilon^2} \log\Big(\frac{m^2 + m}{\delta/2}\Big)$$

con ϵ che deve essere piccolo dato che rappresenta anche la differenza tra il il *kernel* esatto e quello approssimato. Nel riferimento si consiglia di scegliere $\epsilon = 1/\sqrt{m}$. Per ogni n si calcola il margine: con i T concetti precedentemente estratti si mappano i campioni nello spazio delle *features* e per essi si calcola il margine maggiore. I campioni che hanno il margine maggiore determinano quale trasformazione dei campioni scegliere per mappare in un secondo momento anche i campioni del *test set* .
ApproxUnivLearn è un algoritmo che si applica a una classe generica di concetti e istanze numerabili in cui rientrano anche i linguaggi regolari. Un altro risultato interessante anche se esula dallo scopo di questo lavoro è che il *kernel* appena definito conduce a una caratterizzazione alternativa della classe dei linguaggi regolari.
Per *ObPA*3 rimane valido quanto detto negli algoritmi 16 e 17 ma esso costruisce il classificatore implementando l'algoritmo ApproxUnivLearn per i linguaggi regolari. Acciocchè risulti chiaro il lavoro svolto si rimanda allo pseudocodice 19 di alto livello e autoesplicativo per *ObPA*3 inerente la costruzione del classificatore nella *ideal version*. Nel codice reale vi è anche la possibilità di aumentare il valore di T passando un fattore moltiplicativo , che aumenta il numero di *feature* nel caso si abbia molta potenza elaborativa. Una precisazione concernente l'algoritmo 19 riguarda cosa si intenda con la notazione $DFA(1 : i)$. Indicato con $DFA(n)$ tutti i *DFA* con esattamente n stati allora si ha che $DFA(1 : i) = \bigcup_{k=1}^{i} DFA(k)$. Inoltre per giustificare la necessità di ApproxUnivLearn anzichè UniversalKernel per la classe dei concetti dei linguaggi regolari rappresentati dal formalismo dei *DFA* si mette in evidenza che

$$|DFA(k)| = 2^k k^{|\Sigma|k}$$

che cresce molto velocemente. Un altro discorso degno di nota è la realizzazione del *sampling without replacing* per l' estrazione uniforme dei T *feature DFA* da DFA(1:n). Se si avessero tutti gli elementi dell'insieme da cui estrarre cioè i *DFA* la realizzazione sarebbe immediata ma non è questo il caso e il procedimento va simulato. Si crea la distribuzione di probabilità π con un indice k che varia in $\{1, \ldots, n\}$:

$$\pi_k = \frac{|DFA(k)|}{DFA(1 : n)}$$

Per estrarre casualmente un *DFA* è necessario prima stabilire il suo numero di stati e acciocchè si estrae un numero tra 1 ed n in accordo alla distribuzione di probabilità π. E poi si procede ad estrarre casualmente un *DFA* di quella dimensione (si estrae in maniera uniforme un elemento dell'alfabeto per ogni cella della tabella di transizione). Ma è necessario assicurarsi che il *DFA* estratto non sia stato già estratto

[7] Significa che se si estrae un elemento x da un insieme X alla successiva estrazione l'elemento x viene eliminato dall'insieme

in precedenza perchè il *sampling without replacing* non prevede elementi duplicati. Per una realizzazione efficiente si è utilizzata una funzione hash da applicare agli elementi della tabella di transizione di un *DFA* (interpretati come una stringa) come impronta dello stesso. Il digest viene salvato (è anche usato per ordinare i vari *DFA* in modo da cercare in maniera efficiente usando come chiave il digest stesso) e quando viene estratto un nuovo *DFA* se ne calcola il digest e si verifica che non ci sia già un elemento memorizzato con lo stesso digest. Acciocchè è stata usata la funzione hash MurmurHash3 a 128 bit [5]. Con questa funzione la probabilità di una collisione è pressoché nulla.
Inoltre per un indice i piccolo $DFA(i)$ contiene pochi automi e può accadere che $|DFA(i)| < T$, in questo caso si effettua un controllo preventivo e tutti i $|DFA(i)|$ automi vengono creati deterministicamente (il *DFA* viene creato a partire da una permutazione che rappresenta la tabella di transizione) . In questo caso può accadere che le features in questione essendo in numero esiguo non riescano a rendere il *training set* linearmente separabile e alcuni campioni saranno misclassificati e in tali circostanze la fase di addestramento di *SVM* può diventare più lenta. Per provare a porre rimedio quando viene riconosciuto uno scenario del genere viene impostato il parametro $-e$ che costituisce il criterio di terminazione per l'algoritmo *SVM* a un valore più alto.
Un altro discorso rilevante è relativo a un limite della libreria SVM^{light} la quale non mette a disposizione la funzionalità di *SVM* nella versione *hard margin* che ha senso solo quando i campioni sono linearmente separabili ed è questo il caso. Quindi è stato necessario utilizzare *soft-margin* impostando il parametro C di trade-off al valore 1. Infatti siccome si è sicuri di avere un training set linearmente separabile, si cerca di massimizzare il margine e non di ridurre gli errori di classificazione (quest'ultima opzione comporterebbe di penalizzarli con un C grande in modo da fare meno misclassificazioni) perchè gli errori di classificazione sono zero. In quest'ottica un valore di C piccolo come può essere C=1 permette di massimizzare il margine approssimando hard margin.
Rispetto ad *ObPA*1 ed *ObPA*2 con *ObPA*3 non si effettua più la $grid - search$ per la ricerca degli *iperparametri* migliori mentre la model selection viene automaticamente effettuata con la ricerca delle *features DFA* che massimizzano il margine. Per realizzare la model evaluation è necessario prima convertire i dati nello spazio di Hilbert tramite i *feature DFA* per poi confrontare la loro classificazione rispetto al classificatore a margine massimo trovato nella precedente fase di model selection.

4.3.5 Approssimazione di EQ e MQ

Terminata la fase di addestramento e costruzione del classificatore la successiva fase è del tutto analoga a quanto visto per i metodi classici di *active learning* e nella fattispecie *ObP*. La sostanziale differenza in *ObPA* è che le *MQ* ed *EQ* sono realizzate mediante l'oracolo approssimato e in ultimo mediante il classificatore precedentemente realizzato.

Algoritmo 19 OBPA3

Input: a $TRS = training\_set$, a $TES = test\_set$, δ
Output: *app_oracle*

$m = |TRS|$ ▷ m è la dimensione del training set
$T \leftarrow \frac{m}{2}\log(\frac{m^2+m}{\delta/2})$ ▷ Formula a cui si riduce T per $\epsilon = 1/\sqrt{m}$
$T\_DFA\_Final = \{\}$ ▷ Inizializzazione vuota. Conterrà i feature DFA finali
$best\_margin = -\infty$
$best\_model = \{\}$
for $i \leftarrow 1, ceil(\sqrt{m})$ **do**
    $T\_DFA \leftarrow uniform\_sampling(T, DFA(1:i))$ ▷ Estrai T features DFA
    $feature\_TRS = \{\}$ ▷ Conterrà i campioni trasformati nel feature space

    for all $x \in TRS$ **do**
        $feature\_x = \epsilon$ ▷ Inizializza con la stringa vuota
        for all $featuredfa\ a \in T\_DFA$ **do**
            $feature\_x \leftarrow feature\_x \cdot \lambda^a(x)$
        end for all
        put $feature\_x$ *in* $feature\_TRS$
    end for all

    $temp\_margin \leftarrow compute\_margin(feature\_TRS)$ ▷ Calcolalo con *SVM*
    store current_model ▷ Salva il modello *SVM* corrente.
    if temp_margin > best_margin **then**
        $best\_margin \leftarrow temp\_margin$
        $best\_model \leftarrow current\_model$
        $T\_DFA\_Final \leftarrow T\_DFA$
    end if
    $i \leftarrow i + 1$
end for
▷ A seguire si mappa TES nel feature space e si fa model evaluation
$feature\_TES = \{\}$
for all $x \in TES$ **do**
    $feature\_x = \epsilon$ ▷ Inizializza con la stringa vuota
    for all $featuredfa\ a \in T\_DFA\_Final$ **do**
        $feature\_x \leftarrow feature\_x \cdot \lambda^a(x)$
    end for all
    put $feature\_x$ *in* $feature\_TES$
end for all
$statistical\_results \leftarrow model\_evaluation(best\_model, feature\_TES)$
return app_oracle(best_model,T_DFA_Final)

MQ approssimata

L'approssimazione di una *MQ* per una stringa s è quasi immediata. Se $|s|$ supera la lunghezza della stringa più lunga incontrata in fase di addestramento si lancia un'eccezione ma in pratica l'algoritmo termina perchè è una situazione ingestibile. Quanto detto è vero solo per *ObPA*1 e *ObPA*2 mentre *ObPA*3 riesce a gestire stringhe di lunghezza diversa. Si effettua il preprocessig di $|s|$ a seconda della versione di *ObPA* utilizzata e si etichetta la stringa come positiva e si effettua la predizione.[8] Se si ottiene una previsione corretta (accuracy tornata del 100%) la stringa è positiva secondo il classificatore altrimenti (accuracy nulla) la stringa è negativa. Per questioni di efficienza per evitare la scrittura su file in *ObPA*3 interagendo a basso livello con la libreria *SVM* si è riusciti a ricavare il vettore dei pesi w in modo da realizzare la *MQ* come $sign(w \bullet s + b)$ (cfr. App. C.2.2).

EQ approssimata

L'idea è realizzare un *EQ* (tra il *DFA* H e il classificatore che rappresenta il target) tramite una serie di *MQ* su un insieme di stringhe. Il modo in cui è creato l'insieme di stringhe è spiegato dettagliatamente in 5.2.4 cui si rimanda. L'*EQ* è superata quando la percentuale di campioni dell'insieme classificata alla stessa maniera sia dall'ipotesi H che dal classificatore precedentemenete addestrato supera una certa soglia. La soglia è impostata come un parametro esterno. Avendo realizzato in *ObPA*3 l'ottimizzazione che bypassa la lettura da file per effettuare una *MQ* approssimata, l'*EQ* è effettuata come una successione di *MQ* approssimate.

4.4 Scelte Progettuali

Un lavoro d'implementazione degno di nota è costituito dalla preliminare ingegnerizzazione del codice. Come detto il prescelto algoritmo di *active learning* è stato *ObP*, che nella sua versione base presentata nel capitolo 3 necessita di un *DFA* target che funge da Oracolo onnisciente. Nell'ottica di preservare questa versione dell'*ObP* e contemporaneamente integrare nella libreria le funzionalità dell'*ObPA* si è deciso di implementare una versione polimorfica. Più in dettaglio si è creata una gerarchia di classi per l'entità astratta Oracolo come in figura 4.1.
L'algoritmo *ObP* invece di possedere (composizione) un *DFA* target ha un oggetto della classe base astratta Oracolo. Quando si invoca il costruttore dell' *ObP* si deve passare anche l'indirizzo di un oggetto della classe Oracolo. Tramite il tipo di oggetto Oracolo(si ha un handle della classe base Oracolo che punta allo specifico oggetto della classe derivata passato) passato viene determinato in maniera trasparente con il polimorfismo se si desidera utilizzare l'*ObP* piuttosto che l'*ObPA*. Le chiamate alle funzioni effettuate sull'oggetto Oracolo all'interno di *ObP* hanno

[8] la predizione avviene scrivendo la stringa elaborata su file nel formato adeguato per SVM^{light} e si invoca la funzione che effettua la predizione passandogli il percorso del file dove c'è la stringa post-elaborata e il file dove è memorizzato il migliore modello precedentemente trovato.

Figura 4.1: Ereditarietà Oracolo

la stessa *signature* , cioè le *MQ* e le *EQ* , ma il tipo dell'oggetto Oracolo passato determina automaticamente di quale classe derivata invocare le funzioni (che avranno diverse implementazioni a secondo se appartengono alla classe exactOracle o a quella approximateOracle). In questo modo la classe dell'algoritmo *ObP* ha subito pochissime modifiche e il lavoro d'implementazione di questa tesi si è focalizzato sulla classe approximateOracle. Infine all'interno della classe observation_pack , per impedire al *client* di vedere le strutture dati e le modifiche effettuate su di esse e in ultimo il modo in cui funziona l'algoritmo, è necessario effettuare una copia interna dell'Oracolo passato dal *client* ma non conoscendo a priori di quale tipo, e quindi di quale classe, è l'Oracolo vi è stata la difficoltà su come invocare il costruttore di copia. A tal fine si è usata la tecnica del *clone idiom* .Quando si ha un riferimento polimorfico cioè un puntatore alla classe base che punta a un oggetto della classe derivata all'occorrenza può sorgere il problema di determinare qual è il tipo della classe derivata . Allora nella classe base si dichiara un metodo virtuale puro che tutte le classi derivate devono quindi implementare. Quest'implementazione consiste nella creazione dinamica di un nuovo oggetto della classe derivata chiamando il costruttore di copia della classe derivata sull'oggetto corrente (tramite new classeDerivata(*this)). Questo nuovo oggetto creato verrà tornato al chiamante. Si noti che il costruttore di copia della classe derivata deve provvedere a chiamare il costruttore di copia della classe base e che in questi costruttori deve avvenire una deep copy di tutti i membri dinamici. In figura 4.2 l'interfaccia della classe base astratta.

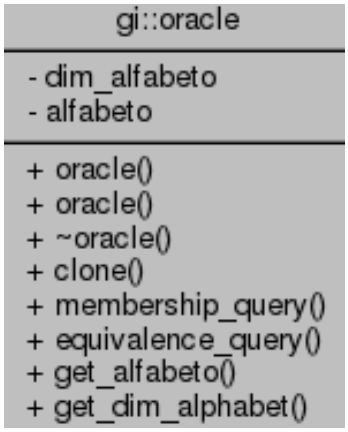


Figura 4.2: Interfaccia classe base Oracolo

Capitolo 5

Risultati sperimentali

La valutazione delle *performances* di un algoritmo di *GI* possono essere influenzate sia positivamente che negativamente da alcuni fattori come evidenziato in [56]. Alcuni di questi fattori riguardano la complessità del *DFA* target:

- numero di stati
- dimensione dell'alfabeto
- la lunghezza del più lungo dei cammini minimi dallo stato iniziale a qualunque altro stato
- il numero di transizioni totali[1]

altri il "sample set":

- numero di campioni del *training set* e del *test set*
- campioni suddetti strutturalmente completi

Nell'effettuazione degli esperimenti è necessario tenerne conto onde evitare dei risultati pessimisticamente o ottimisticamente *biased*. Per molti di questi parametri ci si è attenuti all'impostazione della competizione STAMINA [56].

5.1 Dataset

In letteratura si trovano molti algoritmi innovativi di *GI* i quali ,non conoscendone ancora le caratteristiche, in fase di sperimentazione sono stati preventivamente testati su dei linguaggi regolari "non banali" ma relativamente semplici. In un secondo momento i limiti dell'algoritmo sono testati contro linguaggi più complessi che ricalcano da vicino la complessità degli automi testati nelle varie competizioni di *GI* o su dataset reali. Per le finalità di questo lavoro si è scelto di perseguire questa strada.

[1] (escluse quelle che vanno nello stato pozzo per i *DFA*)

5.1.1 Tomita Dataset

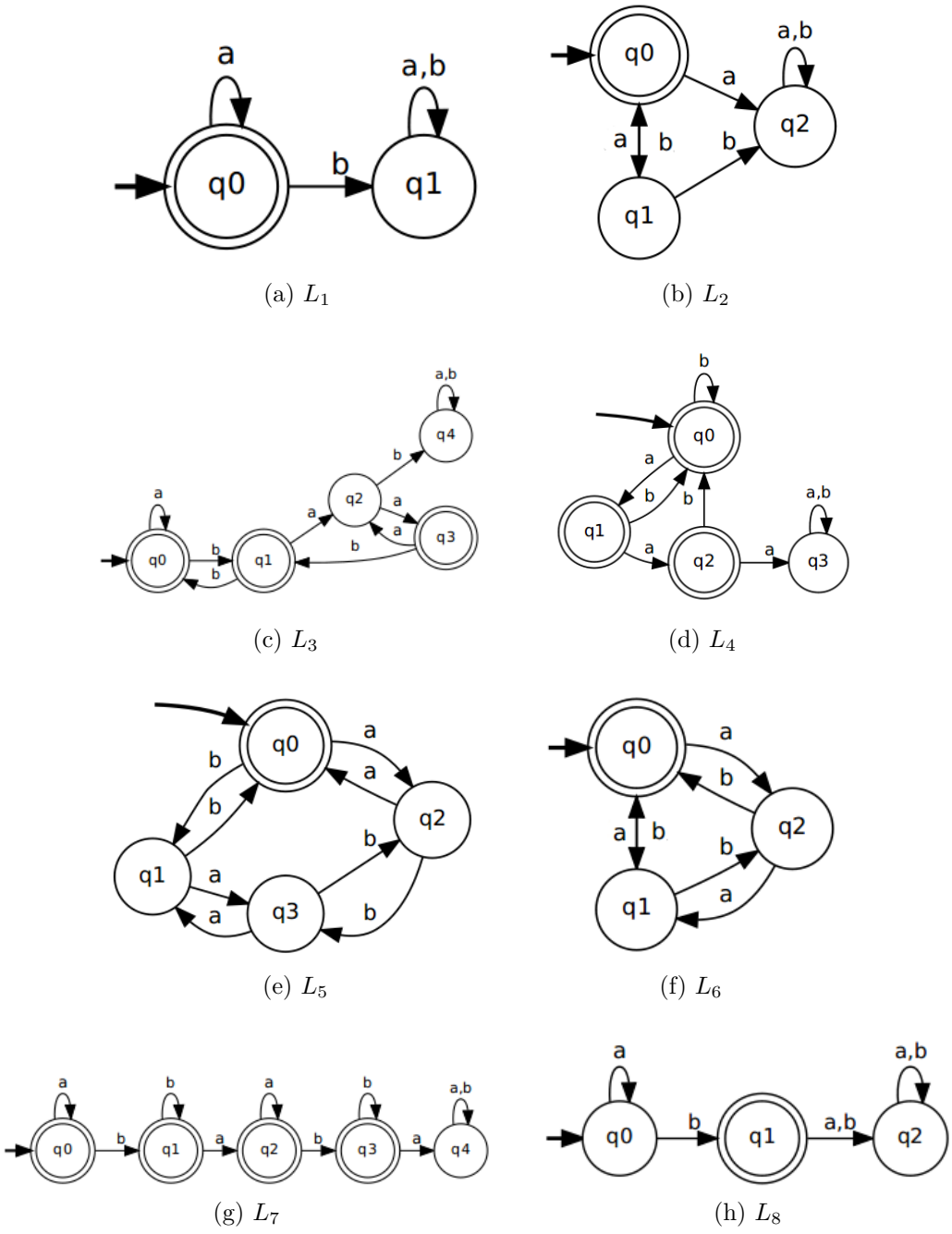
Tomita in [51] definisce sette linguagi regolari, da lui usati per provare il suo algoritmo di inferenza grammaticale basato sulla tecnica di hill–climbing. Successivamente in [19], questo insieme viene ampliato aggiungendo altri otto linguaggi regolari. L'elenco completo dei quindici linguaggi regolari si trova in tabella 5.1 e raggiunge un massimo di sei stati con L_{10} e l'alfabeto è binario tranne in due casi, L_9 e L_{15} , in cui $|\Sigma| = 3$. Alcuni di essi sono descritti dall'espressione regolare corrispondente. Nei casi in cui l'espressione regolare fosse troppo complicata al fine di capire immediatamente la "regolarità" del linguaggio, si è preferito fornirne una descrizione informale. Ad esempio l'espressione regolare per il linguaggio L_6 è $((a(ab)^*(b|aa))|(b(ba)^*(a|bb)))^*$: non è immediato riconoscere in questo caso la regolarità aritmetica posseduta dal linguaggio, costituito da tutte le stringhe in cui il numero dei caratteri a differisce dal numero dei caratteri b per un numero divisibile per tre.

L_1	a^*
L_2	$(ba)^*$
L_3	Ogni stringa che non contiene un numero dispari di a consecutivi dopo un numero dispari di b consecutivi
L_4	Ogni stringa che non contiene la sottostringa aaa
L_5	Ogni stringa che contiene un numero pari di a ed un numero pari di b
L_6	Ogni stringa in cui il numero di a contenute differisce dal numero di b contenute per un numero divisibile per tre.
L_7	$a^*b^*a^*b^*$
L_8	a^*b
L_9	$(a^* + c^*)b^*$
L_{10}	$(aa)^*(bbb)^*$
L_{11}	Ogni stringa che contiene un numero pari di a ed un numero dispari di b
L_{12}	$a(aa)^*b$
L_{13}	Ogni stringa che contiene un numero pari di a
L_{14}	$(aa)^*ba^*$
L_{15}	$bc^*b + ac^*a$

Tabella 5.1: Linguaggi Tomita

5.1.2 Dataset casuale

Un dataset di *DFA* target sicuramente più probante dalla complessità paragonabile a quella delle applicazioni reali è stato creato estraendolo in maniera casuale. La librearia GI-learning non metteva a disposizione un metodo per l'estrazione di *DFA*



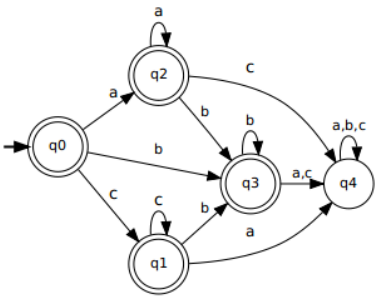
(a) L_1 (b) L_2

(c) L_3 (d) L_4

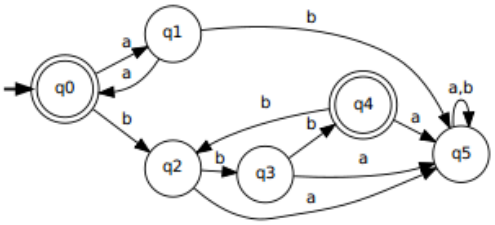
(e) L_5 (f) L_6

(g) L_7 (h) L_8

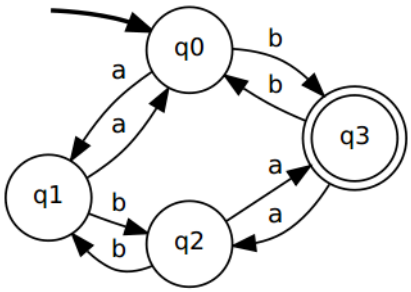
Figura 5.1: Linguaggi di Tomita



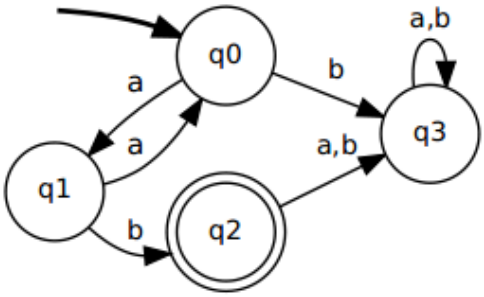
(i) L_9



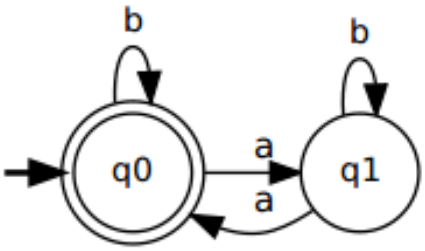
(j) L_{10}



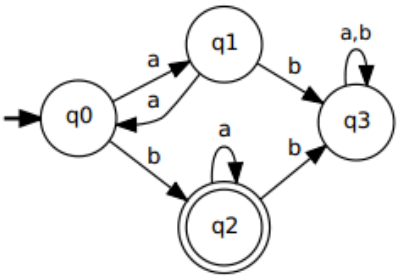
(k) L_{11}



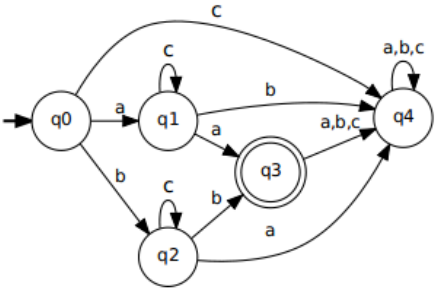
(l) L_{12}



(m) L_{13}



(n) L_{14}



(o) L_{15}

Figura 5.1: Linguaggi di Tomita

casuali ed allo scopo si è utilizzata una libreria esterna ,la LearnLib[2]. Mediante la stesura di codice ad hoc ci si è interfecciati con la LearnLib e con il metodo *randomICDFA* sono stati creati i *DFA* casuali, infine si è convertiti i *DFA* ottenuti nel formato specifico per la lettura e il caricamento degli automi all'interno di GI-learning. In precedenza si è detto che le performances sugli algoritmi di *GI* sono influenzate da diversi fattori ta cui la dimensione e il numero di stati del *DFA* target che quindi sono parametri che vanno impostati in maniera oculata. Sono stati creati sedici *DFA* casuali con dimensione dell'alfabeto crescente che varia in $\{2, 5, 10, 36\}$ e numeri di stati in $\{5, 10, 20, 50\}$. Come dimensione massima dell'alfabeto si è scelto trentasei perchè corrispondente alla dimensione dell'alfabeto italiano più le dieci cifre del sistema decimale. Il limite superiore del numero di stati del *DFA* target è stato fissato a cinquanta ricalcando la dimensione del *DFA* target scelto nelle varie competizioni di *GI* come ABBANDINGO e STAMINA.
Data la complessità degli automi non è stata possibile la rappresentazione visuale come avvenuto per il dataset Tomita.

5.2 Generazione di campioni da un DFA

Se si decidesse di generare i campioni come stringhe casuali fino a una certa lunghezza in cui ogni simbolo costituente una stringa è estratto in maniera uniforme tra tutti i simboli dell'alfabeto si incorrerebbe nel problema che si verrebbe ad avere un insieme non bilanciato a causa della predominanza di campioni negativi dato che tipicamente su Σ^* sono in minoranza le stringhe accettate da un *DFA*. Inoltre per l'addestramento di un classificatore e anche per un algoritmo di *GI* è molto importante avere un insieme caratteristico (definizione 2.3) o perlomeno un insieme di stringhe che raggiunge gran parte degli stati e delle transizioni di un *DFA*. In [56] è descritto un algoritmo in grado di assicurare con altà probabilità la completezza strutturale dei campioni e di produrre campioni bilanciati. L'algoritmo effettua un **random walk** sul *DFA* target e si rimanda al riferimento per i dettagli. La proprietà dei campioni di essere strutturalmente completi(c'è solo una probabilità alta che lo siano ma non la certezza) potrebbe condurre a delle stime ottimisticamente biased dell'algoritmo *ObPA* dato che in uno scenario reale invece i campioni a disposizione dell'utente sono di norma del tutto casuali (anche con random walk lo sono ma come detto in modo da esplorare tutti i percorsi del *DFA*) . Nonostante ciò si è deciso di utilizzare comunque *random walk* perchè questa procedura è diventata uno standard de facto per generare i campioni nelle competizioni tra algoritmi riguardanti *GI*.

5.2.1 W-Method

Per la generazione di un *test set* per fare model evaluation è improbabile che delle stringhe generate casualmente testino adeguatamente un classificatore. Inoltre per

[2] http://www.learnlib.de/

la generazione del *test set* dacade la valenza del discorso che campioni strutturalmente completi potrebbero condurre a sovrastimare positivamente un classificatore, assunto che rimane valido per il *training set*, anzi il possedimento di questa proprietà dei dati nel *test set* è ricercata per testare in maniera affidabile un classificatore che difatti approssima $\mathcal{L}$. Quindi l'utilizzo di *random walk* per la generazione del *test set* è assolutamente appropriato.
ObPA però necessita anche del confronto dell'ipotesi intermedia H ottenuta ad ogni *round* con il *DFA* target ma quest ultimo non è noto ma è approssimato da un classificatore[3] preliminarmente costruito tramite *SVM* a partire dai campioni disponibili. Di conseguenza l'*EQ* come spiegato in 4.3.5 va implementata sottoponendo una serie di stringhe sia al classificatore che al *DFA* ipotesi H ed è fondamentale che queste stringhe siano selezionate attentamente. Estrarre le stringhe con l'algoritmo *random walk* sul *DFA* target in questo caso genera delle stringhe particolarmente significative ma solo per il *DFA* H e possibilmente non rappresentative del *DFA* target dato che è ignoto.
E ancora sia *random sampling* che *random walk* possono produrre un *test set* che ha alcuni campioni in comune con il *training set* con cui è stato addestrato il classificatore che approssima l'oracolo ottenendo una possibile corrispondenza mistificatoria dell' *EQ*.
Inoltre per la *ideal version*, è possibile confrontare il *DFA* finale inferito da *ObPA* con il *DFA* target ed anche in questo caso si ripresentano i problemi precedentemente esposti cioè eseguendo *random walk* su uno dei due automi (o il target o l'ipotesi) potrebbe essere prodotto un insieme di stringhe per testare la similarità dei due linguaggi che non esplora parte dell'altro *DFA* (quello su cui non si è eseguito *random walk*) risultando quindi non del tutto appropriato allo scopo. Pertanto adesso è descritto l' algoritmo **W-Method** [10] che è esente dai problemi summenzionati.

Algoritmo W-Method

Tramite il W-Method nel contesto della *GI* si realizza un' *EQ* esatta impiegando solo *MQ* [6]. Infatti esso crea, senza la necessità di avere il *DFA* target A, un *test set* X dall'ipotesi H in modo che se $\forall x \in X \quad \lambda^A(x) = \lambda^H(x)$ allora $A \cong H$ (per rispondere alle *MQ* lato target non è strettamente necessario avere il *DFA* target ma si suppone uno scenario in cui si possa conoscere l'esito di $\lambda^A(x)$ senza la sua conoscenza). Quindi il W-Method si può usare per realizzare un'*EQ* in maniera esatta tramite *MQ* tuttavia il numero di stringhe generate dal *W-Method* può essere enorme e quindi subentrano problemi di efficienza. In questa tesi è incognito anche l'esito delle *MQ* che è approssimato tramite le *SVM* quindi l'*EQ* realizzata tramite le stringhe prodotte dal *W-Method* sarà comunque un'approsimazione. Inoltre quanto detto è vero se sono verificate alcune condizioni tra cui il numero esatto di stati del *DFA* target che in *ObPA* non è noto nella *release version* e può essere solo

[3] il classificatore approssima l'Oracolo, ma in ultima istanza ad essere approsimato è il linguaggio alias *DFA* target dato che un Oracolo nel caso ideale ha il *DFA* target e tramite esso risponde alle query

stimato producendo quindi un insieme di stringhe che approssimano l'equivalenza senza garantirla. Gli altri requisiti del *W-Method* sono [6]:

- H ed A sono deterministici
- H è canonico
- Non ci sono stati irraggiungibili nè in H nè in A
- Si conosce $\|A\|$
- H e A devono essere completamente specificati cioè da ogni stato per ogni Σ si raggiunge uno stato che non sia lo stato pozzo(questa condizione può essere rilassata [6])

Si descrive brevemente il W-Method per sommi capi. La ratio è che nel target si potrebbero avere degli extra-stati che non vengono esplorati con il *test set* generato con metodi classici e che potrebbero causare un comportamento indesiderato ed alterare la valutazione senza averne avvisaglie. Si inizia stimando il numero di stati nel *DFA* target (in modo che il test set esplori tutti gli stati del target). É importante eseguire il *W-Method* sul *DFA* che ha il numero minore di stati (quando si usa il W-method nella *ideal version* per testare la similarità tra il target ed H. Invece durante le generazioni delle ipotesi intermedie si ha soltanto H e quindi il W-Method va eseguito necessariamente su questo *DFA*)stimando gli stati dell'altro.
Ipotizzando in chiave espositiva che il *DFA* col numero minore di stati sia H si procede generando lo *state cover* C su H cioè $C = \{c : \forall q \in Q^H \ \hat{\delta}^H(q_\epsilon, c) = q\}$ cioè l'insieme di stringhe necessarie per raggiungere ogni stato a partire dallo stato iniziale. Due stati q_1, q_2 di H sono detti *W-distinguibili* se dato un insieme di stringhe $W \subseteq \Sigma^* \ \exists x \in W : ((\lambda^H_{q_1}(x) = 1 \wedge \lambda^H_{q_2}(x) = 0) \vee (\lambda^H_{q_1}(x) = 0 \wedge \lambda^H_{q_2}(x) = 1))$. W è detto un *characterization set* se tutti gli stati distinti di H (essendo H minimo tutti gli stati sono distinti) presi a due a due sono *W-distinguibili*. Definito $k = \|A\| - \|H\|$ essere una stima (si parla di stima perchè il target A è ignoto) di quanti stati ha il target in più dell'ipotesi si ha che il *test set* Y è:

$$Y = C(\{\epsilon\} \cup \Sigma \cup \cdots \cup \Sigma^{k+1})W$$

Il difetto di questa tecnica è che la complessità dipende in maniera esponenziale [10, p. 181] da $\|A\| - \|H\|$ e quando questa differenza di stati diventa grande il metodo è ingestibile e il numero di stringhe generate dal *W-Method* diventa enorme.

5.2.2 Generazione dei campioni per SVM

Per effettuare l'addestramento delle *SVM* è necessario creare un *training set*. Anche la dimensione del *training set* e del *test set* è estremamente rilevante per le performances di un classificatore statistico e quindi è necessario scandagliare il comportamento dell'algoritmo al variare della dimensionalità di questi parametri. Attenendosi

a quanto fatto nella competizione STAMINA [56] si crea un *sample set* tramite *random walk*, che fornisce stringhe in maniera equamente bilanciata tra accettanti e rigettanti, sul *DFA* target e da esso si estrapolano due insiemi **disgiunti**[4] *training set* e un *test set*. In STAMINA il *sample set* è di dimensione 20000 e da esso si estrae un *test set* di dimensione 1500 e dalle rimanenti stringhe si estrae con una procedura che non è rilevante descrivere il *training set* di quattro diverse dimensioni che dipendono da un valore impostato dall'esterno ma grossolanamente si può affermare che le quattro dimensioni del *training set* sono $\{20000, 10000, 5000, 2500\}$. Ricalcando da vicino quanto appena descritto in STAMINA si è creato con *random walk* un insieme di 1500 stringhe da cui si è estratto un insieme di 1000 elementi per il *test set* e 500 per il *training set*. Il metodo implementato per effettuare lo *splitting* (realizza un "sampling without replacing") realizza anche la stratificazione (cfr. App. D.2.1) e funziona anche nel caso che i campioni non siano equamente bilanciati. Rispetto a STAMINA il *test set* anzichè 1500 elementi ne ha 1000 perchè è comunque un numero di campioni sufficientemente significativo e perchè per alcuni automi è risultato complicato generare un numero elevato di campioni con *random walk* come si spiegherà a breve. Per lo stesso motivo il *training set* è molto ridotto rispetto a STAMINA e ciò potrebbe costituire un notevole svantaggio. Un'altra ragione per un *training set* ridotto è pratico dato che SVM^{light} ha dei seri problemi di convergenza quando la dimensione dell'insieme di addestramento diventa dell'ordine di 2000–3000 elementi quindi aumentarne la dimensione potrebbe rendere l'algoritmo estremamente lento. Si puntualizza che la velocità dell'algoritmo di *SVM* di trovare una soluzione dipende non solo da quanti sono in numero i campioni ma anche dagli specifici dati, sia da un punto di vista di contingenza di quest ultimi che dalla difficoltà nel separarli perchè essi riflettono la complessità del linguaggio target che rappresentano cioè più il target è complesso e più è probabile che i dati siano difficili da separare con *SVM*. Da un punto di vista di un'eventuale utilizzazione il limite di 500 campioni non costituisce un limite dato che i campioni in questo caso sono quelli forniti dall'utente e tipicamente non sono un numero ingente, ma ciò ha impedito di testare il comportamento di *ObPA* su *training set* grandi e uniformarsi a quanto fatto nelle varie competiziono di *GI*.
Nella *ideal version* del codice il *setting* per la generazioni dei campioni per l'addestramento è quello appena descritto. Nella *release version* i campioni non vanno generati su un*DFA* target perchè ignoto ma vanno inseriti dall'esterno. L'utente dovrà provvedere ad inserirli su file rispettando uno specifico formato:

$$etichetta \quad simbolo\_alfabeto \quad simbolo\_alfabeto \quad simbolo\_alfabeto \ldots$$

Ad esempio se l'alfabeto è $\Sigma = \{0\, tao\, 2\}$ e la stringa $tao2002tao2$ è accettante, nel file ci dovrà essere una riga siffatta:

$$1 \quad tao \quad 2 \quad 0 \quad 0 \quad 2 \quad tao \quad 2$$

[4] Affinchè le performances non siano ottimisticamente biased è necessario che l'addestramento avvenga su un *training set* che non ha elementi in comune con il *test set*

Il primo e unico 1 è l'etichetta per una stringa accettante, l'etichetta per una stringa rigettante deve essere -1. Inoltre è necessario inserire nel file prima gli esempi positivi e poi a seguire quelli negativi. Sono ammessi duplicati perchè potrebbero essere significativi.

Per alcuni linguaggi del dataset Tomita sono stati riscontrati dei problemi nella generazione del *sample set* tramite *random walk*. Si ha che *random walk* nella ricerca di campioni positivi effettua un cammino probabilistico nel *DFA*, che si arresta con una probabilità tale da produrre campioni che superano la profondità del *DFA* (cioè di una lunghezza tale da esplorare con buona probabilità tutti gli stati del grafo); ma se il *DFA* ha uno stato pozzo può accadere che molte delle stringhe casuali generate arrivino allo stato pozzo nel quale il cammino casuale termina perchè per ogni simbolo dell'alfabeto da uno stato pozzo si termina ancora nello stesso stato pozzo. In *DFA* piccoli con pochi stati come i Tomita è altamente probabile terminare casualmente in uno stato pozzo e quindi si riescono a generare solo pochi campioni positivi anche facendo molti tentativi ,infatti per i linguaggi incriminati che sono $L_1, L_2, L_8, L_9, L_{10}, L_{12}, L_{14}, L_{15}$ neanche impostando una soglia altissima del numero massimo di tentativi da fare ci si è avvicinati ai 1500 oppure 500 campioni richiesti (la soglia è 600000 ed in alcuni casi come L_7 la generazione dei campioni con *random walk* risulta lenta). Allora per i linguaggi suddetti malgrado quanto detto in precedenza (uniformarsi a STAMINA) si è proceduto gioco forza alla generazione casuale dei campioni. É importante fare notare che i campioni positivi dei linguaggi problematici sono stati costruiti casualmente ma sfruttando sia la conoscenza della struttura dell'automa che del linguaggio evitando nella fattispecie i problemi relativi al *random sampling* per il *test set* precedentemente esposti. Ad esempio per il linguaggio L_2 nel caso si debbano generare $num\_samples$ campioni positivi si estrae un numero casuale tra 0 e $2 * num\_samples$ e il numero estratto indica quante volte consecutivamente andrà ripetuta la sottosequenza *ba*. Considerazioni analoghe per gli altri linguaggi problematici. La generazione manuale dei campioni per i linguaggi suddetti è relativa solo ai campioni positivi, quelli negativi vengono generati ancora con *random walk* (una versione modificata all'occorrenza) a partire dai campioni casuali positivi generati in precedenza[5].

5.2.3 Comparazione di DFA target e DFA inferito da ObPA

Nella sottosezione 5.2.1 è stato detto che ,nella *ideal version*, quando *ObPA* termina è necessario confrontare il *DFA* target con il *DFA* inferito dall'algoritmo *ObPA*. Il *test set* per effettuare il confronto va creato con il W-Method perchè: rende irrilevante l'eventuale presenza nel *test set* di campioni già usati in fase di addestramento , genera un insieme di campioni estremamente significativo per il confronto, non necessità di un'impostazione manuale del numero di campioni nel *test set* che è automaticamente tornato dal W-Method. Non sarà richiesta neanche la stima del

[5] *random walk* genera i campioni negativi a partire da quelli positivi effettuando un numero casuale di modifiche su di essi.

numero di stati del *DFA* target dato che quest ultimo in *ideal version* è noto. Nonostante i pregi evidenziati il W-Method ha originato molti problemi pratici ed è stato necessario provvedere con rimedi ad hoc. Nei seguenti casi non si usa il W-Method per la generazione dei campioni per il confronto:

1. Differenza di stati tra *DFA* target e *DFA* inferito maggiore di 10
2. *Test set* creato dal W-Method troppo grande, maggior di 500000 campioni
3. *Test set* creato dal W-Method troppo piccolo, minore di 100 campioni

Nel caso 1 a causa della complessità esponenziale del W-Method dipendente dalla differenza del numero di stati tra *DFA* target e *DFA* inferito il W-Method non termina in tempi accettabili. Nel caso 2 il W-Method termina ma ritorna un numero eccessivo di campioni che renderebbero eccessivamente lenta la fase successiva in cui si devono calcolare le statistiche sul *test set* (anche se i due *DFA* differiscono per pochi stati, ma hanno un numero elevato di stati il W-Method potrebbe generare un numero considerevole di stringhe; anche per *DFA* piccoli che differiscono per pochi stati ma con un grande alfabeto vengono generate tante stringhe), quindi si è fissata la soglia di 500000 campioni. Il caso 3 deriva principalmente dalla situazione in cui il *DFA* inferito da *ObPA* e il *DFA* target hanno lo stesso numero di stati oppure sono automi piccoli (con pochi stati) che differiscono per pochi stati: in tali situazioni il numero di campioni generati dal W-Method è esiguo.
In tutti e tre i casi si passa automaticamente all'utilizzo di *random walk* (nei casi 2 e 3 è richiesta l'esecuzione di entrambi gli algoritmi perchè per capire di essere ricaduti in uno dei casi summenzionati e passare all'esecuzione di *random walk* è necessaria la preventiva esecuzione del *W-Method*).
Il *W-Method* va sempre invocato usando come chiamante il *DFA* con meno stati (e questo può essere sia il *DFA* inferito che il *DFA* target).
La comparazione (cioè verificare la rispondenza o meno dei campioni sui due *DFA*) invece avviene usando sempre il *DFA* inferito come "test" cioè si deve testare come classifica il *DFA* inferito rispetto al *DFA* target ad esempio si ha un falso positivo quando il *DFA* inferito classifica un campione come positivo e il *DFA* target come negativo.
Quando si rientra in uno dei tre casi e si usa *random walk*, siccome quest ultimo rispetto al *W-Method* può essere meno attendibile, si creano 1500 campioni sia sul *DFA* target che su quello inferito ; si hanno due *test set* allo scopo di non trascurare aspetti significativi dei due linguaggi. I campioni sono bilanciati (*random walk* genera campioni bilanciati) tuttavia quando si calcolano le statistiche sui due *test set* (cioè le misure, cfr. App. D.5) in entambi i casi si usa come "test" il *DFA* inferito e ad esempio si ha un falso positivo quando il *DFA* target classifica un campione come negativo e il *DFA* inferito da *ObPA* lo classifica come positivo. Inoltre anche se si generano 1500 stringhe bilanciate sul *DFA* inferito con *random walk*, esse possono risultare non bilanciate rispetto al *DFA* target e questo spiega perchè nelle statistiche potrebbe comparire un numero di stringhe positive e negative differente dalla ripartizione dicotomica di 750 positive e 750 negative.

Una gestione particolare si è resa necessaria nel caso del Tomita dataset perchè come spiegato nella sottosezione 5.2.2 per i linguaggi di Tomita L_1,L_2,L_8,L_9,L_{10},L_{12},L_{14},L_{15} *random walk* non riesce a generare i campioni e lo stesso accade in questo contesto e per essi si è dovuto generare i campioni manualmente (alla stregua di quanto fatto in fase di addestramento 5.2.2). Inoltre per questi linguaggi di Tomita problematici si generano i 1500 campioni del *test set* solo sul *DFA* target, perchè il *DFA* inferito da *ObPA* potrebbe essere simile al target o molto diverso (il punto è che non lo si sa in anticipo) e quindi *random walk* potrebbe "impigliarsi" o meno e comunque non si possono generare le stringhe accettanti manualmente perchè non si conosce il linguaggio corrispondente al *DFA* inferito da *ObPA*.

Le misure riportate in uscita (che come appena spiegato in alcuni casi sono duplici perchè relative a due *test set*) sono quasi tutte descritte nell'Appendice D.5 e nella fattispecie sono:

- TP,FP,TN,TP
- Campioni positivi e negativi (valutati sul *DFA* target)
- Accuracy
- $Precision^+$, $Recall^+$, $Recall^-$
- F-measure calcolata per i campioni positivi
- Balanced Classification Rate (BCR) (misura che tiene contemporaneamente conto dei campioni positivi e negativi)
- MCC(Mattehws correlation coeffient)

Particolare rilevanza ai fini della valutazione come spiegato nell'Appendice D.5 è stata data alla misura MCC.
Infine si puntualizza che il *W-Method* pressupone due automi canonici per cui sia il *DFA* inferito da *ObPA* che il *DFA* target vanno preventivamente minimizzati.

5.2.4 Generazione dei campioni per un EQ

Per l'effettuazione di un *EQ* approssimata è necessario predisporre un insieme di stringhe (cfr. sottosezione4.3.5). L'utilizzo del *W-Method* per la generazione dei campioni è una scelta naturale. In questo caso si ha un unico *DFA* cioè l'ipotesi H intermedia prodotta da *ObP* e dall'altra parte si ha il classificatore inizialmente creato che approssima il *DFA* target. Il *W-Method* va eseguito su H che deve essere preventivamente minimizzato dato che *ObP*, come spiegato nella sottosezione 3.4.2, assicura che il *DFA* finale sia minimo ma tale proprietà non è garantita per le ipotesi intermedie (a differenza dell'algoritmo L^*). *ObPA* utilizza il *W-Method* ,per generare le stringhe su cui effettuare un *EQ* approssimata, applicandolo all'ipotesi H ma nel momento in cui , per uno dei tre motivi già esposti nella sottosezione 5.2.3, si è

impossibilitati ad utilizzarlo si impiega *random walk* e da quel momento in poi si utilizza *random walk* anche per le altre eventuali ipotesi intermedie H dato che è altamente probabile che i problemi in cui si è incorsi al passo corrente siano presenti addirittura amplificati nei successivi *round* dell'algoritmo. Essendo il numero di stati del target ignoto si utilizza un algoritmo adattivo sul numero di stati dell'ipotesi per effettuarne la previsione: basandosi sulla constatazione che al crescere del numero di stati di H la previsione del numero di stati del target deve essere sempre più vicina a $\|H\|$ pena un numero di campioni esorbitante ritornato dal *W-Method* si ha che

$$num\_stati\_target = floor(k * \|H\|)$$

con k un valore $\in \{7, 5, 2.5, 1.8, 1.5\}$ in maniera corrispondente al superamento dei rispettivi stati di $\|H\| \in \{2, 3, 6, 10, 16\}$. Per meglio chiarire si ha che per esempio se $3 <= \|H\| < 6$ $k = 2.5$. Inoltre, se $\|H\| > 15$ $k = 1.1$ Se le stringhe generate col W-Method sono in numero esiguo si tenta di aumentare la previsione sul numero degli stati del target e verificare se rieseguendo il *W-Method* si riesce ad incrementare il numero di campioni prodotti superando la soglia minima di 100 campioni.
Nel caso si usi *random walk* si cerca di generare 1500 campioni, se non ci riesce per motivi analoghi ai linguaggi Tomita si usa un *test set* ridotto ma nel caso che non vengano creati neanche 100 campioni si ha il lancio di un'eccezione e la terminazione di *ObPA*.
Parla del fatto che la soglia per l'eq si è impostata con un'accuracy del 90% ma non va bene questa misura. Se si ha a che fare con un *DFA* come Tomita2 ad esempio che genera molte più stringhe negative che positive si ha che molte delle stringhe estratte dal W-Method se non tutte sono negative e nel primo round di *ObPA* l'iptesi è il DFA che rigetta tutte le stringhe l'accuracy è al 100% e nonostante un buon classificatore si ottiene un risultato errato. Allora si è usato Matthews come misura che tenendo conto di entrambe le classi evita questo problema.

5.3 ALtro

Allo stato dell'arte l' *ObP* costituisce il secondo algoritmo di riferimento nell'ambito dell'apprendimento di linguaggi regolari. L'algoritmo più performante è invece il più recente TTT algorithm [48].
Inserire discorso che la lineare separabilità (hard margin) non è direttamente implementabile in svm light e si usa il parametro C=1. SI potrebbe provare ad usare un'altra libreria che ha questa lineare separabilità o ad usare una ricerca nello spazio del parametro C (perchè anche con dati lineramente separabili per evitare overfitting e problemi con gli outliers si usa spesso soft margi lo stesso).
Inserire il discorso che in ideal version alla fine è comunque necessario confrontare l'ipotesi finale col target per rendersi conto del dfa inferito quanto è buono. Infatti che l'ipotesi sia verificato tramite un'equivalence query essere simile al classificatore non è sufficiente in quanto il classificatore è già di per sè un'approssimazione del target e quindi un'ipotesi che approssima bene il classificatore potrebbe non approssimare bene il target e quindi un confronto finale tra target e ipotesi è necessario.

Inserire il discorso dell'esplosione del DFA. E che questo è dovuto anche al fatto che quando viene tornato un controesempio che non è un controesempio si fanno degli split errati. Siccome la versione implementata in librearia è ONEGlobally questa può generare più di uno split alla volta a differenza di ONELocally. Inoltre il controesempio viene sfruttato più di una volta, quindi anche un solo errore può condurre nel controesempio può condurre ad un DFA ipotesi completamente divergente dal target. Quindi potrebb essere meglio usare ONELocally che però è stato implementato ma non funzionava bene al 100 per 100.
Fai vedere le difficoltà anche nell'approssimare con OneGlobally OBP il linguaggio che il classificatore rappresenta, e non solo difficoltà nell'approssimare il linguaggio target col classificatore in addestramento.
Parla del fatto che è pssibile provare ad implementare SVM PERFCT per velocizzare il codice e provare così ad usare un training set di dimensioni maggiori e si spera migliori risultati.
Parla del caso di campioni non bilanciati che lo splitting permette di gestire campioni non bilanciati e che c'è un parametro per gestire i campioni non bilanciati. Nessun esperimento fatto su di essi.
Giustifica risultati anche perchè si approssimano anche le MQ. Fai vedere che comunque esistono dei modelli complessi con cui si hanno buoni risultati (anche se la complessità è tutta da vedeere e non è decisa solo dagli stati dall'alfabeto e dalla profondità, ma dipende anche da quanto sono buoni i campioni estratti col random walk). Illustra i set di esperimenti e i risultati.
Scrivi che non c'è un feedback per la bontà del classificatore come spiegato nel Capitolo 4 ma comunque si ha l'accuracy in ObPA1 e ObPA2 e il margine in ObPA3 (vedi se hai anche altro in obpa 3). E un metodo alla femminina per capire se il classificatore è buono è vedere se il classificatore viene costruito velocemente oppure SVM perde tempo.

Conclusioni

Inserire il discorso che è possibile provare ObPA nel caso si approssimano solo le eq. query e le MQ sono esatte. CHe lo scenario cui si è ricondotti qui è molto simile agli algoritmi di passive learning. Gli algoritmi di active learning invece sono più efficienti proprio perchè hanno piu informazioni e qui riducendo a zero l'informazione aggiuntiva riconducendoci a uno scenario operativo degli algoritmi di passive learning ma sempre inseriti in un algoritmo di active learning si hanno riultati non ottimi. Diminuendo solo parzialmente l'informazione (appr. solo l'eq. come fanno molti lavori in letteratura sicuramente si otterrebbero risultati migliori).
Dato che il W-Method ha una complessità esponenziale (Vedi CHow) esistono metodi simili più efficienti come il Wp-Method e HSI method che si può pensare di implementare per provare a risolvere i problemi col W-Method
Inserire il fatto che la versione ONEGlobally di Observation Pack probabilmente non è adattissima in questo algoritmo. Perchè altra versione come OneLocally fa solo uno split e quindi se il controesempio è errato c'è solo uno split errato invece con OneGlobally ci possono essere (è molto probabile) più split e l'ipotesi diverge molto di più in una direzione errata. (Comunque sottolinea che un solo controesempio errato può essere catastrofico, perchè in pratica saranno 2 linguaggi diversi. Se ad esempio avviene uno split laddove non doveva avvenire indietro non si può tornare.

Appendice A

Preliminari

Lo scopo di quest'appendice è di stabilire una comune sintassi e semantica per concetti che sono rilevanti in tutta la tesi. Le definizioni e le notazioni qui introdotte sono essenziali per la maggior parte dei capitoli e quindi andrebbero lette.
L'appendice è divisa concettualmente in quattro parti. Nella prima parte si introdurranno questioni puramente matematiche e si definiranno grammatiche e linguaggi e automi e annesse proprietà, nella seconda parte si delinea una notazione e alcuni dettagli implementativi per l'algoritmo *ObP*, nella terza parte sono prese in esame le SVM infine nell'ultima parte si tratteranno gli argomenti relativi all'addestramento di un classificatore statistico con particolare attenzione per le SVM.

A.1 Notazione matematica

L'obiettivo di questa sezione è di introdurre i concetti matematici propedeutici per questa tesi. Senza dubbio una conoscenza matematica di base è necessaria e chiaramente non può essere introdotto ogni singolo elementare concetto.

A.1.1 Insiemi

Con $\mathbb{N}$ si indica l'insieme di numeri naturali interi non negativi incluso 0 (cioè $\mathbb{N} = 0, 1, 2, \ldots$). L'insieme di interi positivi è denotato da $\mathbb{N}^+$. Si definisce con $\mathbb{B} = \{0, 1\}$ l'insieme di valori booleani dove 0 è associato al valore logico *falso* ed 1 al valore logico *vero*.
Dato un generico insieme X, $|X|$ denota la sua cardinalità, cioè il numero di elementi che contiene.
Si definisce l'operazione di elevamento a potenza di un insieme per un altro con la notazione A^B come l'insieme di tutte le funzioni che vanno dall'insieme B all'insieme A cioè se $f \in A^B$ allora $f : B \to A$.

A.1.2 Funzioni

Per qualsiasi $x \in \mathbb{R}^{\mathbb{N}}$ si definisce il suo *supporto* come il numero di coordinate diverse da zero:

$$supp(x) = \{i \in \mathbb{N} : x_i \neq 0\}$$

Si definisce l'*inner product* di due elementi $x, y \in \mathbb{R}^A$ con:

$$x \star y = \sum_{a \in A} x_a y_a$$

Ad esempio se $A = \{1, 3\}$ e $x = \{(1, 7), (3, 9)\}$ e $y = \{(1, 5.2), (3, 4)\}$ si ha che

$$x \star y = 7 \times 5.2 + 9 \times 4 = 72.4$$

A.1.3 Relazione d'equivalenza

Una relazione binaria riflessiva ,simmetrica e transitiva $\approx \subseteq X \times X$ su un insieme X è detta una ***relazione d'equivalenza***. Dato un insieme X ed un elemento $x \in X$ si denota con $[x]_\approx = \{x' \in X \mid x \approx x'\}$ la *classe di equivalenza* di x(rispetto alla relazione d'equivalenza $\approx$).
Una relazione d'equivalenza $\approx$ su un insieme X si dice che *satura* un sottoinsieme $X' \subseteq X$ se e solo se X' è l'unione di alcune delle classi d'equivalenza di $\approx$. In simboli si ha:

$$X' = \bigcup_{x \in X'} [x]_\approx \ ,$$

e ogni classe di equivalenza $[x]_\approx$ di $\approx$ o è un sottoinsieme o è disgiunta da X'.
Il ***quoziente*** (o insieme quoziente) di X rispetto a una relazione d'equivalenza $\approx$ è definito come l'insieme di tutte le classi d'equivalenza, ed è indicato da $X/\approx = \{[x]_\approx \mid x \in X\}$. L'***indice di una relazione d'equivalenza*** $\approx$ è definito come il numero di classi d'equivalenza, cioè è uguale a $|X/\approx|$. Una *partizione* di un insieme X è un insieme P i cui elementi, detti *blocchi* ed indicati con C, sono sottoinsiemi (disgiunti e non vuoti) dell'insieme X tali che:

1. se $C \in P$ allora $C \neq \emptyset$
2. se $C_1, C_2 \in P$ e $C_1 \neq C_2$ allora $C_1 \cap C_2 = \emptyset$
3. se $a \in X$ allora esiste $C \in P$ tale che $a \in C$ (è un altro modo di dire che l'unione di tutti gli insiemi C deve formare X)

Il *quoziente* di X forma una *partizione* di X

A.2 Linguaggi e grammatiche

A.2.1 Alfabeto, stringhe e linguaggi

Alfabeto

Si definisce l'***alfabeto*** Σ un qualsiasi insieme finito e non vuoto di simboli.

Stringhe

Una ***stringa*** è definita come una sequenza di simboli presi da un alfabeto. Cioè una stringa s definita su Σ è una sequenza $s = a_1 \dots a_n$ tale che $a_i \in \Sigma$.
$|s|$ denota la lunghezza della stringa s.
La *stringa vuota* è indicata con ϵ e $|\epsilon| = 0$.
Con Σ^* si denota l'insieme di tutte le possibili stringhe ottenibili sull'alfabeto Σ. Inoltre con Σ^+ si denota l'insieme $\Sigma^* - \{\epsilon\}$
I singoli simboli costituenti una stringa $w \in \Sigma^*$ sono indicati con w_i con $0 \leq i < |w|$ quindi $w = w_0 w_1 \dots w_{|w|-1}$. Per qualche intero nell'intervallo $I \subseteq [0, |w|], w_I$ è la stringa risultante prendendo solo le posizioni in w corrispondenti agli indici in I . Quindi $w_{[0,k)}$ è il prefisso di w di lunghezza k, e $w_{[k,|w|)}$ è il suffisso di w che inizia all'indice k(compreso). Si osservi che $w_{[0,0)} = w_{[|w|,|w|)} = \epsilon$ e $w_{[0,|w|)} = w$.
Un ***prefisso*** di una stringa $w \in \Sigma^*$ è una stringa $u \in \Sigma^*$ tale che esiste una stringa $v \in \Sigma^*$ soddisfacente $w = uv$. L'insieme di tutti i prefissi di una stringa w si indica con Pref(w)
Un ***suffisso*** di una stringa $w \in \Sigma^*$ è una stringa $v \in \Sigma^*$ tale che esiste una stringa $u \in \Sigma^*$ soddisfacente $w = uv$. L'insieme di tutti i suffissi di una stringa w si indica con Suff(w)
Una **sottosequenza** di una stringa è una qualsiasi stringa ottenuta rimuovendo dalla stringa di partenza zero o più simboli non necessariamente consecutivi.

Linguaggi

Un linguaggio L su un alfabeto Σ è un qualsiasi sottoinsieme di stringhe di Σ^*
Si definisce l'insieme dei prefissi di un linguaggio L:

$$\text{Pref}(L) = \bigcup_{w \in L} \text{Pref}(w)$$

e l'insieme dei suffissi di L:

$$\text{Suff}(L) = \bigcup_{w \in L} \text{Suff}(w)$$

Un linguaggio L è detto essere ***prefix-closed*** se e solo se $\text{Pref}(L) = L$. Informalmente questa proprietà di un linguaggio L viene sfruttata per indicare che qualunque stringa appartenente ad L si prende, qualunque suo prefisso deve ancora appartene ad L. Analogamente un linguaggio L è ***suffix-closed*** se e solo se $\text{Suff}(L) = L$.

La ***differenza simmetrica*** di due linguaggi L_1 e L_2 denotata con $L_1 \oplus L_2$ è tale che:

$$L_1 \oplus L_2 = \{x \in \Sigma^* : (x \in L_1 \wedge x \notin L_2) \vee (x \notin L_1 \wedge x \in L_2)$$

Un linguaggio può essere identificato mediante due tipi di descrizioni:

1. **Descrizione generativa**
   Consiste nell'utilizzare un formalismo denominato **grammatica generativa** ,introdotto da Noam Chomsky, che consiste in una serie di simboli e regole mediante le quali è possibile generare tutte e sole le stringhe del linguaggio.

2. **Descrizione descrittiva-identificativa**
   Il linguaggio è identificato o tramite un'enumerazione delle stringhe che vi appartengono o tramite una descrizione che cattura le caratteristiche delle sentenze costituenti il linguaggio,ad esempio le espressioni regolari. Un altro sistema formale identificativo sono gli **automi** .

I linguaggi possono essere classificati in base ai due tipi di descrizioni. Infatti è possibile delineare una tassonomia di grammatiche cui si farà corrispondere una classe di linguaggi. Ad ogni classe di grammatiche corrisponderà una classe di linguaggi (tutti quelli che quella classe di grammatiche è in grado di generare). E' possibile effettuare un'analoga corrispondenza tra classi di automi e classi di linguaggi, e quindi anche tra la gerarchia di automi e quella di grammatiche. Si rimanda alla sottosezione A.2.2 per una formalizzazione di questa gerarchia.

A.2.2 Grammatiche

Definizione (Grammatica generativa di Chomsky)**.** Una ***grammatica generativa di Chomsky*** è una quadrupla:

$$G = (\Sigma, V, S, P)$$

dove:
Σ è l'alfabeto, detto insieme di simboli terminali
V è l'insieme di simboli non terminali
S è il simbolo iniziale ed appartiene a V
P è l'insieme delle produzioni costituiti da una testa Ψ (il lato sinistro della produzione) e da una coda Ω aventi in generale questa forma:

$$\Psi \to \Omega \text{ con } \Psi \in (\Sigma \cup V)^* V (\Sigma \cup V)^* \text{ e } \Omega \in (\Sigma \cup V)^*$$

Si effettua una classificazione delle classi di grammatiche (e dei linguaggi ad esse associate) imponendo delle restrizioni sulle regole di produzione [9]:

- **Grammatiche di tipo zero - Unresticted** E' la classe di grammatiche più in alto nella gerarchia e per la quale non vi sono regole di restrizione da applicare alle produzioni. Sono in grado di generare la classe di **linguaggi ricorsivamente enumerabili**. Mediante l'approccio identificativo l'automa che riconosce ed accetta questi linguaggi è la **Macchina di Turing**

- **Grammatiche di tipo uno - Context Sensitive** Le regole di produzione sono così definite:
  $$\alpha_1 A \alpha_2 \to \alpha_1 \Omega \alpha_2 \text{ con } \alpha_1, \alpha_2, \Omega \in (\Sigma \cup V)^* \text{ e } A \in V$$

La classe di linguaggi che queste grammatiche sono in grado di generare è detta **context-sensitive**. Gli automi in grado di riconoscere ed identificare questi linguaggi sono detti **Linear Bounded Automata**

- **Grammatiche di tipo due - Context Free** Le regole di produzione sono così definite:
$$A \to \Omega \text{ con } \Omega \in (\Sigma \cup V)^* \text{ e } A \in V$$
La classe di linguaggi che queste grammatiche sono in grado di generare è detta **context-free**. Gli automi in grado di riconoscere ed identificare questi linguaggi sono detti **Push Down Automata**

- **Grammatiche di tipo tre - Regular** Le regole di produzione sono così definite:
$$A \to \alpha B \text{ oppure } A \to B\alpha \text{ con } A \in V, B \in (V \cup \{\epsilon\}) \text{ e } \alpha \in \Sigma^+$$
I linguaggi che queste grammatiche generano sono detti **linguaggi regolari**. Gli automi in grado di riconoscere ed identificare questi linguaggi sono detti **Finite State Automata (*FSA*)**.

Le diverse classi di linguaggi, e quindi anche di grammatiche ed automi, si includono propriamente in maniera gerarchica come in figura A.1 . Ad una grammatica si può

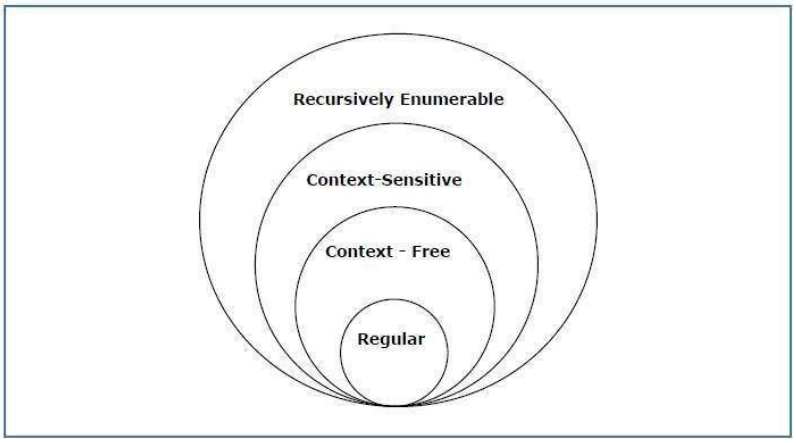


Figura A.1: Gerarchia di linguaggi secondo Chomsky

associare un unico linguaggio. Adesso si definirà come avvenie tale associazione:

Definizione. Il linguaggio generato da una grammatica $\mathcal{G}$ è l'insieme di tutte le stringhe che possono essere derivate a partire dal simbolo iniziale S:

$$L(\mathcal{G}) = \{x \in \Sigma^* : S \overset{\mathcal{G}}{\Rightarrow} x\}$$

E' rilevante notare che come detto una grammatica genera un unico linguaggio ma un linguaggio può essere generato da molteplici grammatiche.
Inoltre la tassonomia di Chomsky non è esaustiva di tutti i linguaggi possibili, infatti

esistono dei linguaggi che non sono ricorsivamente enumerabili cioè non c'è nessuna macchina di Turing che li riconosce.
Infine esistono altre classi di linguaggi che non sono incluse nella classificazione appena esposta e che saranno adoperate nel corso della trattazione:

Definizione (Linguaggio Ricorsivo)**.** Un linguaggio L è detto **ricorsivo** se è decidibile cioè esiste una macchina di Turing M che accetta ogni stringa in $x \in L$ e rigetta ogni stringa $x \notin L$

I linguaggi regolari e context-free sono ricorsivi. Esistono invece linguaggi context-sensitive non ricorsivi, cioè non ne sono un sottoinsieme [33, p. 124]. Tutti i linguaggi ricorsivi sono ricorsivamente enumerabili (cioè sono anche linguaggi semidecidibili, per i quali una stringa non appartenente al linguaggio può essere sia rigettata che andare in ciclo infinito rispetto a una macchina di Turing) ma non è sempre vero il viceversa.

Definizione (Linguaggio primitivo ricorsivo)**.** Un linguaggio è **primitivo ricorsivo** quando la sua funzione caratteristica è primitiva ricorsiva.

Il concetto di funzione caratteristica è spiegato nella sezione A.3. Invece una funzione primitiva ricorsiva è una funzione definita come una delle funzioni di base o combinando le funzioni di base con operazioni come composizione e ricorsione. Per una definizione formale si rimanda a [44]. I linguaggi primitivi ricorsivi sono un sottoinsieme dei linguaggi ricorsivi.

Definizione (Linguaggi superfiniti)**.** La classe dei linguaggi superfiniti è tale se contiene tutti i linguaggi finiti ed almeno un linguaggio infinito.

A.3 Automi a stati finiti

Nella sottosezione A.2.2 sono stati introdotti gli automi e una loro classificazione. Qui ci si concentrerà sullo studio dei *FSA* in relazione ai linguaggi regolari, la classe dei linguaggi a cui questa tesi è rivolta. Come visto gli *FSA* sono un caso speciale di *macchina di Turing* e più nello specifico un caso speciale di **Finite State Machine (*FSM*)** che in questa sede non interessa definire. Qui basterà dire che una *FSM* è un *transiction system* costituita da un insieme finito di stati dove ogni transizione è innescata da un'azione tra un insieme finito di azioni (di solito denotato da Σ). Esistono diversi tipi di *FSM* come le *Mealy Machines* e gli *FSA*. Tra gli *FSA* si annoverano gli **Non-Deterministic Finite Automata (*NFA*)** strettamente correlati ai *DFA* su cui si focalizzerà l'attenzione.

Definizione A.1 (Automa a stati finiti deterministico)**.** Un *DFA* A è una quintupla:

$$A = \left\langle \Sigma, Q^A, q_\epsilon^A, \delta^A, \mathbb{F}_{\mathbb{A}}^A \right\rangle$$

dove:
Σ è un alfabeto
Q^A è un insieme finito di stati
$q^A_\epsilon \in Q^A$ talvolta indicato come q^A_λ è lo stato iniziale
$\delta^A : Q^A \times \Sigma \to Q^A$ è la funzione di transizione
$\mathbb{F}^A_\mathbb{A} \subseteq Q^A$ è l'insieme degli stati accettanti

Inoltre nel corso della trattazione seguendo [17, p. 72] in alcuni casi è conveniente utilizzare anche un'altra definizione per i *DFA* uguale a quella appena data ma comprendente anche un nuovo insieme $\mathbb{F}^A_\mathbb{R} \subseteq Q^A$ che è l'insieme degli stati rigettanti. Quando si parlerà di *DFA* si farà sempre riferimento alla prima definizione,quella classica, a meno che non è specificato o l'utilizzo della seconda definizione risulta tacitamente evidente dall'utilizzo dell'insieme $\mathbb{F}^A_\mathbb{R}$. Si indica con $\|A\| = |Q^A|$. Inoltre in molti frangenti è conveniente utilizzare (questo è un discorso che esula dalla definizione di un *DFA*) una versione estesa della funzione di transizione a una stringa anzichè ad un solo simbolo dell'alfabeto. Si definisce allora $\hat{\delta}^A : Q^A \times \Sigma^* \to Q^A$ definendo induttivamente $\hat{\delta}^A(q, \epsilon) = q$ e $\hat{\delta}^A(q, aw) = \hat{\delta}^A(\delta^A(q, a), w)$ per $q \in Q^A$ e $aw \in \Sigma^+$ e $w \in \Sigma^*$. Per la funzione di transizione estesa nel corso della tesi sarà anche usata interscambiabilmente la definizione $A[w] = \hat{\delta}^A(q^A_\epsilon, w)$. Inoltre si estende quest'ultima notazione agli insiemi di stringhe: $W \subseteq \Sigma^* \mid A[W] = \{A[w] : w \in W\}$.

Una stringa x è detta essere accettata da un *DFA* A se e solo se $\hat{\delta}^A(q_\epsilon, x) = q'$ tale che $q' \in \mathbb{F}^A_\mathbb{A}$ che significa che usando la funzione di transizione estesa $\hat{\delta}^A$ a partire dallo stato iniziale è possibile arrivare ad uno stato accettante. Il linguaggio individuato da un *DFA* A è allora:

$$L(A) = \{x \in \Sigma^* : \hat{\delta}^A(q_\epsilon, x) \in \mathbb{F}^A_\mathbb{A}\}$$

Una relazione analoga a quanto visto tra linguaggi e grammatiche sussiste tra linguaggi e *DFA*: un *DFA* induce un solo linguaggio regolare, ma ad un linguaggio regolare corrispondono più *DFA*.
In molti contesti è utile rifersi alla *funzione di output* di un *DFA* A, λ^A come:

$$\lambda^A : \Sigma^* \to \mathbb{B}, \quad \forall w \in \Sigma^* \quad \lambda^A(w) = \begin{cases} 1 & \text{se } w \in L(A) \\ 0 & \text{altrimenti} \end{cases}$$

La *funzione di output* è la *funzione caratteristica* di L(A).Inoltre λ^A appena definita sopra può essere vista come un caso particolare, per $q = q^A_\epsilon$, di $\lambda^A_q(w)$ che assume valore 1 se $\hat{\delta}^A(q, w) \in \mathbb{F}^A_\mathbb{A}$. Ancora, due *DFA* ,A e A' , sono **equivalenti** denotato da $A \cong A'$, se loro hanno la stessa funzione di output, cioè se $\lambda^A = \lambda^{A'}$, cioè se indivividuano lo stesso linguaggio. Il concetto di equivalenza è rilevante anche tra gli stati dello stesso *DFA* ed informalmente due stati sono equivalenti se non esiste nessuna stringa che li distingue, cioè che partendo da quei due stati porta a stati di arrivo che sono uno accettante e l'altro no.

Definizione (Stati equivalenti)**.** Detto A essere un *DFA*, e q e p $\in Q^A$ stati di A. q e p sono *equivalenti* ,denotato da $q \equiv p$, se $\lambda^A_q = \lambda^A_p$

Una stretta correlazione tra l'equivalenza di stati e l'equivalenza di *DFA* è il seguente risultato: $A \cong A' \Leftrightarrow q_\epsilon^A \equiv q_\epsilon^{A'}$. Un altro concetto importante è quello di *isomorfismo* tra due DFA.

Definizione (Isomorfismo di *DFA*)**.** Detti A ed A$'$ due *DFA* definiti su Σ. A ed A$'$ sono detti isomorfici se esiste un isomorfismo $f : Q^A \to Q^{A'}$, cioè una funzione soddisfacente le seguenti condizioni:

1. $f(q_\epsilon^A) = q_\epsilon^{A'}$
2. $\forall q \in Q^A : q \in F_{\mathbb{A}}^A \Leftrightarrow f(q) \in F_{\mathbb{A}}^{A'}$
3. $\forall q \in Q^A, a \in \Sigma : f(\delta^A(q, a)) = \delta^{A'}(f(q), a)$

L'isomorfismo è un requisito più forte dell'equivalenza: *DFA* isomorfi sono pure equivalenti, ma in generale non è vero il contrario. Quindi dato un linguaggio L esistono più *DFA* in grado di riconoscerlo cioè con la stessa *funzione di output* λ. Tra questi di particolare interesse sono quelli con il minor numero di stati. Un *DFA* A è detto **minimo** se qualunque altro *DFA* A$'$ tale che $A \cong A'$ (con la stessa *funzione di output*) soddisfa $|Q^{A'}| \geq |Q^A|$. Ovviamente in un *DFA* minimo nè stati irragiungibili ne stati equivalenti possono essere presenti perchè potrebbero essere eliminati senza cambiare la funzione di output λ. Inoltre il DFA minimo è sempre unico a meno di una possibile rinomina degli stati. Per motivi storici esiste anche la definizione di *DFA* **canonico** che è intercambiabile con quella di *DFA* minimo ma entrambe indicano lo stesso ente matematico e sono del tutto equivalenti. Formalizzando

Definizione (DFA minimo/canonico)**.** Detto A essere un *DFA* su Σ. A è detto canonico se le seguenti condizioni sono verificate:

1. Tutti gli stati sono raggiungibili: $A[\Sigma^*] = Q^A$
2. Tutti gli stati sono a coppie separabili[1]: $\forall q \neq p \in Q^A : \exists w \in \Sigma^* : \lambda_q^A(w) \neq \lambda_p^A(w)$

Per ogni DFA esiste sempre ,ed è unico (a meno delle etichette degli stati) un *DFA* equivalente che è canonico.

A.3.1 FSA particolari

Alcuni *DFA* ed *NFA* sono particolarmente significativi e ricorreranno spesso nell'ambito di questa tesi. Inoltre è necessario conoscere per il proseguio della trattazione qual è la differenza principale tra *DFA* ed *NFA*. Un *NFA* è detto non deterministico perchè la sua funzione di transizione può avere più di una transizione per un dato simbolo dell'alfabeto (ed inoltre vi possono essere transizioni anche in corrispondenza di ϵ). Inoltre in un *NFA* non vi è necessariamente per ogni stato una transizione in corrispondenza di ogni simbolo dell'alfabeto.

[1] due stati sono separabili o distinti se non sono equivalenti

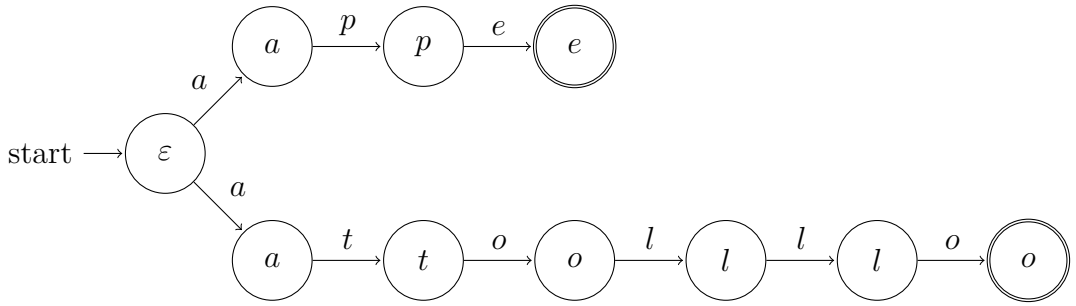


Figura A.2: *MCA(I_+)* per $I_+ = \{\text{ape,atollo}\}$

Maximal Canonical Automaton

Definizione A.2 (Maximal Canonical Automaton)**.** Detto $I_+ = \{x_1, \cdots, x_N\}$ un insieme di esempi positivi, si definisce ***Maximal Canonical Automaton rispetto ad*** I_+ e si denota con ***MCA(I_+)*** un *NFA* costituito da una quintupla $\langle \Sigma, Q, q_\epsilon, \delta, \mathbb{F}_\mathbb{A} \rangle$ dove:

Σ è l'alfabeto su cui è definito I_+
$Q = \{q_u^i : u \in \text{Pref}(x_i) \wedge u \neq \epsilon\} \cup \{q_\epsilon\}, 1 \leq i \leq N$
$q_\epsilon = \{q_\epsilon\}$
$\delta(q_u^i, a) = \{q_{ua}^i : ua \in \text{Pref}(x_i)\}, \forall a \in \Sigma, 1 \leq i \leq N$
$\delta(q_\epsilon, a) = \{q_a^i : a \in \text{Pref}(x_i)\}, \forall a \in \Sigma, 1 \leq i \leq N$
$1 \leq i \leq N, q_{x_i}^i \in \mathbb{F}_\mathbb{A}$
se $\epsilon \in I_+$ aggiungere q_ϵ ad $\mathbb{F}_\mathbb{A}$

Un *MCA(I_+)* per ogni stringa di I_+ ha un percorso dedicato a partire dallo stato iniziale. Si noti che nella definizione data di δ accade che per molti simboli di Σ non c'è la corrispondente transizione per un dato stato. Inoltre se in I_+ sono presenti stringhe che hanno lo stesso simbolo iniziale si avrà indeterminismo sullo stato iniziale q_ϵ. Quindi in generale il *MCA(I_+)* è un *NFA*.Un esempio è dato in figura A.2

Automa quoziente

Sia A un *DFA* su Σ e sia $\approx \subseteq Q^A \times Q^A$ una relazione d'equivalenza sull'insieme Q^A soddisfacente le seguenti due condizioni:

(i) $\approx$ satura $\mathbb{F}_\mathbb{A}^A$

(ii) $\forall q, p \in Q^A : q \approx p \Rightarrow (\forall a \in \Sigma : \delta^A(q, a) \approx \delta^A(p, a))$

Allora da A tramite $\approx$ è possibile ricavare il ***DFA quoziente*** $A/\approx$ così definito:

1. Σ è lo stesso di A

2. $Q^{A/\approx} = Q^A/\approx$
3. $q_\epsilon^{A/\approx} = [q_\epsilon^A]_\approx$
4. $\mathbb{F}_\mathbb{A}^{A/\approx} = \{[q]_\approx : q \in \mathbb{F}_\mathbb{A}^A\}$
5. $\delta^{A/\approx}([q]_\approx, a) = [\delta^A(q,a)]_\approx \quad \forall q \in Q^A, a \in \Sigma$

L'automa quoziente $A/\equiv$,dove la relazione d'equivalenza utilizzata è quella di equivalenza tra gli stati, corrisponde al *DFA* canonico. Quindi banalizzando si può concludere dicendo che l'automa quoziente di un *DFA* è ciò che si ottiene fondendo insieme alcuni stati del *DFA* di partenza in base a una relazione d'equivalenza. Quando la relazione usata è quella di stati equivalenti si ottiene il DFA minimo: quindi $Q^{A/\approx}$ sarà una partizione in cui in ogni blocco (sottoinsieme) ci saranno stati equivalenti tra loro (in uno specifico blocco). Ogni blocco è una classe d'equivalenza.

Prefix Tree Acceptor

Si è visto che l'automa quoziente che si ottiene usando la relazione di equivalenza degli stati su un *DFA* A è il *DFA* minimo. Analogamente è possibile ottenere il ***Prefix Tree Acceptor di*** I_+ indicato con $\boldsymbol{PTA}(I_+)$ applicando la definizione di automa quoziente al $MCA(I_+)$[2] con la relazione: *stati che identificano lo stesso prefisso*. Quindi verrà effettuata la fusione degli stati che condividono lo stesso prefisso. Si definisce la relazione d'equivalenza come:

$$p \approx q \Leftrightarrow \text{ Prefix}(p) = \text{ Prefix}(q)$$

allora:

$$MCA(I_+)/\approx = PTA(I_+)$$

In figura A.3 il *PTA* ricavato a partite dal *MCA* di figura A.2 .

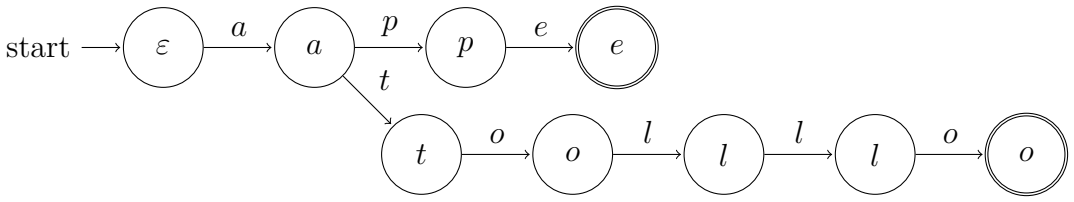


Figura A.3: *PTA*(I_+) per $I_+ = \{$ape,atollo$\}$

Automa Universale

Si indica con ***UA l'automa universale*** che accetta tutte le stringhe definite su Σ. Si ha $L(UA) = \Sigma^*$. Ha un unico stato,che è accettante, con un *self-loop* per ogni simbolo dell'alfabeto.

[2]Anche se tecnicamente può essere un *NFA* la definizione di automa quoziente può essere applicata comunque

A.3.2 Funzioni di output regolari

In una sottosezione di A.3.1 si è visto che è possibile ottenere il *DFA* canonico minimo di un *DFA* A tramite $A/\equiv$ (l'automa quoziente sulla relazione di stati equivalenti). In questa sezione invece si vedrà come ottenere il *DFA* canonico minimo non a partire da un *DFA* preesistente, ma semplicemente sfruttando le proprietà di una funzione di output $\lambda : \Sigma^* \to \mathbb{B}$ che è la funzione caratteristica di qualche linguaggio regolare.

Relazione di Nerode

Si possono caratterizzare le *funzioni di output regolare* come la classe di funzioni $\lambda : \Sigma^* \to \mathbb{B}$ per cui un *DFA* con quella *funzione di output* esiste[3]. Il famoso teorema *Myhill-Nerode* [37] fornisce una caratterizzazione alternative delle *funzioni di output regolari* , che non fa affidamento sulla nozione di *DFA*. Come primo step si definisce la *relazione di Nerode* [37] sulle stringhe che definisce un'equivalenza sulle stringhe secondo λ:

Definizione A.3 (Relazione di Nerode)**.** Sia $\lambda : \Sigma^* \to \mathbb{B}$ una funzione di output a due valori arbitraria definita su Σ. Due stringhe $u, u' \in \Sigma^*$ sono equivalenti secondo $\simeq_\lambda$[4] denotato da $u \simeq_\lambda u'$ se e solo se:

$$\forall v \in \Sigma^* \quad \lambda(uv) = \lambda(u'v)$$

dove $\simeq_\lambda \subseteq \Sigma^* \times \Sigma^*$ è una relazione binaria detta *relazione di Nerode* o *congruenza di Nerode* che definisce l'equivalenza tra stringhe secondo λ

Teorema Myhill-Nerode

La *relazione di Nerode* $\simeq_\lambda$ può essere vista come l'equivalente a livello di stringhe della relazione di equivalenza $\equiv_A \subseteq Q^A \times Q^A$, sugli stati di un *DFA* A. Dovrebbe essere osservato, tuttavia , che $\simeq_\lambda$ può essere definito per *funzioni di output* arbitrarie, non solo regolari. Il teorema *Myhill-Nerode* fornisce una caratterizzazione delle *funzioni di output regolari* basata su $\simeq_\lambda$:

Teorema A.1 (Teorema Myhill-Nerode o di caratterizzazione)**.** *Sia* $\lambda : \Sigma^* \to \mathbb{B}$ *una funzione di output a due valori.* λ *è regolare se e solo se la relazione di Nerode* $\simeq_\lambda$ *ha indice finito.*

Una dimostrazione del teorema si trova in [49]. L'implicazione in uno dei due versi del teorema dice che se $\simeq_\lambda$ ha indice finito, allora λ è *una funzione di output regolare* cioè esiste un *DFA* A con $\lambda^A = \lambda$. Questo *DFA* $A = \left\langle \Sigma, Q^A, q_\epsilon^A, \delta^A, F_{\mathbb{A}}^A \right\rangle$ è definito come:

- Σ è il dominio di λ

[3] L'insieme delle funzione di output regolari identifica la classe dei linguaggi regolari

[4] Si noti che per denotare l'equivalenza non si è usato il simbolo $\equiv$ (equivalenza tra stati) perchè qui si parla di equivalenza tra stringhe

- $Q^A = \Sigma^*/\simeq_\lambda$
- $q_\epsilon^A = [\epsilon]_{\simeq_\lambda}$
- $F_{\mathbb{A}}^A = \{[u]_{\simeq_\lambda} |\ \lambda(u) = 1\}$
- $\delta^A([u]_{\simeq_\lambda}, a) = [u{\cdot}a]_{\simeq_\lambda}$

A è il *DFA* minimo corrispondente al linguaggio identificato da λ.Si osservi come la costruzione del *DFA* A è molto simile alla costruzione del *DFA* minimo usando la relazione di equivalenza sugli stati $\equiv$ usando l'automa quoziente a partire da un *DFA*. Questo approccio alla costruzione degli automi è fondamentale nell'*active learning*.

Appendice B

Preliminari e implementazione dell'Observation Pack

Qui si presentano delle ulteriori notazioni inerenti prevalentemente il capitolo 3 e vengono svelati alcuni dettagli implementativi dell'*ObP*.

B.1 Notazione specifica per l'ObP

In questo sezione vi è la delineazione di un'ulteriore notazione utilizzata principalmente nel capitolo 3 in congiunzione a quella introdotta in A, ma che essendo specifica dell'*ObP* viene presentata in quest'appendice. Inoltre vi è anche la presentazione della notazione utilizzata per presentare un framework introdotto in [50] che facilità la compresione della correttezza e l'implentazione del metodo di gestione del controesempio in *ObP*.

B.1.1 Definizioni

L'*ObP* mantiene un insieme Sp,*prefix-closed*, di **short prefix** detti anche **access sequence** ,che sono prefissi . Ogni short prefix identifica unicamente (cioè short prefix diversi identificano stati diversi in H e nel target) gli stati sia nel target A che nell'ipotesi H. Ogni stato $q \in Q^H$ corrisponde unicamente ad una stringa (lo short prefix) $u \in Sp$, ed è assicurato che $H[u] = q$. u è detta l'access sequence di q (in H), ed è denotata da $\lfloor q \rfloor_H$. Alternativamente quanto detto può essere formulato come $\forall q \in Q^H : \ H[\lfloor q \rfloor_H] = q$. Lo stato iniziale q_ϵ^H è lo stato con access sequence ϵ.
Si estende questa notazione a stringhe arbitrarie $w \in \Sigma^* : \lfloor w \rfloor_H = \lfloor H[w] \rfloor_H$ che significa che w raggiunge uno stato in H e questo stato ha un access sequence u cui w si associa. Quindi la funzione $\lfloor \cdot \rfloor_H : \Sigma^* \to Sp$ trasforma stringhe in access sequences.
Uno short prefix $u \in Sp$ corrisponde ad uno stato in A, cioè $A[u]$. Ci si riferisce ad $A[Sp]$ come gli stati scoperti (dal *learner*) di A. Gli short prefixes quindi stabiliscono una funzione f_{Sp} che collega stati nell'ipotesi e stati scoperti nel *DFA* target A come

segue:

$$f_{Sp} : Q^H \to Q^A, f_{Sp}(q) = A[\lfloor q \rfloor_H]$$

B.1.2 Definizioni per il framework

Prefix Transformation

Prefix transformation è una procedura che consente di trasformare un prefisso di un controesempio $w \in \Sigma^+$ in un access sequence in Sp.

Definizione (Prefix Transformation)**.** Prefix transformation rispetto ad H, π_H , è definita come segue:

$$\pi_H : \Sigma^* \times \mathbb{N} \to \Sigma^* \ , \ \pi_H(w, i) = \lfloor w_{[0,i)} \rfloor_H \cdot w_{[i,|w|)}$$

Si osservi che, $\pi_H(w, 0) = w$ e $\pi_H(w, |w|) = \lfloor w \rfloor_H \in Sp$.

Altre definizioni

Sia $w \in \Sigma^{+1}$ un controesempio che differenzia il target A da H. Sia $m = |w|$ ed i un indice $0 \leq i \leq m$ allora si definisce la funzione α come:

$$\alpha : [0, m+1) \to \mathbb{B},\ \alpha(i) = \begin{cases} 1 & \text{se } \lambda^A(\pi_H(w, i)) = \lambda^H(w) \\ 0 & \text{altrimenti} \end{cases}$$

Dalla funzione α può essere ricavato anche la definizione della funzione β:

$$\beta : [0, m) \to \{0, 1, 2\},\ \beta(i) = \alpha(i) + \alpha(i+1)$$

B.2 Dettagli implementativi dell' ObP

Vengono ora riportati alcuni dettagli implementativi ritenuti più significativi senza pretesa di esaustività. Per la memorizzazione della funzione OT (definizione 3.2.1) di un componente si è utilizzata una map che tipicamente è nella forma chiave-valore dove nella fattispecie la chiave è una stringa (ottenuta dalla concatenazione di un prefisso con un suffisso) ed il valore è l'esito delle *MQ* nel target per quella stringa. Qui si è utilizzato per il valore un doppio campo, oltre all'esito della *MQ* c'è un contatore che rappresena il numero di volte che quella stringa viene a formarsi in un dato componente (una stessa stringa può essere formata da coppie prefisso-suffisso diverse). Questo è dovuto al fatto che quando si effettua uno split di un componente alcuni dei prefissi del componente migrano nei prefissi del nuovo componente. Oltre all'insieme dei prefissi del componente splittato (che va decrementato) si può modificare anche il dominio della funzione OT restringendolo. Nel codice si è scelto di

[1] In *ObP* ϵ non può essere un controesempio

eliminare la concatenazione del prefisso che migra nel nuovo componente con l'insieme di suffissi del componente splittato, restringendo il dominio di OT. Tuttavia per garantire la correttezza è necessario appurare se quella concatenazione di quel prefisso con un dato suffisso si venga a formare anche tramite altri prefissi perchè in questo caso l'eliminazione non deve avvenire ed è per questo motivo che si usa un contatore per ogni stringa (per ogni concatenazione) che va incrementato ad ogni inserimento di una stringa (anche una già esistente) e decrementato nel caso suddetto. Questa scelta è stata fatta nell'ottica di consentire velocemente di appurare se un componente è chiuso e in generale consentire una ricerca molto veloce di un prefisso in un componente. L'alternativa sarebbe stata quella di non utilizzare il contatore e non eliminare mai la stringa formata dalla concatenazione di un prefisso e di un suffisso ma solo il prefisso dal componente splittato. Ciò porterebbe ad una crescita del dominio di ricerca per la funzione OT che degraderebbe le prestazioni della ricerca di un prefisso in un componente (operazione effettuata sovente) anche se potrebbe comportare una diminuizione del numero delle *MQ* e l'eliminazione dell'*overhead* per tenere aggiornato il contatore: questa prospettata diminuizione delle *MQ* con questa seconda scelta è però possibile soltanto se quando si completano le componenti, ad esempio nella funzione UPDATE-FROM-COUNTEREXAMPLE prima di effettuare una *MQ* su una stringa x si controlli se per x non si conosca già l'esito della *MQ* perchè contenuto già nel dominio della funzione OT[2] anche se nel caso vi fossero molti *miss* questa politica potrebbe essere addirittura controproducente. Anche quest'ulteriore strategia non è stata implementata ritenendo poco probabile un *hit* (per implementarla basta decommentare un if in UPDATE-FROM-COUNTEREXAMPLE e aggiungerne un altro nella funzione *sift*). Se si fossero fatte delle scelte opposte a quelle fatte e appena descritte senz altro si sarebbe potuto abbassare ulteriormente il numero di *MQ* ma si è ritenuto che il gioco non vale la candela cioè che il prezzo da pagare in termini di tempo di esecuzione per ottenere ciò è maggiore del beneficio ottenuto.
Si sottolinea che diversamente dal metodo OBP-SPLIT (algoritmo 13) non viene ritornato il nuovo componente ottenuto perchè il chiamante OBP-CLOSEPACK ne è già a conoscenza , quindi il metodo non ritorna niente.
Infine si prende in esame l'implementazione del discrimination tree. Per quest ultimo si è scelto di usare un *vector* di nodi e un insieme di archi. Per ogni nodo si memorizza l'etichetta e se è accettante o meno. Gli archi sono un *vector* di array bidimensionali. L' indice di un nodo nel *vector* di nodi è usato per accedere alla posizione nel *vector* di archi che contiene gli indici dei nodi figli (nell'array contenuto nel *vector* di nodi nella posizione individuata dall'indice del nodo). L'utilizzo di un insieme di nodi e di archi è tipico di un grafo piuttosto che di un albero binario. Si è scelta ugualmente questa implementazione essenzialmente per due ragioni:

- La *Standard Template Library* non mette a disposizione nessuna struttura dati per modellare una albero binario. Esistono delle librerie esterne che mettono a disposizioni un albero n-ario ma l'overhead per gestire n figli ed altre

[2]più è grande il dominio e maggiore sarà la probabilità di ottenere un *hit* per x

operazioni inutili ai fini dell'*ObP* ha fatto propendere per il declinare il loro utilizzo. Un'altra possibilità sarebbe stata quella d'implementare un albero binario *ad hoc* nella maniera classica cioè tramite i puntatori ma essendo le prestazioni simili a quelle della soluzione adottata e descritta sopra non lo si è fatto. Inoltre si è supposto che chiamare un oggetto di una classe esterna (quella dell'eventuale implementazione dell'albero binario), dato che va fatto molte volte, sarebbe divenuto il costo preponderante. Il vantaggio principale nell'usare l'implementazione di un albero binario con i puntatori per il discrimination tree sta nel risparmio di memoria circa doppia ma comunque sempre lineare nella soluzione proposta, e che non avviene mai la riallocazione (operazione che accade quando le dimensioni del vettore superano la capacità dello stesso).

- Utilizzare un vettore indicizzato. Questa soluzione è da scartare perchè se il discrimination tree non è bilanciato e presenta un ramo molto più lungo degli altri sarebbe necessario rendere il vettore molto grande. Il vettore può crescere dinamicamente ed è molto più probabile con un vettore indicizzato superare la capacità totale del vettore (se l'inserimento del nodo avviene nello stesso ramo ciò avviene molto velocemente) che verrebbe quindi riallocato e ricopiato frequentemente degradando le prestazioni.

Appendice C

SVM

La tesi è volta ad investigare lo scenario in cui si utilizzi un classificatore come base per costruire un Oracolo. Da quest esigenza nasce la necessità di utilizzare uno dei tradizionali metodi statistici di *machine learning*. Nella fattispecie si deve risolvere un problema di **classificazione binaria** dato che il modello, una volta avvenuto l'addestramento, ha lo scopo di predire come accettante o rigettante , cioè appartenente o meno ad $\mathcal{L}$, il campione sotto esame.
Tra i moltissimi metodi statistici esistenti forse i più noti sono *Random Forest, Recurrent Neural Network, Convolutional Neural Network* e *SVM*. Tra questi un metodo che è stato molto utilizzato nell'apprendimento di linguaggi è una riadattazione delle *Reti Neurali Ricorrenti*, che si sono dimostrate adeguate allo scopo. In queste sede si è scelto di usare le *SVM* perchè rappresentano un classificatore molto potente, e sebbene esistano dei lavori inerenti il tema [30] non sono stati contestualizzati nell'ambito dell'*active learning* per *IIR* oppure sono limitati all'apprendimento di alcune specifiche classi di linguaggi [29] [12] [11].

C.1 Overview teorica

Il problema che si tenta di risolvere con le *SVM* , che è la radice da cui nascono anche altri modelli , appare sotto diversi nomi come il compromesso bias-varianza, controllo della capacità e dell'*overfitting* dei dati, ma l'idea è la stessa: dato un problema di **apprendimento supervisionato**, cioè con un insieme di addestramento di cardinalità finita che contiene i dati etichettati con la rispettiva classe di appartenenza (il *training set*) , le migliori *performances* di generalizzazione[1] del classificatore saranno ottenute dal raggiungimento del giusto equilibrio tra l'*accuracy* sul training set del classificatore e la capacità del modello. La capacità di un modello di classificazione statistico rappresenta la complessità, la potenza espressiva, la ricchezza piuttosto che la flessibilità della famiglia di funzioni $f(x, \alpha)$:

$$f : X \to Y \quad \text{con } X = \{x_1, x_2, \ldots, x_l\}$$

[1] Cioè una migliore capacità di predire correttamente la classe di appartenenza di dati mai visti, cioè non contenuti nel *training set*. Per questo motivo si parla di classificazione.

X contiene i campioni e rappresenta il *training set* e Y rappresenta i possibili valori delle etichette di ciascun campione e nel caso di **classificazione binaria** $Y = \{0, 1\}$. Inoltre si parla di famiglia di funzioni perchè *SVM* (come gli altri modelli) ha una serie di iperparametri ,rappresentati da α , che identificano i diversi classificatori[2]. Una capacità alta implica che le funzioni $f(x, \alpha)$ sono complesse cioè di alto grado ad esempio: il discorso è analogo a ciò che accade in una feedback-forward con tante unità nascoste. Per il principio del *Rasoio di Occam* (sezione 2.2) è bene selezionare il classificatore con la capacità minima che contemporaneamente si adatta ai dati del *training set* ma come si vedrà i due aspetti sono in contrapposizione e sarà necessario trovarne il giusto compromesso. Questo assunto più avanti sarà approfondito e giustificato. Per il momento basti dire che un classificatore con troppa capacità è come un botanista con una memoria fotografica che quando vede un albero mai visto — cioè appartenente a quello che formalmente viene identificato come *test set* — conclude che non è un albero perchè ha un numero di foglie diverso dagli alberi visti finora: situazione detta di *overfitting* in cui il classificatore si è adattato esclusivamente ai dati del *training set* ma non generalizza (analogo della rete neurale con troppe unità nascoste rispetto alla cardinalità del *training set*). D'altro canto una macchina con bassa capacità è come il fratello pigro del botanista che classifica qualunque cosa sia verde come un albero. In nessuno dei due casi si avrà una buona generalizzazione.

C.1.1 Rischio Atteso

Supponiamo di avere l osservazioni nel *training set*. Ogni osservazione è una coppia: un vettore $x_i \in \mathbb{R}^n$, $i = 1, \cdots, l$ e l'etichetta corretta associata y_i. Per semplicità $y_i \in \{-1, 1\}$. Si assume che le coppie (x_i, y_i) siano estratte da una distribuzione di probabilità cumulativa (CDF) che si indica con $P(x, y)$, e con $p(x, y)$ la corrispondente densità di probabilità (PDF). Supponiamo che il modello sta provando ad imparare il mapping $x_i \to y_i$ per $i = 1, \ldots, l$. Il modello è definito da una serie di possibili mapping $f(x, \alpha)$ tali che $x \to f(x, \alpha)$. Il modello rappresenta la classe di funzioni $f(x, \alpha)$, si parla di classe perchè più funzioni possono essere imparate dal modello cambiando gli *iperparametri* α. Fissare gli α (che possono essere più di uno) per un modello si concretizza in un modello "trained", cioè in uno specifico classificatore. Ad esempio se il modello è una rete neurale gli *iperparametri* α sono tipicamente i pesi e il bias.
Seguendo il riferimento [54], si può definire il **rischio atteso** per un classificatore (α fissato) come:

$$R(\alpha) = \int \frac{1}{2}|y - f(x, \alpha)| \, dP(x, y) = \int \frac{1}{2}|y - f(x, \alpha)| \, p(x, y) \, dx \, dy \tag{C.1}$$

[2]Classificatore e modello non sono termini interscambiabili. Il modello è rappresentato dalle funzioni $f(x, \alpha)$ con α generico ,un classificatore invece è una singola funzione che si ottiene dal modello per una specifica istanza degli *iperparametri* α.

Il rischio atteso è anche detto **true mean error** dato che è calcolato su tutti i possibili valori di x e y $(X \times Y)$ cioè tiene conto di tutte le combinazioni sia sulle coppie nel *training set* sia di tutte quelle mai osservate. Inoltre nell'equazione (C.1) $1/2|y-f(x,\alpha)|$ è una **funzione di loss** , ma se ne poteva scegliere un'altra infatti in generale si ha $V(f(x,\alpha),y)$. La specifica funzione di loss scelta nell'equazione (C.1) può assumere solo valori in $\mathbb{B}$. Tuttavia $R(\alpha)$ così definito non è utile in quanto quasi mai si conosce $P(x,y)$ o una sua stima ,allora si definisce il **rischio empirico** per un classificatore:

$$R_{emp}(\alpha) = \frac{1}{2l}\sum_{i=1}^{l}|y - f(x_i,\alpha)|$$

e rappresenta il tasso di errore medio sul *training set*. $R(\alpha) - R_{emp}(\alpha)$ è l'errore di generalizzazione. Si dice che un modello generalizza se

$$\lim_{l\to\infty} R(\alpha) - R_{emp}(\alpha) = 0$$

cioè al crescere degli elementi nel *training set* gli elementi mal predetti nel *test set* diminuiscono. Ma come già detto il rischio atteso non è quasi mai calcolabile per come è stato definito, allora con un'impostazione molto simile al *PAC-learning* introdotto in 2.1 Vapnik ha dimostrato in [54] che:

Definiti

$$\eta : 0 \leq \eta \leq 1$$
$$h : h > 0 \tag{C.2}$$
$$\varepsilon = \sqrt{\left(\frac{h(\log(2l/h)+1) - \log(\eta/4)}{l}\right)} \tag{C.3}$$

allora si ha che:

$$R(\alpha) \leq R_{emp}(\alpha) + \varepsilon \quad \text{con probabilità } 1-\eta \tag{C.4}$$

dove h è detta **Vapnik Chervonenkis (*VC*) dimension** e ε è detto **learning rate** o ***VC confidence***. La *VC dimension* è una misura della capacità e sarà approfondita nella sottosezione C.1.2. $R_{emp}(\alpha) + \varepsilon$ è talvolta detto ***risk bound*** dato che è un limite superiore del rischio atteso $R(\alpha)$. Il risultato dell'equazione (C.4) ci consente di trovare con probabilità $1-\eta$ il classificatore che minimizza $R_{emp}(\alpha)+\varepsilon$. Tuttavia calcolare questo minimo è molto difficile come è spiegato nella sottosezione C.1.3. Prima di parlarne si spiega cosa è la *VC dimension*.

C.1.2 *VC dimension*

La *VC dimension* è una misura della capacità di un insieme di funzioni $\{f(x,\alpha)\}$ che possono essere apprese da un modello statistico, ed è definita come la cardinalità del

più largo insieme di punti che un modello può ***shatter***. Per *shatter* si intende che una delle funzioni è in grado di etichettare correttamente i punti o meglio di separare punti (che sarebbero dei campioni, delle osservature) con etichettature diverse. Ci si pone nel caso della classificazione binaria e si fissa il codominio di $\{f(x,\alpha)\}$ in $\{-1,1\}\forall x,\alpha$. Se un insieme X di l punti viene etichettato in tutti i 2^l possibili modi, e per ognuna delle 2^l combinazioni, può essere trovata una funzione (con α specifico cioè un classificatore) dell'insieme $\{f(x,\alpha)\}$ che separa (o assegna) correttamente queste etichette, si dice che l'insieme di punti X è "*shattered*" dall'insieme di funzioni. Quindi la *VC dimension* dell'insieme di funzioni $\{f(x,\alpha)\}$ è definita come il **massimo** numero di punti che possono essere "*shattered*" da $\{f(x,\alpha)\}$. Si osservi che avere per esempio una *VC dimension* pari a tre significa che esiste almeno un insieme fatto da tre punti che può essere "*shattered*", ma non è detto (tipicamente non lo è) che tutti gli insiemi di tre punti lo siano[3]. Si fornisce come esempio il calcolo delle *VC dimension* dell'insieme costituito da tutte le rette in $\mathbb{R}^2$ identificate dall'equazione $y = mx + q$ in cui i parametri α sono rappresentati da m e q. I punti possono essere etichettati positivamente o negativamente, quindi per essere "*shattered*" da una retta devono essere separati da quest'ultima in base all'etichettatura. Inoltre si rimarca che i punti possono essere collocati in qualunque maniera, ma una volta collocati le varie etichettature devono essere effettuate mantenendo fissa quella collocazione scelta. Con tre punti, si riesce a trovare una collocazione dei tre punti (qualsiasi tranne tre punti allineati) tale che esiste una retta (valori specifici di m e q) che separari i punti positivi da quelli negativi e si riesce a fare ciò per ciascuna delle $2^3 = 8$ etichettature diverse[4] come si può apprezzare in figura C.1.

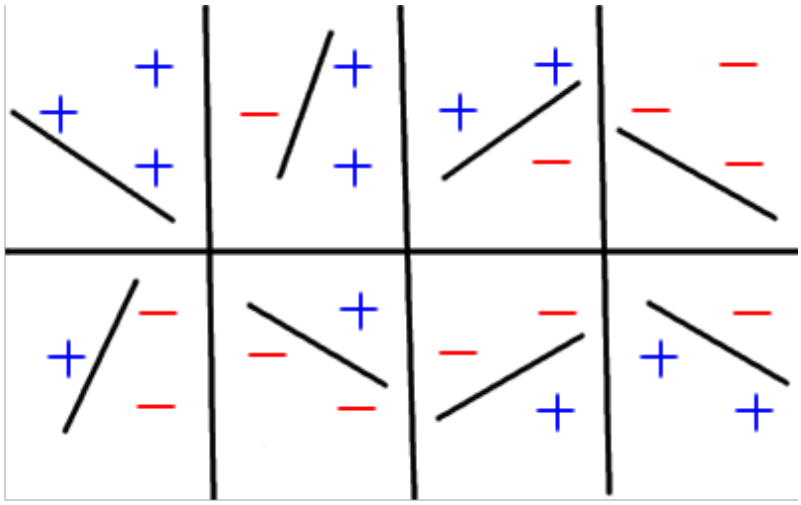


Figura C.1: Tre punti in $\mathbb{R}^2$, shattered da rette

[3]In questo caso si può solo concludere che non esiste nessun insieme di quattro punti che è "*shattered*"

[4]Si noti che in queste 8 etichettature la collocazione dei 3 punti deve restare costante ma si può scegliere un'altra funzione (retta) da un'etichettatura all'altra

Quattro punti non possono essere "*shattered*" quindi la *VC dimension* è tre. Più in generale la *VC dimension* dell'insieme di iperpiani in $\mathbb{R}^n$ è $n+1$. Una dimostrazione di questo risultato si può trovare in [8, p. 37].

C.1.3 Structural Risk Minimization

La *VC dimension* di un modello riveste un ruolo molto importante perchè come si evince dai risultati dell'equazione (C.4) ,fissati η e l, si ha che la *VC confidence* ε aumenta all'aumentare della *VC dimension* h. Quindi per diminuire il rischio atteso $R(\alpha)$ sembra sufficiente diminuire h ma in realtà non è così. Innanzitutto si precisa che un numero di parametri (α) maggiore nel modello non implica una *VC dimension* maggiore infatti esistono dei modelli con un solo parametro e *VC dimension* infinita. Inoltre possedere un valore di h più piccolo non comporta necessariamente di avere un rischio atteso più piccolo e quindi un classificatore migliore infatti un valore di h più piccolo significa che è stato ristretto l'insieme delle funzioni $\{f(x,\alpha)\}$ e può accadere che è stata eliminata proprio la funzione che minimizzava il rischio empirico. Quindi dinimuendo h la *VC confidence* ε diminuisce ma il rischio empirico può aumentare e quindi diminuire h non implica che il rischio atteso $R(\alpha)$ diminuisca[5]. Il motivo di questo comportamento è che la *VC confidence* dipende dalla classe di funzioni $\{f(x,\alpha)\}$ invece il rischio empirico dipende dalla scelta degli *iperparametri* e quindi da una specifica funzione.
Al fine di minimizzare $R(\alpha)$ si deve trovare il giusto compromesso tra due quantità che manifestano un andamento opposto al variare di h: il rischio empirico (che diminuisce all' aumentare della complessità di $\{f(x,\alpha)\}$ cioè della capacità del modello cioè all'aumentare di h) e la *VC confidence* (che diminuisce al diminuire di h). Vapnik in [53] ha proposto una procedura per affrontare il problema appena delineato detta **Structural Risk Minimization (*SRM*)** che consiste nel:

1. Dividere la famiglia di funzioni $\{f(x,\alpha)\}$ in sottoinsiemi autoincludenti come in figura C.3 in modo che i sottoinsiemi abbiano una *VC dimension* (che è un valore intero) crescente. Ciò significa escludere alcune funzioni da $\{f(x,\alpha)\}$ in modo da trovare la classe di funzioni h_1 con *VC dimension* più piccola e così via per h_2,$h_3\ldots$

2. Per ogni classe di funzioni trovate al punto 1. (h_1,$h_2\ldots$) trovare quella con il rischio empirico minimo: cioè equivale a una selezione dei parametri. Ad esempio per la classe h_1 significa trovare la funzione contenuta in $\{h_1(x,\alpha)\}$ che minimizza il rischio empirico (ciè equivale a trovare i parametri α che fanno ciò).

3. Selezionare il classificatore tra quelli trovati al punto 2. che minimizza $R(\alpha)$

SRM garantisce un compromesso tra la qualità dell'approssimazione dei dati nel *training set* e la complessità della funzione approssimante come illustrato in figura C.2

[5]Esistono modelli che hanno buoni riscontri pratici nonostante abbiano $h=\infty$

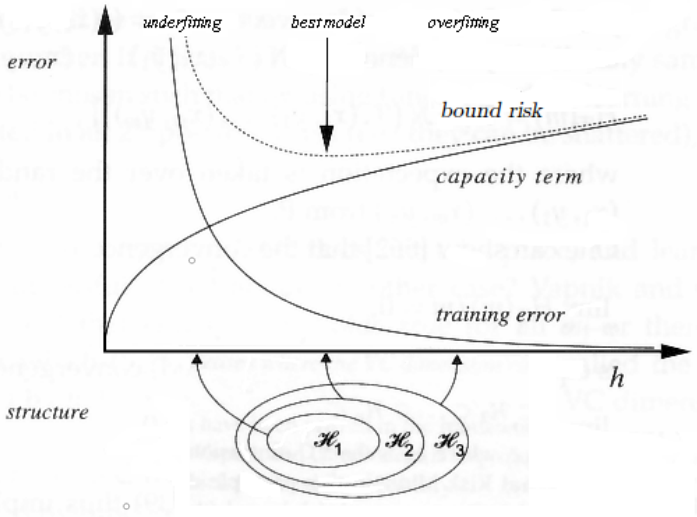


Figura C.2: Il *risk bound* è la somma del rischio empirico e la *VC confidence*. Il più piccolo *risk bound* è raggiunto su qualche elemento della struttura.

SRM non è quasi mai applicabile perchè è molto complicato riuscire a calcolare la *VC dimension* delle "sottofunzioni" e anche se ciò fosse possibile la difficoltà computazionale per calcolare il rischio empirico , a causa della spesso grande dimensionalità dello spazio degli *iperparametri* del modello, è insostenibile. Si vedrà nelle sottosezioni C.2.2 e C.2.3 che il principio di funzionamento delle *SVM* ricalca concettualmente da vicino quello della *SRM*.

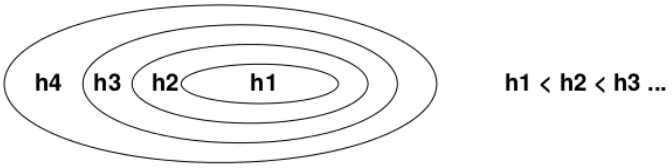


Figura C.3: Famiglia di funzioni del modello suddivise in base alla *VC dimension*

C.2 SVM

Le Support Vector Machines (*SVM*) sono un metodo d'apprendimento supervisionato, introdotte per la prima volta da Vapnik e Chervonenkis nel 1963. Nel corso degli anni sono state introdotte varie versioni di *SVM* tra cui ne sono rimarcabili quattro:

1. Quella originale: Il *Maximal Margin Classifier*.

2. La versione con il ***kernel***.
3. La versione detta ***soft-margin***.
4. La versione *soft-margin* con *kernel* che combina i tre punti precedenti.

La letteratura sull'argomento è sterminata e [41] [8] [38] sono solo alcuni dei riferimenti esistenti. Da essi si è tratto gran parte del materiale presentato in questa sezione. Le *SVM* consentono di affrontare essenzialmente tre tipi di problemi:

- Classificazione binaria
- Classificazione con più classi
- Regressione lineare

La classificazione binaria è naturale con *SVM* ed è adatta a quei problemi in cui i dati possono appartenere a due classi distinte (etichettate di solito con 1 o -1).
Nella classificazione con più classi i dati possono appartenere a più classi. Nella regressione lineare lo scopo è determinare una funzione lineare che meglio approssimi un insieme di dati.
Il problema affrontato in questa tesi attiene ai problemi di classificazione binaria.

C.2.1 Classificazione binaria

Un problema di classificazione per molti versi è simile a un problema di inferenza induttiva (capitolo 1). Da una serie di osservazioni cioè di elementi del *training set* X si deve estrarre il migliore classificatore dallo spazio delle ipotesi $f(x, \alpha)$ secondo qualche criterio di preferenza. Gli elementi del training set in un problema di classificazione binaria supervisionato sono del tipo (x_i, y_i) con $i = 1, \ldots, l$ cioè $|X| = l$, $x_i \in \mathbb{R}^n$ cioè ogni singolo elemento del *training set* ha dimensionalità n, e $y_i \in Y = \{-1, 1\}$ (o $\mathbb{B}$ in alcuni casi). La dicotomia delle etichette dell'insieme Y rende la classificazione di tipo *binaria*. Nel caso delle *SVM* le funzioni sono del tipo:

$$f : X \subseteq \mathbb{R}^n \to \mathbb{R}$$

Da un punto di vista geometrico la classificazione binaria nelle *SVM* consiste nel trovare l' iperpiano ottimo ,secondo qualche criterio, nello spazio $\mathbb{R}^n$ che separi i dati del *training set* etichettati positivamente da quelli etichettati negativamente. Se almeno un iperpiano siffatto esiste i dati del *training set* sono detti essere **linearmente separabili**. Un esempio in figura C.5
Da un punto di vista analitico è necessario una funzione lineare che classifichi correttamente il *training set*. Una funzione lineare,cioè un iperpiano, è nella forma:

$$f(x) = w \bullet x + b = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

Quindi la dipendenza dai parametri α si riconduce alla dipendenza da w — che è un vettore nello spazio $\mathbb{R}^n$ che rappresenta la normale all'iperpiano — e b uno scalare che consente all'iperpiano di muoversi parallelamente a se stesso.
Una volta scelto il classificatore dallo spazio delle ipotesi, l'ingresso $x = (x_1, x_2, \ldots, x_n)$ del *test set* (cioè di un insieme di elementi non sottoposti in fase di addestramento al modello) sarà classificato in base all'esito della funzione $sign(f(x))$ cioè associato alla classe positiva se $f(x) \geq 0$, altrimenti sarà associato alla classe negativa.

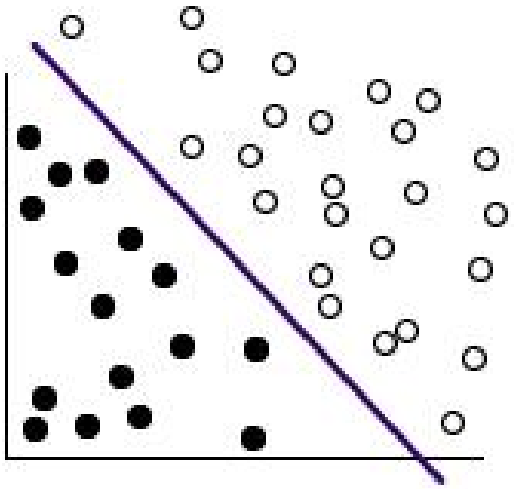


Figura C.4: Dati in $\mathbb{R}^2$ linearmente separabili

C.2.2 Classificatore a margine massimo

Il *Maximal Margin Classifier* rappresenta la versione iniziale di *SVM* e spesso in letteratura questa tecnica è nota anche come **hard margin**. Si applica quando i dati sono **linearmente separabili**. Come accennato nella sottosezione C.2.1 è necessario trovare il "migliore" iperpiano che separa linearmente i dati nel *training set*. Ai nostri scopi per lineare separabilità si intende che si può trovare una coppia (w, b) tale che i seguenti vincoli sono rispettati:

$$\begin{aligned} w \bullet x_i + b &\geq \quad 1 \qquad \text{per } y_i = +1 \quad \forall i \\ w \bullet x_i + b &\leq -1 \qquad \text{per } y_i = -1 \quad \forall i \end{aligned}$$

I due vincoli possono essere convenientemente combinati insieme in modo da ottenere un unico vincolo più compatto:

$$y_i(w \bullet x_i + b) - 1 \geq 0 \qquad \forall i \tag{C.5}$$

Il vincolo (C.5) deriva dalla constatazione che la ***decision function*** $sign(w \bullet x + b)$, cioè l'iperpiano separatore ,rispettante il vincolo (C.5), che viene selezionato come

migliore , è tale che se w e b sono scalati[6] per la stessa quantità scalare $\alpha \in \mathbb{R}^+$ il vincolo (C.5) è ancora rispettato. Dunque per eliminare questa ridondanza , e per rendere ogni *decision function* corrispondente ad un'unica coppia (w, b), viene imposto il seguente vincolo:

$$\min_{i=1,\dots,l} |w \bullet x_i + b| = 1 \tag{C.6}$$

che è un modo equivalente di scrivere il vincolo (C.5). Nel gergo tecnico delle *SVM* si suole dire che si è scelto un parametro α tale che il *margine funzionale dell'iperpiano*[7] è pari a 1 cioè la valutazione della ***decision function*** nei punti del *training set* più vicini all'iperpiano separatore è tale che[8]

$$w \bullet x_i + b = \quad 1 \qquad \text{per } y_i = +1 \tag{C.7}$$

$$w \bullet x_i + b = -1 \qquad \text{per } y_i = -1 \tag{C.8}$$

L'nsieme di iperpiani che soddisfano (C.5) sono detti **Iperpiani Canonici**[9]. Per un'*iperpiano canonico* in cui il margine geometrico e il *margine funzionale dell'iperpiano* coincidono si può dare una definizione intuitiva di **margine**.

Definizione C.1. Si indichi con d_+ la più breve distanza dell'iperpiano separatore dal più vicino esempio positivo del *training set* e con d_- la più breve distanza dall'esempio negativo più vicino del *training set*. Allora:

$$margine = d_+ + d_-$$

Si può dimostrare che la distanza di un generico punto x da un iperpiano $w \bullet x + b$ è pari a:

$$d = \frac{|w \bullet x + b|}{\|w\|}$$

In accordo alla normalizzazione definita in C.6 la distanza tra l'*iperpiano canonico* e il più vicino degli elementi del *training set* è $\frac{1}{\|w\|}$. Quindi il margine di un *iperpiano separatore canonico* secondo la definizione C.1 è allora $\frac{2}{\|w\|}$. L'immagine C.5 può essere chiarificatrice.

Lo scopo di una *SVM* è:

1. classificare correttamente il *training set*
2. e selezionare tra quelli che rispettano il punto 1 quello che generalizza meglio.

Detto succintamente il goal di una *SVM* è trovare l'iperpiano canonico ottimo che separa il *training set* , dove per ottimo si intende quello che massimizza il margine.

[6] Per scalati si intende moltiplicati

[7] Informalmente il *margine funzionale dell'iperpiano* è la distanza minima tra tutti i punti del *training set* e l'iperpiano separatore

[8] Si può dimostrare che l'esistenza di tali punti è assicurata

[9] In molte trattazioni si definisce il margine geometrico come il margine funzionale rispetto all'iperpiano normalizzato rispetto a $\|w\|$ e poi si dimostra che margine funzionale e margine geometrico coincidono imponendo il vincolo (C.5)

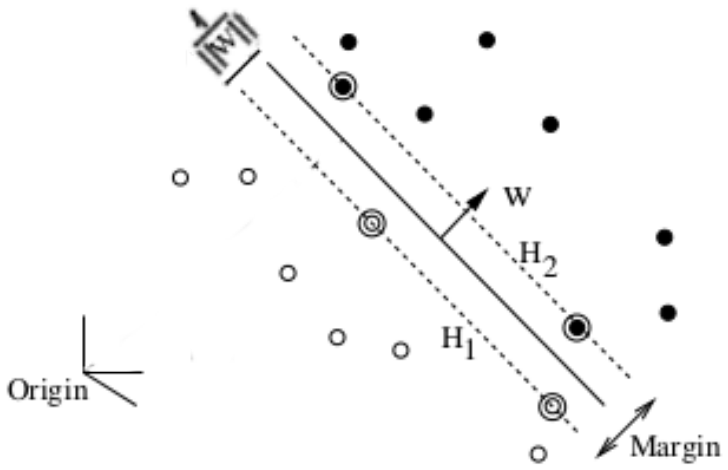


Figura C.5: Iperpiano separatore per dati linearmente separabili. H_1 e $H2$ sono paralleli dato che hanno la stessa normale w come si apprezza nelle equazioni C.7 e C.8

Legame con SRM

In questa sottosezione si vedrà come la tecnica **hard margin** di *SVM* è in stretta relazione con *SRM* introdotto nella sottosezione C.1.3. Si ha infatti che essendo i dati linearmente separabili la rispondenza con il training set è totale quindi $R_{emp} = 0$. Quindi il rischio atteso dipende unicamente dalla *VC confidence* che a sua volta dipende dalla *VC dimension* h. Quindi il classificatore migliore nel caso di *SVM* sarà quello con *VC dimension* h minima. La *SRM* in questo caso diventa una ricerca del classificatore con *VC dimension* minima. Si consideri inoltre il seguente teorema:

Teorema C.1. *Sia X un insieme di punti x di uno spazio n-dimensionale che contiene tutti gli esempi di apprendimento. Sia R il diametro della più piccola "palla" (da pensare n-dimesionale) centrata nell'origine che contiene tutti i punti di X. Allora la VC dimension ,h, dell'insieme di iperpiani* $w \bullet x + b = 0$ *aventi margine* γ *è limitata superiormente da:*

$$h \leq min\left(\left\lceil \frac{R^2}{\gamma^2} \right\rceil \ , \ n\right)+1$$

Siccome il margine è $\gamma = 2/\|w\|$ la *VC dimension* h più piccola (che abbiamo detto minimizza la *VC confidence* e avendo il R_{emp} nullo minimizza anche il rischio atteso) si ottiene per la $\|w\|$ minima cioè per il margine (pari a $2/\|w\|$) massimo. Da quanto detto segue che le *SVM* ricercano l'iperpiano col margine massimo per ottenere il classificatore migliore che da cioè più garanzie di generalizzazione secondo il risultato dell'equazione (C.4) e della *SRM*. Il discorso è analogo ,ma leggermente differente, nel caso in cui i dati non sono linearmente separabili come nel **soft-margin**(sottosezione C.2.3): in questo caso l'errore empirico non è nullo e le *SVM*

nell'ottica di minimizzare il *risk boud* cercano per il migliore compromesso tra gli errori nel *training set* e la massimizzazione del margine .

Formulazione matematica

Dalla sottosezione C.2.2 si evince che *SVM* deve risolvere un problema di programmazione lineare infatti il goal è massimizzare il margine soggetto a dei vincoli lineari cioè un problema di massimizzazione con vincoli. Ricapitolando dato il *training set* X è necessario trovare tra tutti gli iperpiani che separano i dati (i quali si assume siano linearmente separabili) quello che massimizza il margine, che è pari a $\frac{2}{\|w\|}$. Matematicamente si ha:

$$\begin{aligned} &massimizzare && \frac{2}{\|w\|} \\ &soggetto\ a && y_i(w \bullet x_i + b) - 1 \geq 0 \qquad \forall i \end{aligned}$$

Osservando che $\|w\|^2 = w \bullet w$ si può dare una formulazione alternativa ma equivalente che cambia leggermente la natura del problema rendendolo quadratico ma è più conveniente come si vedrà più avanti:

$$minimizzare \quad \frac{1}{2}(w \bullet w) \tag{C.9}$$

$$soggetto\ a \qquad y_i(w \bullet x_i + b) - 1 \geq 0 \qquad \forall i \tag{C.10}$$

Per risolvere un problema di **minimizzazione vincolato** si usano i **moltiplicatori di Lagrange**. Ad esempio si ha da $\underset{x,y}{minimizzare}\ f(x,y)$ col vincolo $g(x,y) = 0$. Si ha inoltre che $g(x,y) = y + x - 1$ allora si ha che $y = 1 - x$ costituisce il cosiddetto **feasible set** cioè il luogo dei punti dove la soluzione può venire a trovarsi. Lagrange ha dimostrato che la soluzione si ha laddove il gradiente[10] di $f(x,y)$ e quello di $g(x,y)$ puntano nella stessa direzione cioè

$$\nabla f(x,y) = \lambda \nabla g(x,y) \tag{C.11}$$

La quantità scalare λ si chiama **moltiplicatore di Lagrange** e si rende necessaria per il fatto che $\nabla f(x,y)$ e $\nabla g(x,y)$ devono avere la stessa direzione ma non necessariamente lo stesso modulo, λ è quindi un fattore moltiplicativo che ne tiene conto in modo da eguagliare anche i moduli. Il metodo dei moltiplicatori di Lagrange è valido solo per vincoli di uguaglianza ($g(x,y) = 0$) e in questo caso nelle condizioni per la risoluzione non ci sarà alcun vincolo su λ. Si definisce la funzione di Lagrange:

$$L(x,y,\lambda) = f(x,y) - \lambda g(x,y)$$

allora

$$\nabla L(x,y,\lambda) = \nabla f(x,y) - \lambda \nabla g(x,y)$$

[10]Il gradiente è una quantità vettoriale, cioè un vettore espresso tramite le sue componenti. Per i nostri scopi le singole componenti sono formate facendo le derivate parziali della funzione rispetto a ogni singola variabile

e dall'equazione (C.11) si deduce che $\nabla L(x, y, \lambda) = 0$ cioè:

$$\begin{cases} \frac{\partial L}{\partial x} = 0 \\ \frac{\partial L}{\partial y} = 0 \\ \frac{\partial L}{\partial \lambda} = 0 \end{cases}$$

Per trattare casi in cui sono presenti anche dei vincoli di ineguaglianza come nel caso delle *SVM* (vedasi vincoli (C.5)) si utilizzano le **Karush-Kuhn-Tucker (KKT)** condizioni che consistono nell'imporre — oltre alle condizioni del metodo dei moltiplicatori di Lagrange per i vincoli di uguaglianza che permangono — che il moltiplicatore relativo ad ogni vincolo di ineguaglianza venga posto maggiore uguale a 0 , e le condizioni di complementarità che consistono nell'imporre che il prodotto tra un moltiplicatore e il corrispondente vincolo sia uguale a zero. Il metodo dei moltiplicatori di Lagrange è un caso speciale (sono presenti solo vincoli di uguaglianza) delle KKT condizioni che invece hanno validità generale. Ad esempio se avessimo due vincoli $g_1(x, y) \geq 0$ e $g_2(x, y) \geq 0$ si avrebbe che $L(x, y, \lambda_1, \lambda_2) = f(x, y) - \lambda_1 g_1(x, y) - \lambda_2 g_2(x, y)$ da cui si calcola il gradiente come fatto nell'esempio precedente ottenendo:

$$\begin{cases} \dfrac{\partial L}{\partial x} = 0 \\ \dfrac{\partial L}{\partial y} = 0 \\ \dfrac{\partial L}{\partial \lambda_1} = 0 \\ \dfrac{\partial L}{\partial \lambda_2} = 0 \\ \lambda_1 \geq 0 \\ \lambda_2 \geq 0 \end{cases}$$

I vincoli di complementarità $\lambda_1 g_1(x, y) = 0$ e $\lambda_2 g_2(x, y) = 0$ non sono presenti perchè già posseduti in questo caso.
Adesso, alla luce di quanto è stato esposto, si risolverà il problema matematico posto da *SVM* e delineato nelle equazioni (C.9) e (C.10). Siccome si hanno l vincoli (tanti quanti gli elementi nel *training set*) si deve introdurre un moltiplicatore α_i con $i = 1, \ldots, l$ per ognuno degli l vincoli d'ineguaglianza ottenendo il seguente lagrangiano:

$$L_p = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i (w \bullet x_i + b) + \sum_{i=1}^{l} \alpha_i \tag{C.12}$$

dove l'ultima sommatoria scaturisce dal -1 dei vincoli in (C.10). Adesso si deve imporre pari a zero ∇L_p ,e siccome è un vettore imporre pari a zero le singole derivate parziali , e imporre le KKT condizioni ottenendo:

$$\begin{cases} \dfrac{\partial L_p}{\partial w} = 0 \\ \dfrac{\partial L_p}{\partial b} = 0 \\ \dfrac{\partial L_p}{\partial \alpha_i} = 0 \quad \forall i \\ \alpha_i \geq 0 \quad \forall i \\ \alpha_i[y_i(w \bullet x_i + b) - 1] = 0 \quad \forall i \end{cases}$$

che svolgendo le derivate parziali diventa:

$$\begin{cases} w - \sum_{i=1}^{l} \alpha_i y_i x_i = 0 & \text{(C.13a)} \\ \sum_{i=1}^{l} \alpha_i y_i = 0 & \text{(C.13b)} \\ y_i(w \bullet x_i + b) - 1 = 0 \quad \forall i \\ \alpha_i \geq 0 \quad \forall i \\ \alpha_i[y_i(w \bullet x_i + b) - 1] = 0 \quad \forall i & \text{(C.13c)} \end{cases}$$

Il problema appena impostato è un problema di programmazione quadratico dato che la funzione obiettivo ($\frac{1}{2}\|w\|^2$) è quadratica, ma è anche un **problema convesso di programmazione quadratica** perchè la funzione obiettivo è anche convessa essendo un paraboloide, ed i punti che soddisfano i vincoli cioè il *feasible set* è pure un insieme convesso[11]. Questo sta a significare che si otterrà lo stesso risultato affrontando il problema **duale** (quello descritto finora è il problema **primario**) : anzichè minimizzare rispetto alle variabili w e b soggetto a vincoli che coinvolgono i moltiplicatori lagrangiani α, si massimizza rispetto alle variabili α (variabili duali) soggetto alle relazioni ottenute prima per w e b cioè le equazioni (C.13a) e (C.13b) oltre che ad $\alpha_i \geq 0$. Alla formulazione duale si possono aggiungere le KKT condizioni della forma primaria che ci forniscono ulteriori informazioni per comprendere la struttura della soluzione. La condizione (C.13c) :

$$\alpha_i[y_i(w \bullet x_i + b) - 1] \quad \forall i$$

ci dice che solo i punti del *training set* x_i per i quali $y_i(w \bullet x_i + b) = 1$,cioè i punti più vicini all'iperpiano separatore (quelli doppiamente cerchiati in figura C.5 sugli iperpiani H_1 e H_2) , hanno valori α_i non nulli. Pertanto solo tali punti contribuiranno al calcolo dei pesi w e per tale motivo sono chiamati **vettori di supporto**.

[11] Qualsiasi vincolo lineare definisce un insieme convesso, e un insieme di N vincoli lineari simultanei definisce l'intersezione di N insiemi convessi, che è ancora un insieme convesso.

La formulazione duale del problema è detta **Wolf duale**, vediamo come si ottiene. Si sostituiscono i vincoli di eguaglianza (C.13a) e (C.13b) riguardanti b e w ($\frac{\partial L_p}{\partial w} = 0$ e $\frac{\partial L_p}{\partial b} = 0$) nella formulazione primaria cioè nell'equazione (C.12). Si ha:

$$\begin{aligned}
L_p &= \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i (w \bullet x_i + b) + \sum_{i=1}^{l} \alpha_i = \frac{1}{2}(w \bullet w) - \sum_{i=1}^{l} \alpha_i y_i (w \bullet x_i) \quad - \\
&- \sum_{i=1}^{l} \alpha_i y_i b + \sum_{i=1}^{l} \alpha_i \underset{\text{Si sostituisce l'equazione (C.13a)}}{=} \frac{1}{2}\left(\sum_{i=1}^{l} \alpha_i y_i x_i \bullet \sum_{i=1}^{l} \alpha_i y_i x_i\right) \quad - \\
&- \left(\sum_{i=1}^{l} \alpha_i y_i \Big(x_i \bullet \sum_{j=1}^{l} \alpha_j y_j x_j\Big)\right) - b \sum_{i=1}^{l} \alpha_i y_i + \sum_{i=1}^{l} \alpha_i \underset{\text{b sparisce per l'equazione (C.13b)}}{=} \frac{1}{2} \\
&\sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \, (x_i \bullet x_j) - \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \, (x_i \bullet x_j) + \sum_{i=1}^{l} \alpha_i \quad = \\
&= -\frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \, (x_i \bullet x_j) + \sum_{i=1}^{l} \alpha_i
\end{aligned} \tag{C.14}$$

quindi si è trovato che la formulazione duale da massimizzare è:

$$L_d = -\frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \, (x_i \bullet x_j) + \sum_{i=1}^{l} \alpha_i \tag{C.15}$$

soggetta ai vincoli :

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \quad \forall i \tag{C.16}$$

$$\alpha_i \geq 0 \qquad \forall i \tag{C.17}$$

Si procede come per la forma primale, cioè imponendo che il lagrangiano della forma duale L_d sia pari a zero. Si risolve il sistema, in cui la dipendenza da w e b è sparita (visionare equazione (C.15)), e si trovano gli α_i.
L'obiettivo finale è trovare l'iperpiano $f(x) = w \bullet x + b$ con i w e b cioè la coppia (w, b), che massimizzano il margine. Mediante la formulazione duale si trovano gli α_i che garantiscono i "migliori" (w, b) e vengono sostituiti nell'equazione (C.13a) per calcolare il vettore w ricordando che soltanto i *vettori di supporto* ,che hanno $\alpha_i \neq 0$,

contribuiscono al calcolo di w. Quindi si ha:

$$f(x) = w \bullet x + b = \left(\sum_{i=1}^{l} \alpha_i y_i x_i\right) \bullet x + b =$$
$$= \left(\sum_{i=1}^{l} \alpha_i y_i (x_i \bullet x)\right) + b =$$
$$= \left(\sum_{i \in sv} \alpha_i y_i (x_i \bullet x)\right) + b \tag{C.18}$$

Per calcolare b si sostituisce in C.18 un elemento x del *training set* — ed anche la corrispondente etichetta y — che rappresenta un vettore di supporto, ed i valori ottenuti vengono poi mediati quindi:

$$b = \frac{1}{|sv|} \sum_{i \in sv} \left(y_i - \left(\sum_{j \in sv} \alpha_j y_j (x_i \bullet x_j) \right) \right) \tag{C.19}$$

Fase di Test

Una volta trovati w e b, e quindi una volta che la *SVM* è stata *trained* come si usa il classificatore trovato? Immaginiamo di avere un campione n-dimensionale del *test set* m allora si usa l'iperpiano $w \bullet x + b$ per classificare il campione *unseen* m in base all'esito di $sign(w \bullet m + b)$ (l'etichetta di m è 1 oppure -1 in base a $sign(w \bullet m + b)$)

C.2.3 Soft Margin

L'esposizione della sottosezione C.2.2 è valida per un *training set* linearmente separabile. Nella maggiore parte dei casi reali i dati tipicamente non sono linearmente separabili cioè non ci sarà una *"feasible region"* perchè non esisterà nessun valore (w, b) ,cioè nessun iperpiano , per cui i vincoli C.7 e C.8 siano rispettati. L'idea ,in [14],è di rilassare tali vincoli mediante l'introduzione delle ***variabili slack*** così definite:

Definizione. Detta f la funzione caratteristica di un classificatore, che ha come dominio il training set X, e un campione $\in X(x_i, y_i)$, si definisce variabile slack del campione, rispetto alla funzione f e al margine γ, la quantità :

$$\xi((x_i, y_i), f, \gamma) = \xi_i = max(0, \gamma - y_i f(x_i))$$

e il *vettore slack* di X come:

$$\xi = \langle \xi_1, \xi_2, \ldots, \xi_l \rangle$$

Si noti che $\xi_i > \gamma$ indica un'errata classificazione del campione x_i. Quindi i vincoli C.7 e C.8 sul *training set* vengono riformulati definendo:

$$w \bullet x_i + b \geq \quad 1 - \xi_i \qquad \text{per } y_i = +1 \tag{C.20}$$
$$w \bullet x_i + b \leq -1 + \xi_i \qquad \text{per } y_i = -1 \tag{C.21}$$
$$\xi_i \geq 0 \qquad \forall i \tag{C.22}$$

Un errore in (C.20) e (C.21) occorre quando il corrispondente ξ_i eccede l'unità, ne consegue che $\sum_{i=1}^{l} \xi_i$ è un limite superiore al numero di errori sul *training set*. Allora la funzione obiettivo da minimizzare diventa:

$$\frac{1}{2}\|w\|^2 + Cf\Big(\sum_{i=1}^{l} \xi_i^k\Big) \tag{C.23}$$

soggetta ai vincoli (C.20) (C.21) e (C.22). Il problema appena formulato è convesso per qualunque k ma si sceglie sempre $k = 1$ perchè assicura l'unicità della soluzione (con $k = 1$ ne le ξ_i ne i moltiplicatori di Lagrange del problema primale appaiono nel Wolf duale problema come sarà accennato). La soluzione di questo problema di ottimizzazione è chiamato iperpiano **soft margin**. Inoltre la scelta maggiormente adottata è di approssimare[12] $f\Big(\sum_{i=1}^{l} \xi_i\Big)$ con $\Big(\sum_{i=1}^{l} \xi_i\Big) = \|\xi\|_1$ per questo talvolta si parla d'iperpiano soft margin norma-1.[13]. Il problema originale delineato in (C.23) minimizza il numero di errori di classificazione sull'insieme di addestramento ,e quindi minimizza il rischio empirico, ma è un problema NP-completo. Invece con l'approssimazione fatta si minimizza la norma di ξ che non equivale più a minimizzare il numero di errori sui campioni del *training set* ma consiste nel minimizzare gli scostamenti dei punti misclassificati dal nostro iperpiano mentre determina l'iperpiano con margine massimo per i restanti punti di X. E questo non è più un problema NP-completo. Ricapitolando la formulazione del problema di minimizzazione soft margin[14] è :

$$\textit{minimizzare} \quad \frac{1}{2}(w \bullet w) + C\sum_{i=1}^{l} \xi_i \tag{C.24}$$
$$\textit{soggetto a} \qquad y_i(w \bullet x_i + b) \geq \quad 1 - \xi_i \tag{C.25}$$
$$\xi_i \geq 0 \tag{C.26}$$

La scelta di definire la funzione obiettivo (C.24) tramite la norma risulta ottima dal punto di vista dell'errore di generalizzazione perchè si può dimostrare che quest ultimo è limitato superiormente da una quantità direttamente proporzionale alla norma dello *slack vector* ξ, per cui minimizzare la norma consente di minimizzare anche l'errore di generalizzazione.

[12] Si definisce la norma-1 di un vettore x a l componenti come $\|x\|_1 = \sum_{i=1}^{l} |x_i|$

[13] Esiste anche la formulazione norma-2 ma non ci interessa approfondirla

[14] Si intende da ora in avanti sempre norma-1

Formulazione matematica

Si procede alla stessa maniera di hard margin ,cioè per dati linearmente separabili, su dati non linearmenti separabili con la differenza che la funzione obiettivo e anche i vincoli sono cambiati (equazioni (C.24) (C.25) (C.26)) per tollerare degli errori di classificazione del *training set* e al contempo cercare di massimizzare il margine. Il problema primale diventa:

$$L_p = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}^{l}\alpha_i\{y_i(x_i \bullet w + b) - 1 + \xi_i\} - \sum_{i=1}^{l}\mu_i\xi_i \tag{C.27}$$

Imponendo $\nabla L_p = 0$ e le KKT condizioni si ha:

$$\begin{cases} \dfrac{\partial L_p}{\partial w} = \ w - \displaystyle\sum_{i=1}^{l}\alpha_i y_i x_i = 0 & \text{(C.28a)} \\ \dfrac{\partial L_p}{\partial b} = \displaystyle\sum_{i=1}^{l}\alpha_i y_i = 0 \\ \dfrac{\partial L_p}{\partial \alpha_i} = y_i(w \bullet x_i + b) - 1 + \xi_i = 0 \quad \forall i \\ \dfrac{\partial L_p}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \quad \forall i \\ \xi_i \geq 0 \quad \forall i \\ \alpha_i \geq 0 \quad \forall i \\ \mu_i \geq 0 \quad \forall i \\ \mu_i\xi_i = 0 \quad \forall i \\ \alpha_i\{y_i(w \bullet x_i + b) - 1 + \xi_i\} = 0 \quad \forall i \end{cases}$$

Con dei calcoli analoghi a quanto fatto per il caso linearmente separabile si perviene alla forma duale che consiste nel:

$$\textit{massimizzare} \quad L_d = -\frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j\,(x_i \bullet x_j) + \sum_{i=1}^{l}\alpha_i \tag{C.29}$$

$$\textit{soggetto a} \qquad 0 \leq \alpha_i \leq C \tag{C.30}$$

$$\sum_{i=1}^{l}\alpha_i y_i = 0 \tag{C.31}$$

Come per hard margin imponendo $\nabla L_d = 0$ e risolvendo si trovano gli α_i che sono usati in C.28a per calcolare il vettore dei pesi w, cui come prima contribuiranno solo i vettori di supporto in quanto sono i soli per cui $\alpha_i \neq 0$:

$$w = \sum_{i \in sv}\alpha_i y_i x_i \tag{C.32}$$

Il valore di b può essere ricavato come in (C.19). Si osservi come la formulazione duale soft margin sia uguale , fatta eccezione per il vincolo sugli α_i , alla formulazione duale del problema del margine massimo.

Parametro C e legami con SRM

Nella funzione obiettivo della formulazione soft margin (C.24) compare il parametro C, talvota detto parametro **trade-off** , che è essenzialmente un parametro di **regolarizzazione** che regola il compromesso tra raggiungere un basso errore sul *training set* e minimizzare $\|w\|$ (cioè massimizzare il margine, che è $2/\|w\|$ ed avere presumibilmente un minore errore di generalizzazione). Infatti nella forma duale l'unica dipendenza da C è negli α_i nell'equazione (C.40) ed avere un valore di C piccolo limita il valore degli α_i e a sua volta del vettore w che come si vede dall'equazione (C.32) aumenta all'aumentare degli α_i. Quindi quando C è più piccolo w è pure più piccolo , e quindi il margine è più grande e di conseguenza per il teorema C.1 si avranno migliori performances di generalizzazione tuttavia un C piccolo comporta una minore *accuracy* sul *training set*. Viceversa , per gli stessi motivi, un valore di C più grande classifica meglio il *training set* ma avrà presumibilmente un maggiore errore di generalizzazione. Quindi la regolazione del parametro C è uno step vitale perchè *SRM* è parzialmente implementato tramite la regolazione di questo parametro dato che l'incremento di C incrementa la complessità della classe delle ipotesi ,perchè in ultima analisi aumentare C comporta l'aumento della *VC dimension* h. Molto spesso per trovare il migliore valore per C si usa la validazione incrociata su vari valori del parametro. Quindi l'implementazione di *SRM* nel soft margin è approssimata tramite la regolazione del parametro C.

C.2.4 Kernels

L'approccio soft margin per addestrare dati non linearmente separabili non garantisce usualmente ottime prestazioni. Un'alternativa molto potente è quella introdotta in [7]. L'idea per gestire dati non linearmente separabili è quella di mappare questi dati in uno spazio di maggiore dimensionalità $\mathcal{H}$ detto **spazio delle features** o **spazio delle caratteristiche** in cui i dati diventano lineramente separabili e poi applicare la tecnica hard margin. Questo mapping può essere fatto tramite una funzione ϕ siffatta:

$$\phi : \mathbb{R}^n \to \mathcal{H}$$

L'immagine C.6 illustra quanto appena detto.
Inoltre $(x_i \bullet x_j)$ nella formulazione duale in (C.15) diventa $(\phi(x_i) \bullet \phi(x_j))$. Tuttavia utilizzando il **kernel trick** è possibile pensare di usare una **kernel function k** tale che

$$k(x_i, x_j) = \phi(x_i) \bullet \phi(x_j) \tag{C.33}$$

che consente di usare il *kernel* K nelle *SVM* senza esplicitamente conoscere ϕ. Una puntualizzazione su ϕ è doverosa. Se $\mathcal{H}$ è di dimensione M ϕ può essere una funzione

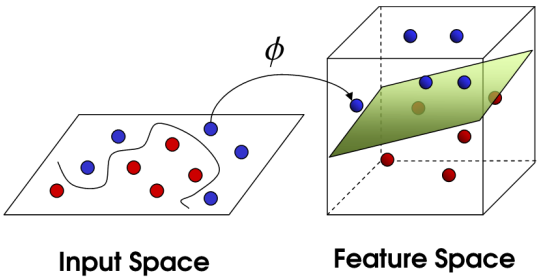


Figura C.6: *Dati in* $\mathbb{R}^2$ *resi linearmente separabili ,dalla funzione* ϕ*, nello spazio di Hilbert di dimensione tre*

che per calcolare ciascuna delle M coordinate del generico campione $x \in \mathbb{R}^d$ nello spazio H utilizza M differenti funzioni cioè:

$$\phi(x) = [\phi_1(x), \ldots, \phi_M(x)]$$

e allora il kernel diventa:

$$k(x_i, x_j) = \phi(x_i) \bullet \phi(x_j) = \sum_{d=1}^{M} \phi_d(x_i)\phi_d(x_j) \tag{C.34}$$

Un esempio è il kernel gaussiano nella tabella C.1 in cui si dimostra che $\mathcal{H}$ è di dimensione infinita così sarebbe complicato lavorare con ϕ esplicitamente. Usando il *kernel trick* si aggira il problema, infatti non vanno più svolti nè il prodotto scalare nè le due valutazioni della funzione $\phi()$ per ogni coppia (x_i, x_j) ma solo la più semplice ed efficiente valutazione del kernel, il quale se rispetta delle determinate condizioni assicura che implicitamente si sta svolgendo un prodotto scalare(ma senza svolgerne il calcolo esplicito che con alte o infinite dimensionalità di $\mathcal{H}$,come per il kernel gaussiano in tabella C.1 , è impossibile). Un altro esempio chiarificatore in cui i campioni x del *training set* appartengono allo spazio $\mathbb{R}^2$ è il seguente. Dato il *kernel*:

$$k(x_i, x_j) = (x_i \bullet x_j)^2$$

è facile trovare una funzione $\phi : \mathbb{R}^2 \to \mathcal{H}$ tale che si ha:

$$K(x, y) = (x \bullet y)^2 = \phi(x) \bullet \phi(y) \qquad \text{con } x, y \in \mathbb{R}^2$$

Infatti se $x = \langle x_1, x_2 \rangle$ e $y = \langle y_1, y_2 \rangle$ si ha che

$$(x \bullet y)^2 = (x_1 y_1 + x_2 y_2)^2 = x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 y_1 x_2 y_2 \tag{C.35}$$

e scegliendo $\mathcal{H} = \mathbb{R}^3$ si può avere

$$\phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{pmatrix}$$

quindi:

$$\phi(x) \bullet \phi(y) = \begin{pmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{pmatrix} \bullet \begin{pmatrix} y_1^2 \\ \sqrt{2}\, y_1 y_2 \\ y_2^2 \end{pmatrix} = x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 y_1 x_2 y_2$$

che coincide con quanto ottenuto in (C.35). Inoltre fissato un *kernel* non sono univoci nè i mapping ϕ nè la dimensionalità dello spazio $\mathcal{H}$. Per esempio per il *kernel* precedente si sarebbe potuto scegliere $\mathcal{H} = \mathbb{R}^3$ e

$$\phi(x) = \frac{1}{\sqrt{2}} \begin{pmatrix} (x_1^2 - x_2^2) \\ 2 x_1 x_2 \\ (x_1^2 + x_2^2) \end{pmatrix}$$

oppure $\mathcal{H} = \mathbb{R}^4$ e

$$\phi(x) = \begin{pmatrix} x_1^2 \\ x_1 x_2 \\ x_1 x_2 \\ x_2^2 \end{pmatrix}$$

Per quanto riguarda $\mathcal{H}$ questo è uno spazio di Hilbert che generalizza lo spazio euclidiano classico, e ha intrinsecamente un *inner product* definito. Comunque non ci si soffermerà su di esso piuttosto è importante comprendere in quali circostanze dato un *kernel* esiste una coppia $(\mathcal{H}, \phi)$ con la proprietà (C.33).

Teorema C.2 (Teorema di Mercer)**.** *Dato un kernel continuo e simmetrico*[15] *nell'intervallo chiuso* $\overrightarrow{a} \leq \overrightarrow{x} \leq \overrightarrow{b}$ *e* $\overrightarrow{a} \leq \overrightarrow{y} \leq \overrightarrow{b}$*. Allora k(x,y) può essere espanso come*

$$k(\overrightarrow{x}, \overrightarrow{y}) = \sum_{i=1}^{\infty} \lambda_i \, \phi_i(\overrightarrow{x}) \phi_i(\overrightarrow{y}) \tag{C.36}$$

con $\lambda_i > 0$*. Condizione necessaria e sufficiente affinchè tale espansione sia valida e la sua convergenza assoluta e uniforme è:*

$$\int_{\overrightarrow{b}}^{\overrightarrow{a}} \int_{\overrightarrow{b}}^{\overrightarrow{a}} g(\overrightarrow{x}) k(\overrightarrow{x}, \overrightarrow{y}) g(\overrightarrow{y}) \, d\overrightarrow{x} \, d\overrightarrow{y} \geq 0$$

$$\text{per } \forall g(\cdot) : \quad \int_{\overrightarrow{b}}^{\overrightarrow{a}} g^2(\overrightarrow{x}) \, d\overrightarrow{x} < \infty$$

Quindi una funzione *kernel* che soddisfa le condizioni del teorema di Mercer rappresenta un prodotto scalare in uno spazio delle features ($\mathcal{H}$) generato da una qualche trasformazione non lineare. Si noti che tale spazio delle features $\mathcal{H}$ può essere infinito e il fatto che $\forall i \ \lambda_i > 0$ implica che il kernel K è semidefinito positivo. Si ha che un kernel simmetrico K semidefinito positivo soddisfa le condizioni del Teorema di Mercer quindi condizione necessaria e sufficiente affinchè un Kernel rappresenti

Tipo di Kernel	**k(x , y)**
Polinomiale	$(x \bullet y + 1)^p$
Gaussiano	$e^{-\frac{\|x-y\|^2}{2\sigma^2}}$
Iperbolico	$\tanh(kx \bullet y - \delta)$

Tabella C.1: Tipi di kernels più conosciuti

un prodotto scalare in qualche spazio di Hilbert $\mathcal{H}$ è che il kernel sia simmetrico semidefinito positivo.[16]
Nella tabella C.1 si annoverano i kernel predefiniti più noti e maggiormente adottati. Un kernel di per sè non ci dice niente su ϕ e la dimensione di $\mathcal{H}$(spazio delle caratteristiche). Tuttavia è stato dimostrato che il *kernel* polinomiale ha per $\mathcal{H}$ una dimensione di $\binom{n+p-1}{p}$ dove si ricorda che n è la dimensionalità di ogni campione del *training set*. Invece il *kernel* gaussiano ha una dimensionalità di $\mathcal{H}$ infinita. Il suo funzionamento è molto simile alle classiche *Radial Basis Function* infatti il numero di centri ($|sv|$), i centri stessi (cioè i vettori di supporto) ed i pesi (gli α_i) sono prodotti automaticamente da *SVM*. La quantità $\|x - y\|$ rappresenta la distanza euclidea quadrata e quindi una misura di similarità e il parametro $\gamma = 1/2\sigma^2$ decide il peso di y (se questo è un vettore di supporto) nell'influenzare la classificazione di x. Un γ piccolo significa un gaussiano con una grande varianza e quindi y (si sottintende che y è un vettore di supporto) avrà un'influenza rilevante su x anche se la distanza tra x e y è grande, al contrario se γ è grande la varianza del gaussiano è piccola e y non avrà una influenza molto significativa su x (specie se molto distanti). Le prestazioni del *gaussiano* danno risultati eccellenti paragonabili alle classiche RBF.
Il *kernel* iperbolico è detto anche neurale perchè ricalca da vicino una rete neurale ed in particolare un'unità sigmoidale a due strati. Il primo strato consiste in $|sv|$ insiemi di pesi, ed ogni insieme consta n_l (la dimensionalità dei campioni) pesi, e il secondo strato consiste di $|sv|$ pesi (gli α_i) cosicché una valutazione richiede di prendere una somma pesata dei sigmoidi, essi stessi valutati sul prodotto scalare dei dati nel *test set* con i vettori di supporto. Il *kernel* iperbolico-neurale è poco utilizzato perchè solo per alcuni specifici valori di k ,δ e dei dati x le condizioni del teorema di Mercer sono soddisfatte ed è stato dimostrato che molte combinazioni dei parametri ci si riconduce al *kernel* gaussiano.
La ragione per mappare i campioni dallo spazio originario in uno spazio a dimensione molto superiore (*feature space* che come visto per il *kernel* gaussiano può essere

[15]Simmetrico significa che k(x,y) = k(y,x).

[16]Ad essere simmetrica semidefinita positiva è la matrice associata al kernel. I valori della matrice sono calcolati sulle possibili coppie dei vettori del *training set* cioè su $K(x_i, x_j)$ con $i, j = 1, \ldots, l$ e quindi possono essere raccolti in una matrice $K \in \mathbb{R}^l \times \mathbb{R}^l$ denominata matrice del kernel. E' questa matrice che deve essere simmetrica semidefinita positiva affinchè siano rispettate le condizioni del teorema di Mercer

anche infinita) è il **teorema di Cover sulla separabilità** [16]:

Teorema C.3. *Un problema di classificazione complesso, formulato attraverso una classificazione non lineare dei dati ad altà dimensionalità, ha maggiore probabilità di essere linearmente separabile che in uno spazio a bassa dimensionalità.*

Formulazione matematica

Come indicato dal teorema C.3 nello spazio $\mathcal{H}$ si avrà presumibilmente una lineare separazione dei dati e questo consente di utilizzare la formulazione hard margin in (C.15) che resta pressochè invariata. L'unica sostituzione da fare è nell'utilizzo del mapping nel prodotto scalare. Quindi la nuova formulazione con kernel diventa:

$$L_d = -\frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j y_i y_j \left(\phi(x_i) \bullet \phi(x_j)\right) + \sum_{i=1}^{l}\alpha_i = -\frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j y_i y_j \, k(x_i, x_j) + \sum_{i=1}^{l}\alpha_i \tag{C.37}$$

soggetta ai vincoli :

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \quad \forall i$$

$$\alpha_i \geq 0 \qquad \forall i$$

Anche in fase di test non ci sono problemi dato che la classificazione del campione di test x scaturisce dalla valutazione di $sign(w \bullet \phi(x) + b)$ ma $w = \sum_{i=1}^{|sv|} y_i\alpha_i\phi(x_i)$ dove gli x_i sono i vettori di supporto. Quindi sostituendo si ha:

$$sign\Big(\sum_{i=1}^{|sv|} \alpha_i y_i \phi(x_i) \bullet \phi(x) + b\Big) = sign\Big(\sum_{i=1}^{|sv|} \alpha_i y_i \, k(x_i, x) + b\Big) \tag{C.38}$$

quindi anche in fase di test non abbiamo la necessità di calcolare ϕ che comporterebbe l'effettuazione di prodotti scalari di vettori con un numero di componenti eleveto se non addirittura infinito (come il *kernel* gaussiano). E' proprio questo il vantaggio che apporta il **kernel trick**.

Prodotto scalare come misura di similarità

Il motivo per cui si predilige la formulazione duale rispetto a quella primaria è da ricercare nel fatto che in essa compare il prodotto scalare che come visto conduce alla tecnica dei *kernels*. Un altro motivo è che il prodotto scalare è una **misura di similarità** e quindi trascendendo i *kernels* classici della tabella C.1 ,se si rivelano inappropriati, è possibile definire un *kernel* su misura che implicitamente definisce una misura di similarità *customizzata* tra i campioni. Riferendoci ,senza perdere di

generalità,alla formulazione duale hard margin in (C.15) [17] si cerca di interpretare più in dettaglio come funziona il prodotto scalare ricordando che L_d va massimizzato. Assumendo che x_i e x_j sono due campioni si ha che:

1. Se x_i e x_j sono **completamente dissimili** come in figura C.7 il prodotto scalare è zero e non contribuiscono a L_d (anche se non sono esattamente ortogonali ma quasi il prodotto scalare sarà piccolo ed anche il contributo a L_d).

2. Se x_i e x_j sono **completamente simili** (o quasi) cioè il loro prodotto scalare è uno (o quasi uno) si può fare un ulteriore distinguo:

   - y_i e y_j cioè le rispettive etichette sono uguali (cioè entrambe 1 o entrambe -1), allora siccome $\alpha_i, \alpha_j \geq 0$ $\alpha_i \alpha_j y_i y_j (x_i \bullet x_j)$ per quei due specifici campioni x_i, x_j è positivo (o nullo) ma moltiplicato per -1/2 è negativo quindi L_d diminuisce. Se ne deduce che campioni simili (prodotto scalare uguale o vicino a uno) con la stessa etichetta come in figura C.9 sono ignorati da *SVM* (perchè altrimenti il margine diminuirebbe) e quindi *SVM* non li rende vettori di supporto.
   - y_i e y_j cioè le rispettive etichette sono opposte (cioè 1 e -1 o viceversa). Inoltre i vettori sono simili e il prodotto scalare è vicino a 1 come in figura C.8 quindi $\alpha_i \alpha_j y_i y_j (x_i \bullet x_j) \leq 0$ e moltiplicato per -1/2 diventa positivo e quindi contribuisce ad incrementare il margine. Quindi *SVM* renderà tali elementi del *training set* vettori di supporto (perchè fanno aumentare il margine)

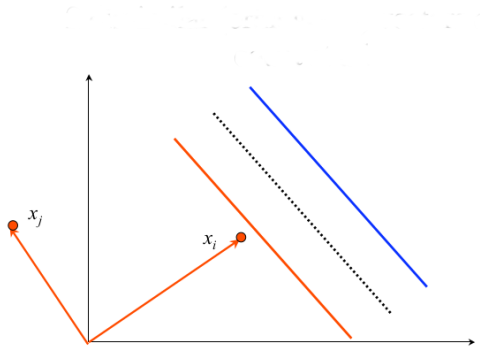


Figura C.7: *Due campioni dissimili(ortogonali), indipendentemente dall'etichetta, non contano affatto.*

[17] Per la formulazione con kernels la misurà di similarità è relativa ai campioni trasformati cioè a $\phi(x_i) \bullet \phi(x_j)$

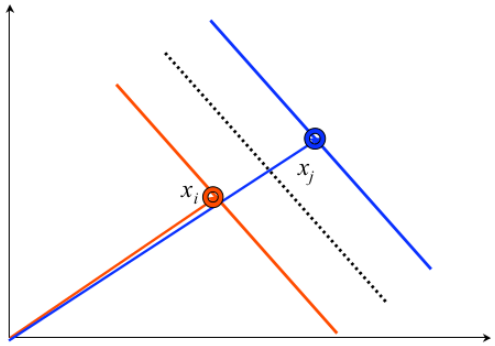


Figura C.8: *Due campioni molto simili,x_i e x_j, con etichette diverse, tendono a massimizzare il margine e sono resi vettori di supporto.*

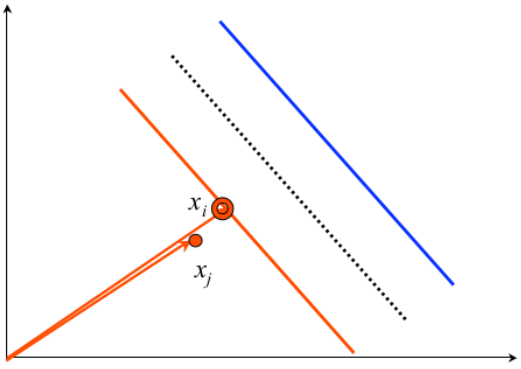


Figura C.9: *Due campioni molto simili,x_i e x_j, con etichette uguali, tendono a minimizzare il margine e quindi non vanno resi vettori di supporto. Oppure se uno di loro è un vettore di supporto (perchè con altri campioni contribuisce ad aumentare il margine) l'altro non va reso un vettore di supporto (in modo da avere coefficiente nullo) in modo da non influenzare negativamente il margine.*

String kernels

Come accennato in precedenza in C.2.4 in determinati ambiti è possibili definire un *kernel* che rappresenta una misura di similarità adatta allo specifico problema da affrontare o in cui gli elementi del *training set* sono strutturati, come ad esempio stringhe ed alberi. Un esempio molto importante , e rilevante ai fini della tesi, è quello dei **string kernels**, dei quali è possibile trovare un'introduzione in [17, p. 60]. In questa sede a titolo esemplificativo se ne descriveranno un paio. In generale l'obiettivo di uno *string kernel* è quello di rappresentare o estrapolare delle features delle stringhe che possono essere rilevanti per un problema specifico in termini di similarità o dissimilarità. Uno dei più semplici è il **Parikh kernel** che associa ad ogni stringa un vettore di numeri naturali dove ogni componente è il numero di occorrenze di un simbolo dell'alfabeto nella stringa. La dimensione del *feature space* dipende dal numero di elementi nell'alfabeto. Si consideri l' esempio nella tabella C.2 in cui si hanno quattro elementi del *training set* $\{bac, baa, cab, bad\}$ e i quattro simboli dell'alfabeto $\{a, b, c, d\}$ che rappresentano le funzioni ϕ_1,ϕ_2,ϕ_3,ϕ_4 che mappano gli elementi nel *feature space*. Si ha allora che la valutazione del *kernel* tra gli elementi *bad* e *baa* è

$$k(bad, baa) = 1 \cdot 2 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 = 3$$

Un altro *string kernel* molto noto è **all k-subsequences kernel** che calcola il prodotto scalare di due vettori in cui ogni componente è il numero di occorrenze di sottosequenze (che si ricorda possono essere contigue o non contigue) di lunghezza al più k in una stringa. Nella tabella C.3 vi un esempio di *All 2-subsequences kernel*.[18] Si ha allora che la valutazione del *kernel* nell'esempio tra gli elementi *bad* e *baa* è

$$k(bad, baa) = 6$$

All k-subsequences kernel ha sempre almeno valore 1 per la presenza della $feature\ \lambda$ cioè la stringa vuota che è sottosequenza di ogni stringa. Anche per i *string kernels* tipicamente non si calcolano le funzioni ϕ e i relativi prodotti scalari ma si usa il *kernel trick* che viene realizzato tramite tecniche di programmazione dinamica. Per un'implementazione di molti dei più noti *string kernels* compresi quelli esposti in questa sede si veda [46]. I kernel non sono solo un escamotage per renderne efficiente il calcolo, che comunque in alcuni casi come uno spazio delle caratteristiche enorme o infinito sarebbe impossibile, ma hanno due pregi notevoli che rendono *SVM* molto versatile ed applicabile nei più svariati contesti:

1. Consentono di trattare con elementi del *training set* che non sono in forma vettoriale ma che sono strutturati come ad esempio alberi e grafi

2. Consentono di trattare con elementi del *training set* che hanno lunghezza differente infatti finora si è sottaciuto che le *SVM* nella loro formulazione senza

[18] Dalla tabella nell'esempio per motivi di spazio sono omesse le *features*, cioè le sottosequenze , per le quali tutti i campioni hanno valore zero come ad esempio la sottosequenza *cc*

kernel possono funzionare solo con elementi della stessa lunghezza. Una trasformazione ϕ, implicitamente indotta da un kernel che rispetta le condizioni di Mercer, uniforma le stringhe alla stessa lunghezza M (M è la dimensione dello spazio di Hilbert $\mathcal{H}$) e quindi permette di trattare anche campioni di lunghezza diversa.

Dati	ϕ_1	ϕ_2	ϕ_3	ϕ_4
	a	b	c	d
bac	1	1	1	0
baa	2	1	0	0
cab	1	1	1	0
bad	1	1	0	0

Tabella C.2: *Parikh kernel su quattro stringhe e su un alfabeto quaternario*

Dati	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7	ϕ_8	ϕ_9	ϕ_{10}	ϕ_{11}	ϕ_{12}	ϕ_{13}	ϕ_{14}
	λ	a	b	c	d	aa	ab	ac	ad	ba	bc	bd	ca	cb
bac	1	1	1	1	0	0	0	1	0	1	1	0	0	0
baa	1	2	1	0	0	1	0	0	0	2	0	0	0	0
cab	1	1	1	1	0	0	1	0	0	0	0	0	1	1
bad	1	1	1	0	1	0	0	0	1	1	0	1	0	0

Tabella C.3: *All 2-subsequences kernel su quattro stringhe e su un alfabeto quaternario*

C.2.5 Versione soft margin kernelized

Le tecniche esposte nelle sottosezioni C.2.4 e C.2.3 sono combinate insieme per ottenere la versione delle *SVM* di riferimento maggiormente adottata. Infatti sebbene i *kernels* rendano presumibilmente il *training set* linearmente separabile non c'è alcuna garanzia che questo avvenga realmente. Inoltre anche se i deti vengono resi linearmente separabili è consigliabile utilizzare ugualmente la tecnica soft margin anzichè quella del classificatore a margine massimo (hard margin) perchè può accadere che a causa della presenza di alcuni ***outliers***[19] ci sia *overfitting* , cioè eccessivo adattamento al *training set* e scarsa capacità di generalizzare , che potrebbe essere scongiurato scegliendo un **decision boundary**[20] con soft margin che anche se ignora gli *outliers* misclassificandoli porta alla scelta di un iperpiano separatore con un

[19]Sono delle anomalie cioè degli elementi del *training set*, tipicamente in numero esiguo, che si discostano nettamente dal resto dei campioni

[20]cioè un iperpiano separatore

margine più grande e quindi con migliori capacità di generalizzare. Per questa ragione anche se la scelta di un kernel rendesse i dati linearmente separabili (asserzione che comunque non è sempre verificata) si predilige soft margin ad hard margin per cui usualmente la versione di riferimento di *SVM* combina i *kernels* con soft margin e la formulazione matematica del problema duale da risolvere diventa:

$$massimizzare \quad L_d = -\frac{1}{2}\sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \, k(x_i, x_j) + \sum_{i=1}^{l} \alpha_i \tag{C.39}$$

$$soggetto\ a \qquad 0 \leq \alpha_i \leq C \tag{C.40}$$

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \tag{C.41}$$

C.2.6 Globalità e unicità della soluzione

Uno dei vantaggi delle *SVM* rispetto alle reti neurali è che assicurano una soluzione deterministica nel senso che dato un *training set* la soluzione non dipende da elementi aleatori cosa che accade nelle reti neurali dove l'inizializzazione casuale dei pesi può condurre a una soluzione differente da un'esecuzione all'altra. Un altro vantaggio delle *SVM* è che essendo un problema di programmazione quadratica convessa con vincoli lineari assicura di trovare una soluzione globale mentre nelle reti neurali si ha la certezza di trovare un minimo locale ma non è assicurato di trovare la soluzione migliore. Nelle *SVM* il rispetto delle KKT condizioni è una condizione necessaria per trovare un minimo o un massimo globale ma in generale non sono condizioni sufficienti cioè vi è solo la garanzia di un minimo locale. Tuttavia la natura convessa del problema, rende le condizioni KKT anche sufficienti, assicura che la soluzione trovata sia sempre quella globale e questo è il motivo per cui si usa la formulazione quadratica alternativa in C.9. Ma è necessario garantire anche l'**unicità** della soluzione $\{w, b\}$ perchè potrebbero esistere delle altre coppie $\{w, b\}$ che siano pure soluzione (esistenza più minimi globali equivalenti) oppure la stessa soluzione, ma con una diversa espansione di $w = \sum_{i=1}^{|sv|} \alpha_i y_i x_i$ e questo è pure rilevalente perchè potrebbe essere un'espansione migliore con meno vettori di supporto[21]. E' garantito che la soluzione sia unica se la formulazione duale C.15 o la formulazione duale con *kernel* C.37 è strettamente convessa. Anzichè le formulazioni C.15 e C.37 si può definire l'**Hessiano** del problema che è una formulazione matriciale ottenuta come descritto nella nota 16. Si ha che:

Teorema. *Un problema quadratico è strettamente convesso se e solo se l'Hessiano associato è definito positivo.*

Quando l'Hessiano non è definito positivo la natura convessa del problema assicura che l'Hessiano sia comunque almeno semidefinito positivo cosa che assicura la glo-

[21] Una soluzione con tanti vettori di supporto vicino al numero di elementi nel *training set* è indice di *overfitting*.

balità della soluzione ma non l'unicità. Riassumendo la globalità della soluzione è sempre garantita. L'unicità è garantita solo se l'Hessiano è definito positivo.

C.2.7 Dati non bilanciati

Una situazione che merita un approfondimento è quando i dati del *training set* sono sbilanciati cioè i dati con un'etichetta positiva superano significativamente i dati con etichetta negativa o viceversa. Il quesito cui si deve rispondere è se un *training set* sbilanciato influenza o meno il funzionamento di *SVM*. Innanzitutto si ricorda che nella formulazione soft margin e anche quella soft margin combinata con l'uso di un *kernel* minimizzare la funzione obiettivo significa minimizzare l'errore sul *training set* (errore empirico) e massimizzare il margine (cioè minimizzare $\|w\|$) simultaneamente e il parametro C controlla il compromesso tra queste due quantità. Ponendoci, a titolo esemplificativo , nel caso in cui i campioni positivi sono molto meno di quelli negativi si ha che l'iperpiano separatore si muoverà nella direzione opposta a dove sono la maggior parte dei campioni negativi al fine di produrre un margine più grande col prezzo da pagare che è quello di incrementare gli errori di classificazione degli esempi positivi. Ma dato che i campioni positivi sono in minoranza l'errore di classificazione dei campioni positivi non sarà particolarmente significativo e complessivamente il valore della funzione obiettivo sarà minimizzato, che è il goal di *SVM*. Questo comporta un *overfitting* al *training set* che poi avrà delle ripercussioni negative in fase di generalizzazione dato che il classificatore ottenuto si comporterà in maniera simile a un classificatore *majority class*[22]. Esistono diversi approcci per affrontare questo problema ad esempio quello proposto in [57][23] in cui si assegna un peso σ_i ad ogni variabile slack ξ_i. Si impone un bilanciamento sull'errore accumulativo (su tutte le variabili slack) tra campioni positivi e negativi(cioè si impone che il contributo all'errore globale dei campioni positivi sia uguale al contributo di quelli negativi cioè che:

$$\sum_{y_i=1} \sigma_+^2 \xi_i^2 = \sum_{y_i=-1} \sigma_-^2 \xi_i^2$$

usando la formulazione soft-margin norma-2) e assumendo che l'errore è uguale su ogni campione si ottiene un valore globale(slegato dal singolo campione) del peso degli errori e si può scrivere $N_+\sigma_+^2 = N_-\sigma_-^2$ dove N_+ e N_- sono rispettivamente il numero di campioni positivi e di quelli negativi. Imponendo $\sigma_- = 1$ si ottiene $\sigma_+ = \sqrt{N_-/N_+}$ (anche se in realtà si utilizza una formulazione leggermente diversa) in modo da usare un unico parametro(perchè $\sigma_- = 1$) per trattare con *training set* non bilanciati anzichè due. Aggiungendo questo parametro e cambiando leggermente la formulazione di *SVM* (splittando la sommatoria per variabili slack positive e negative) si dovrebbe ottenere un classificatore che sulla carta riesce a ottenere risultati migliori a partire da *training set* non bilanciati.

[22] E' un classificatore che assegna a tutti i campioni la stessa etichetta(o positiva o negativa)

[23] La tecnica proposta in questo riferimento è applicata per una formulazione leggermente diversa di *SVM* detta WPSVM tuttavia è valida anche per le *SVM* classiche

C.2.8 Algoritmo di ottimizzazione

La rappresentazione matematica del problema che *SVM* intende risolvere — che nella sua espressione ultima è rappresentata dalla forma duale e dai relativi vincoli ,nelle quattro formulazioni precedentemente esposte — si approccia tipicamente con metodi numerici anzichè ricercare una soluzione analitica. In questa sede non interessa descrivere nel dettaglio i vari risolutori esistenti ma soltanto esporne le linee guida in modo da potere comprendere alcuni dei parametri dei *tools software* che implementano *SVM*.
L'approccio alla soluzione di questo tipo di problemi è iterativo: si parte da un punto all'interno della *feasible region* e ci si muove non lasciando tale regione e cercando di massimizzare il valore della funzione obiettivo, fino a che non risulta soddisfatto qualche criterio. I criteri per determinare la convergenza possono essere ottenuti sfruttando le caratteristiche dei sistemi convessi. Questi sono essenzialmente tre:

- **Controllare la crescita della funzione duale**
  Tale funzione raggiunge il massimo nella soluzione. Controllare il valore della funzione e, specialmente il suo aumento ad ogni passo, fornisce un semplice criterio di convergenza. L'addestramento si ferma quando il rapporto di crescita scende sotto una certa soglia. Sfortunatamente , questo criterio si è dimostrato non sempre affidabile.

- **Gap di ammissibilità**
  All'ottimo, la differenza tra il valore della funzione obiettivo del problema in forma primale e del problema duale è nulla. Di conseguenza, un possibile criterio è quello di monitorare tale differenza nota come *gap di ammissibilità*.

- **Monitorare il soddisfacimento delle condizioni KKT**
  Esse sono condizioni necessarie e sufficienti (in caso di problemi convessi come quello delle *SVM*) per la convergenza, per cui ne forniscono un criterio naturale. Ovviamente questi criteri vanno verificati all'interno di un certo margine di tolleranza.

Molto importante,in ogni caso, è il livello di tolleranza utilizzato per verificare il criterio di convergenza scelto. Una accuratezza eccessivamente elevata potrebbe comportare un tempo di esecuzione dell'algoritmo altrettanto elevato ed ingiustificato dal punto di vista dei risultati ottenuti.

C.3 SVM^{light}

Questa tesi è volta a costruire un classificatore che approssimi un *Oracolo* a partire da un insieme di campioni labellati usando come modello le *SVM*. Quindi è stato necessario scegliere una delle tante implementazioni di *SVM* e tra le più note e utilizzate si hanno senz altro **LIBSVM** e **SVM**^{light}. La scelta è ricaduta su SVM^{light} di Thorsten Joachims perchè è sembrato un tool più semplice da modificare e *customizzare* ed essendo un' implementazione di *SVM* in linguaggio C relativamente

semplice da includere all'interno della libreria GI-leraning che è scritta in C++. SVM^{light} è particolarmente adatto per problemi di *text classification* ma essendo un tool generico è adatto a tutti i problemi di classificazione binaria compreso la classificazione di stringhe che interessa in questa sede. Inoltre consente l'utilizzo di un *kernel* personalizzato come uno *string kernel*. Malgrado il porting in GI-leraning, la classificazione binaria,la definizione di un kernel personalizzato fosse possibile anche in LIBSVM la scelta è ricaduta su SVM^{light} perchè quest'ultima è sembrata più semplice da personalizzare ai propri fini.Il lato negativo della medaglia è stato che a causa dell'assenza nativa di strumenti predefiniti in SVM^{light} come la *cross validation*,lo *scaling*, metodi di ricerca dei parametri come *grid-search* ecc. è stato necessario implementare questi strumenti per fruirne. In questo modo si è però avuto pieno controllo di quello che si stesse facendo senza dovere usare implementazioni di terzi (escluso le *SVM* stesse),in alcuni casi poco documentate, a scatola chiusa. SVM^{light} mette a disposizione il calcolo di accuracy,precision e recall sul *test set* , un calcolo efficiente ma solo approssimativo di Leave-One-Out.
SVM^{light} è descritto alla pagina di riferimento: http://svmlight.joachims.org/. Qui sarà trattato solo il caso della classificazione binaria. SVM^{light} consiste di due eseguibili, svm_learn per addestrare il *training set* ed *svm_classify* per fare predizioni sui nuovi dati usando il modello precedentemente trovato. Tipicamente si avrà un train_file contenente il *training set* e un test_file contenente il *test set*. Entrambi i file devono rispettare un ben preciso formato.
Il primo step è addestrare una *SVM* ad esempio con il seguente comando:

svm_learn -c 1.5 -x 1 train_file model_file

che scrive il modello trovato alla fine dell'addestramento sul file model_file. c ed x sono dei parametri che saranno accennati più avanti. In model_file ci sarà il valore di b dell'iperpiano separatore e una riga per ogni vettore di supporto che inizierà con il corrispondente valore $\alpha_i y_i$. Per fare predizioni sul test set si usa il comando:

svm_classify test_file model_file prediction_file

Il comando legge il *test set* dal file di testo e l'iperpiano trovato in precedenza da model_file e scrive la predizione fatta tramite l'iperpiano su prediction_file. L'ordine delle linee in prediction_file corrisponde all'ordine in test_file (ad esempio la prima riga in prediction_file indica la predizione del primo campione in test_file). In prediction_file ci sono dei valori che indicano la distanza dall'iperpiano (dal modello) ed il segno determina la classe predetta.

C.3.1 Formato dei file

Sia train_file che test_file hanno il seguente formato:

$$< y > < numeroFeature >:< valore > \ \ldots \ < numeroFeature >:< valore >$$

<y> indica la classe del campione cioè può essere 1,-1,0. Il valore 1 indica un campione positivo, -1 un campione negativo , 0 un campione del *test set* di cui non si

conosce la classe di appartenenza e si vuole che sia *SVM* a predirla. Tuttavia nel caso in cui test_file contenga per ogni riga, cioè ogni campione, l'etichetta corretta anzichè 0 saranno riportate in output varie misure (accuracy,precision,recall ecc.) che sono misure orientative sulle capacità di generalizzare del predittore trovato (l'iperpiano) (cioè si conoscono le corrette etichette dei campioni nel *test set* e si vuole capire se il modello trovato fa le corrette predizioni per questi campioni). Ogni coppia <numeroFeature>:<valore> indica il valore di una particolare *feature* (cioè componente) di un campione. Per esempio, 3:0.7 5:0.1 specifica il campione $x = (0, 0, 0.7, 0, 0.1)$ infatti le features non esplicitamente specificate sono pari a 0 per default. Le coppie devono essere ordinate per numero di feature crescente. Il più piccolo numero di una feature è 1.

C.3.2 Parametri

Saranno introdotti e spiegati solo i parametri utlizzati nel lavoro sperimentale di tesi:

-v Indica il livello di verbosità delle stampe in output. Il valore di default è 1.

-x Se impostato a 1(default 0) calcola un'approssimazione di LOO.

-c E' il parametro *trade-off C* introdotto nella formulazione soft-margin. Il migliore valore di C dipende dai dati e va determinato empiricamente. Assume un valore decimale.

-j E' il parametro fattore di costo e scaturisce da un ragionamento del tutto analogo a quello fatto in C.2.7 per tenere conto di dati non bilanciati. Joachims utilizza una formulazione soft margin (norma-1):

$$\frac{1}{2}\|w\|^2 + C_+ \sum_{i:y_i=1} \xi_i + C_- \sum_{j:y_j=-1} \xi_j$$

che impostando $j = N_-/N_+$ diventa:

$$\frac{1}{2}\|w\|^2 + C\left(j \sum_{i:y_i=1} \xi_i + \sum_{j:y_j=-1} \xi_j\right)$$

j quindi è statico nel senso che non va ricercato con questo escamotage con una validazione incrociata, a differenza di C. Quindi questo parametro funge da peso e se ad esempio i campioni negativi sono il doppio di quelli positivi j=2 ed i campioni positivi saranno pesati con un peso maggiore di quelli negativi.

-t Seleziona il tipo di kernel:

0: lineare(default)

1: polinomiale

2: gaussiano

3: sigmoidale

4: personalizzato

-g E' il parametro γ del kernel gaussiano.

-e E' un parametro di ottimizzazione. Imposta la tolleranza per il criterio di terminazione nell'algoritmo di risoluzione di SVM^{light} tra un'iterazione e l'altra. Di default è 0.001 Aumentare questo valore comporta una fase di addestramento più veloce, ma aumentarlo troppo può condurre a trovare un iperpiano con scarse performances.

Appendice D

Conoscenze preliminari nell'addestramento dei classificatori

Questa appendice è propedeutica al capitolo 4 e descrive succintamente alcune tecniche basilari di *machine learning* nell'addestramento dei classificatori che saranno impiegate per la costruzione del classificatore che approssima un *Oracolo*. La trattazione non sarà esaustiva ma è volta a fare chiarezza su alcune delle scelte effettuate nel costruire l'*Oracolo approssimato*. Per comprendere appieno quest'appendice è consigliabile leggere prima l'Appendice C.

D.1 Bias-Varianza tradeoff

I concetti di modello,classificatore,classificazione,*training set* e *test set* sono stati introdotti nell'appendice C. In breve, il compito di un modello è quello di selezionare la funzione tra la famiglia di funzioni $\{f(x,\alpha)\}$ (equazione (C.1)) che da migliori garanzie di predire campioni mai visti. Dati $X = x_1, \ldots, x_l$ e le relative etichette $Y = y_1, \ldots, y_l$ si assume che c'è qualche relazione tra i due insiemi che può essere espressa come:

$$Y = f(X) + \epsilon \tag{D.1}$$

dove ϵ è un errore casuale a media zero, detto errore irriducibile. $f(\cdot)$ è una funzione ignota che rappresenta il mapping corretto tra ingressi ed etichette ed è necessario scovare la funzione $\hat{f}(\cdot) \in \{f(x,\alpha)\}$ che meglio approssima la reale relazione $f(\cdot)$ tra ingressi ed uscite (le etichette) . Si ha inoltre che non si troverà mai $f = \hat{f}$ a causa dell' errore irriducibile. Si usa come misura l'errore quadratico medio (MSE) tra $f(x)$ e $\hat{f}(x)$ per campioni x appartenenti al *test set* dato che quello che interessa minimizzare è l'errore sulle predizioni (è possibile calcolare l'MSE anche su campioni appartenenti al *training set*). Allora seguendo [23] si può dimostrare che l'MSE atteso sul test set valutato per un dato valore x_i è:

$$E[(y_i - \hat{f}(x_i))^2] = Var(\hat{f}(x_i)) + [Bias(\hat{f}(x_i))]^2 + Var(\epsilon). \tag{D.2}$$

dove

$$Bias(\hat{f}(x_i)) = E[\hat{f}(x_i)] - f(x_i)$$
$$Var(\hat{f}(x_i)) = E[(\hat{f}(x_i) - E[\hat{f}(x_i)])^2]$$

E è il valore atteso e $E[(y_i - \hat{f}(x_i))^2]$ rappresenta la media degli errori quadratici medi (MSE) sul *test set* che si potrebbe ottenere stimando ripetutamente $f(\cdot)$ (quindi ottenendo varie $\hat{f}(\cdot)$) su tanti *training set* diversi, e valutato ognuno in x_i. Il **bias** rappresenta di quanto la media della nostra stima (in un punto del *test set*) differisce dal valore reale (in quel punto) e si parla di **ottimisticamente biased** se avviene una sovrastima (cioè il valore stimato è maggiore del valore reale) viceversa si parla di **pessimisticamente biased**. Questo concetto si può estendere ad un insieme cioè si dice ad esempio che l'accuracy predetta dal modello trovato sul *training set* è ottimisticamente biased se il nostro modello stima un'accuracy maggiore dell'accuracy reale.La **varianza** invece indica la quantità di cui $\hat{f}(\cdot)$ ci si attende che cambi se noi la stimassimo usando un *training set* differente. Riepilogando un bias alto indica che la predizione fatta dal modello è lontana dal valore corretto e un'alta varianza riferisce che anche piccoli cambiamenti nel *training set* producono grandi cambiamenti di $\hat{f}$ cioè il modello è fortemente influenzato dalla specificità del *training set*. Quindi queste due quantità devono essere minimizzate ma sono in contrapposizione tra esse quando una cresce l'altra decresce ed inoltre sono in stretta relazione con la complessità del modello scelto, come si apprezza in figura D.1 quindi è necessario scegliere il modello che garantisce il giusto compromesso tra le due.

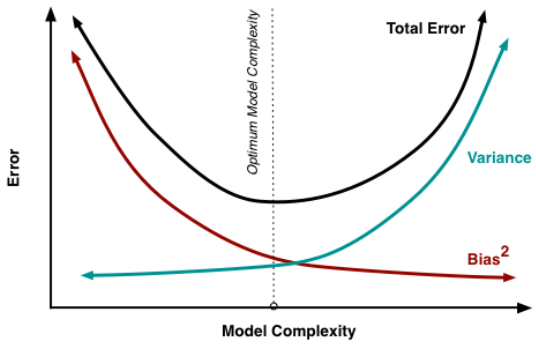


Figura D.1: *Bias-Varianza trade-off*

D.2 Model selection,Model evaluation,algorithm selection

Il workflow nella creazione di un classificatore che funga da predittore per un problema di classificazione supervisionato è tipicamente il seguente:

Algorithm selection E' necessario scegliere il modello[1]. Spesso conoscendo pregi e difetti dei vari modelli la selezione avviene manualmente a secondo del modello che sembra più cofacente al problema da risolvere. E' possibile tuttavia effettuare la selezione usando delle tecniche per comparare i vari modelli tra di loro per stabilire quale esibisce le migliori perfomances in base a qualche misura.

Model selection Con l'intento di migliorare le perfomances di generalizzazione è necessario scegliere ,a partire dal *training set*, il classificatore[2] con le migliori performances, in base a qualche misura , dallo spazio delle ipotesi. A tal fine i modelli hanno uno o più *iperparametri* cioè delle manopole da regolare.

Model evaluation Si vogliono stimare le performances di generalizzazione, le performances predittive (in base a qualche misura) del nostro classificatore basandoci sul *test set*.

Per raggiungere gli obbiettivi suddetti diventa fondamentale stabilire una misura da utilizzare. Nella sezione D.5 vi è una breve panoramica delle misure più note che si possono adottare.

D.2.1 Holdout

Questa è una delle tecniche più semplici ma anche efficaci. Si parte dal caso semplice nel quale si vuole fare solo model evaluation e già si conoscono i parametri migliori e quindi non è neccessario fare model selection. Il principio ispiratore è che fare l'addestramento del classificatore e la sua valutazione sullo stesso insieme produce un'accuracy ottimisticamente biased. **Holdout** divide i campioni disponibili in un *training set* e un *test set* in maniera casuale di solito in maniera 2/3 1/3. Si fa l'addestramento sul *training set* e poi il classificatore ottenuto viene valutato sul *test set*. Infine si rieffettua l'addestramento su tutti i campioni inizialmente disponibili[3]. Questo metodo ha due grandi limiti:

1. La suddivisone casuale dei campioni in holdout può alterare le proprietà statistiche dei campioni. Questi sono assunti essere estratti dalla stessa distribuzione di probabilità ed essere statisticamente indipendenti. Questa proprietà può venire meno: ad esempio partendo da campioni bilanciati si potrebbero generare *training set* e *test set* non bilanciati.

2. Utilizzare un numero ridotto di campioni (2/3) per effettuare l'addestramento può produrre una stima delle performances pessimisticamente biased. Si ha

[1]Malgrado in letteratura si parli di selezione dell'algoritmo e più corretto parlare di selezione di un modello perchè, ad esempio in un modello come *SVM* l'algoritmo rappresenta la specifica tecnica risolutiva utilizzata per affrontare il problema matematico delineato da *SVM*

[2]Malgrado in letteratura si parli di model selection è un classificatore che va selezionato.

[3]Riaddestrare alla fine del processo su tutti i campioni possibili è una tecnica valida solo se si è adottata la stratificazione(vedasi D.2.1) nello splitting dei campioni iniziali,perchè assicura che i campioni nei vari insiemi splittati restino statisticamente indipendenti

cioè che il classificatore non ha ancora raggiunto la sua capacità e si potrebbe apprendere un classificatore migliore utilizzando un numero di campioni più grande.

Il primo problema può essere superato usando una tecnica chiamata **stratificazione** che consiste nel suddividere l'insieme di campioni iniziale in due insiemi in modo che il bilanciamento delle classi di campioni con diverse etichette sia ancora rispettato nei risultanti sottoinsiemi. Il secondo problema invece è strutturale in quanto se si usassero tutti i campioni disponibili per l'addestramento poi si dovrebbero valutare le performances su questi stessi campioni producendo stime ottimisticamente biased. Inoltre lo scenario tipico è quello in cui si vuole fare sia model selection che model evaluation. In quest ultimo caso è necessario dividere l'insieme di campioni di partenza in (usando la stratificazione):

- **un training set**
- **un validation set**
- **un test set**

Prima si fa model selection effettuando l'addestramento sul *training set* nello spazio dei parametri. Ad esempio se si usa una tecnica come **grid-search** per ogni parametro vanno scelti i valori oppure il range in cui i valori possono variare e il passo di variazione e poi si deve effettuare l'addestramento per ogni istanza dei parametri. Alla fine ognuno di questi classificatori va valutato in base a qualche misura sui campioni del *validation set* e si seleziona il modello che massimizza tale misura. Quindi in questo modo si scelgono gli iperparametri migliori. A questo punto si rieffettua l'addestramento del classificatore risultato migliore usando sia i campioni del *training set* che quelli del *validation set*. A questo punto si fa model evaluation, cioè il classificatore ottenuto viene valutato sul *test set*. Alla fine si riaddestra su tutti i campioni inizialmente disponibili (con l'istanza di parametri trovata). Avendo usato in fase di model evaluation un classificatore addestrato con meno campioni(*training set* e *validation set*) di quelli totali è molto probabile che le misure rilevate siano delle stime pessimisticamente biased.

D.2.2 Validazione incrociata

Con la *cross validation* (validazione incrociata) si supera il limite di holdout che è quello di produrre delle stime pessimisticamente biased. Il principio di funzionamento si basa sulla constatazione che usare un numero di campioni maggiore per l'addestramento tendenzialmente fa diminuire la varianza e anche il bias. Occorre sottolineare che con questa tecnica non otteniamo un predittore ma una misura della qualità del predittore nel generalizzare. Ne esistono molte varianti, una delle più note è **k-fold-validation** che consiste nel suddividere l'insieme di campioni in k folds (cioè insiemi) ed addestrare un classificatore sui campioni di k-1 folds ed usare il restante fold per effettuare la validazione ottenendo una misura d'accuracy. Poi si rifà

la stessa cosa usando però un altro insieme di validazione riaddestrando un predittore sugli altri k-1 folds. E così via finchè tutti i k folds vengono usati esattamente una volta come insieme di validazione. Infine si fa la media dei k indici di errore ricavati(sono i valori di accuracy) ottenendo una stima della capacità di generalizzare. In questo modo si fanno k addestramenti. L'idea principale che vi sta dietro è che tutti gli elementi dell'insieme hanno l'opportunità di essere testati producendo una misura più attendibile. Un'altra tecnica è **LOO(Leave-one-out cross-validation)** che è uguale a k-fold-validation con k pari alla dimensione dell'insieme quindi l'insieme di validazione di volta in volta sarà composto da un solo elemento ma sarà necessario fare un numero di addestramenti pari al numero di campioni nell'insieme. LOO è da preferire quando l'insieme è particolarmente piccolo. LOO è quasi unbiased quindi ha un bias pessimistico anche minore di k-fold-validation ma esibisce una più grande varianza. Aumentando k il bias decresce (più accurato),la varianza aumenta (più variabilità), il costo computazionale aumenta.
Quindi dovendo fare model selection e model evaluation si suddividono i campioni iniziali in un *training set* e un *test set* (usando la stratificazione) e poi si applica k-fold-validation sul *training set* su ogni istanza degli iperparametri (usando grid-search ad esempio) e si seleziona il migliore classificatore (k-fold-validation assicura una misura più attendibile e quindi una scelta dei parametri e di conseguenza del classificatore presumibilmente migliore). In seguito si effettua l'addestramento sull'intero *training set* con gli iperparametri precedentemente ottenuti e si valuta il classificatore sul *test set*(in quest ultimo step di solito non si usa l'accuracy ma una delle misure descritte nella D.5 ad esempio). In ultima analisi si effettua un addestramento su tutti i campioni inizialmente disponibili(*training set* e *test set*).

D.2.3 Algorithm selection

Oltre che la selezione del classificatore è possibile anche stabilire quale sia il modello più adatto per un problema specifico in maniera oggettiva. Queste tecniche non sono molto utilizzate perchè spesso è il progettista che sceglie un modello piuttosto che un altro in anticipo, scegliendo in base alle caratteristiche del problema stesso il modello che sembra più idoneo. Ed in questa tesi è stato questo il caso dato che si è scelto di usare *SVM* quindi questa sottosezione non sarà approfondita più di tanto. Qui si precisa che si è optato per quest'ultima opzione in quanto il confronto dei vari modelli ne comporta comunque l'implementazione del modello per il problema in questione quindi avrebbe comportato un notevole sforzo implementativo. In questa sede si menziona solo che una delle tecniche più note in tal senso è **nested-cross-validation**.

D.2.4 Boosting

Esistono delle tecniche alternative che si basano sulla combinazione di classificatori e propongono di addestrare T classificatori diversi per lo stesso problema e di combinare i risultati. In generale la combinazione di classificatori ,si applica a problemi

ritenuti difficili che danno risultati non soddisfacenti con le tecniche classiche, al fine di ridurre il rischio di *overfitting*. Una di queste tecniche è il **boosting** ,il cui obiettivo principale è ridurre il bias, che si basa sull'addestramento sequenziale di un certo numero di classificatori detti deboli che combinati insieme producono un classificatore forte [4]. Per capirne in dettaglio il funzionamento si porta l'esempio di uno degli algoritmi di boosting più noto: AdaBoost descritto per la classificazione binaria. Ogni campione del training set ha un peso w e quei campioni classificati dai classificatori precedenti in modo errato avranno un peso maggiore (al primo passo i pesi sono assunti uniformi, ad esempio tutti pari ad $1/l$). Ad ogni passo t si seleziona il classificatore h_t che riduce l'errore pesato sui campioni del *training set* cioè che minimizza:

$$\epsilon_t = \sum_{\substack{i=1 \\ h_t(x_i) \neq y_i}}^{l} w_{i,t}$$

Poi si calcola il peso da usare per il classificatore corrente come $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$. Invece i pesi per il *training set* da utilizzare al passo successivo a partire dai pesi al passo corrente si otterrano come:

$$w_{i,t+1} = w_{i,t} e^{-y_i h_t(x_i)}$$

Assumendo di usare T classificatori deboli il classificatore forte H_T e la classificazione di un campione del *test set* x, si ottengono come:

$$H_T(x) = sign\left(\sum_{i=1}^{T} \alpha_i h_i(x)\right)$$

D.3 Data Preprocessing

Nella maggior parte dei casi i campioni grezzi in input non possono essere usati direttamente. I campioni vanno manipolati al fine di avere delle migliori performances o perchè i campioni sono composti da attributi (gli attributi sono le singole componenti dei campioni) categorici ed il modello presuppone in ingresso attributi numeri.

D.3.1 Codifiche

L'esigenza di adottare una codifica dei campioni deriva quasi sempre dal fatto che il modello adottato richiede in ingresso attributi numerici invece i campioni si compongono di attributi categorici[5]. Qui si descrivono due tipi di codifiche.

[4] Per classificatore debole e forte si intende rispettivamente con scarsa capacità di generalizzare (accuracy leggermente superiore al 50%) e con ottime capacità di generalizzazione

[5] Un esempio di attributo categorico è un componente di un campione che può assumere solo i valori rosso,verde,giallo

Integer Encoding

Per ogni attributo categorico di ogni campione, si associa una specifica categoria ad un numero dell'insieme $\mathbb{N}$. Ad esempio per l'attributo categorico che rappresenta i colori del semaforo una possibile codifica è:

$$verde \rightarrow 0 \tag{D.3}$$
$$giallo \rightarrow 1 \tag{D.4}$$
$$rosso \rightarrow 2 \tag{D.5}$$

In questo modo il campione $\langle verde, 9\rangle$ che è composto da due attributi che indicano il colore del semaforo e l'ora viene codificato come $\langle 0, 9\rangle$ e si osservi che il secondo attributo essendo già numerico non viene codificato.

One Hot Encoding

One Hot Encoding (*OHE*) (One Hot Encoding) assegna un bit ad ogni classe di un attributo. La lunghezza di questa codifica è quindi il numero di classi dell' attributo. Riproponendo l'esempio della sottosezione D.3.1 una possibile codifica per le classi dell'attributo categorico semaforo è:

$$verde \rightarrow 001 \tag{D.6}$$
$$giallo \rightarrow 010 \tag{D.7}$$
$$rosso \rightarrow 100 \tag{D.8}$$

Se il numero di classi non è troppo grande questa codifica può essere più stabile rispetto a *integer encoding*. Nel caso di campioni codificati con *OHE* non è necessario fare lo **scaling** o la **normalizzazione** (sottosezione D.3.2).

D.3.2 Scaling

Lo **scaling** o **normalizzazione** si rende necessaria quando esistono attributi che variano in un range di valori molto diversi gli uni dagli altri. Infatti se si usa un modello che calcola una distanza, come la distanza euclidea, tra i campioni si ha che l'attributo che ha i valori più grandi domina questa misura rendendo ininfluente il contributo degli altri attributi. Per evitare che ciò avvenga si può riscalare il range degli attributi in modo che varino in [0,1] o in [-1,1]. Lo scaling va effettuato per tutti gli attributi ma procedendo per singolo attributo: sia x il vettore associato a uno specifico attributo formato dal valore per quell attributo di tutti i campioni del *training set*, si ha che un possibile vettore normalizzato x' in [0 1] si ottiene con:

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

Questo calcolo va effettuato per tutti gli attributi. Un'altra tecnica di scaling è la **standardizzazione** che calcola la media $\overline{x}$ e la deviazione standard σ del vettore

attributo x ed ottiene il vettore standardizzato x' come:

$$x' = \frac{x - \overline{x}}{\sigma} \tag{D.9}$$

x' sarà a media zero e a varianza unitaria. Questo calcolo va effettutato per ogni attributo.La standardizzazione riduce il range di variazione ma a differenza del precedente non assicura che i dati varino esattamente nel range [0 1] . Il pregio della standardizzazione consiste nel fatto che a differenza del primo metodo illustrato è in grado di gestire gli *outliers* cioè un valore di un attributo molto grande rispetto agli altri valori dello stesso attributo. In generale un altro vantaggio di effettuare lo scaling è che rende più veloci alcuni modelli come le *SVM* ad esempio. Inoltre nel caso in cui i dati fossero precedentemente stati codificati con *OHE* lo scaling non si effettua perchè gli attributi avranno valori binari(0 oppure 1 quindi valori già normalizzati). Nei casi pratici per impedire di effettuare una divisione per zero quando la deviazione standard è nulla è necessario aggiungere un valore piccolo. Infine è molto importante scalare i campioni del *test set* con gli stessi valori con cui sono stati scalati i campioni del *training set* in altre parole per scalare il *test set* si devono utilizzare gli stessi valori di media e varianza calcolati e salvati precedentemente per il *training set* invece che calcolarli dal *test set*.

D.3.3 Feature extraction

Feature extraction consiste nell'estrarre dai campioni delle *features*, ossia delle informazioni rilevanti. In alcuni casi accade che i dati originali, o anche sottoposti a codifica o scaling, sono inappropriati per risolvere un determinato problema. In molti frangenti delineare delle *features* idonee al problema consente di ottenere risultati migliori. Analogo è il concetto di *feature selection* in cui una *feature* diventa un sottoinsieme degli attributi originali, mentre in *feature extraction* si applica una qualche trasformazione agli attributi selezionati. Un altro vantaggio insito in questa tecnica è la diminuizione della dimensionalità dello spazio di input individuando e rimuovendo attributi non necessari o ridondanti. Ciò dovrebbe produrre dei modelli meno complessi ed aumentare l'accuracy del predittore.

D.4 Addestrare una SVM

Per incrementare le prestazioni del modello *SVM* esistono delle regole d'oro che andrebbero seguite. Rifacendosi a [27] e [31] si ha che le linee guide per utilizzare le *SVM* al meglio sono le seguenti:

Preprocessing dei campioni *SVM* esige che i campioni da presentare in ingresso siano sotto forma di vettori di numeri reali. Quindi è necessario effettuare:

- **Codifica** Nel caso di attributi categorici ad m classi con m piccolo è conveniente utilizzare una codifica come *OHE* anzichè Integer Encoding. Integer Encoding può essere adottata nel caso in cui m sia grande.

- **Scaling** Lo scaling dei campioni è molto importante con *SVM* non solo per i motivi delineati nella sottosezione D.3.2 (che restano validi) ma anche perchè se si usano alcuni tipi di *kernel* come ad esempio il *kernel polinomiale* valori degli attributi molto grandi possono causare dei problemi numerici (il risultato del prodotto scalare di questo *kernels* produrrà valori troppo grandi per essere rappresentati correttamente). Allora si effettua una normalizzazione nel range [-1 1] o [0 1] oppure una standardizzazione. Si osservi che *kernels* come quello gaussiano sono già normalizzati cioè variano automaticamente in un range predefinito ed in questi casi vengono a cadere i problemi numerici e tecniche come la standardizzazione, nonostante non garantiscono a priori un range prefissato , possono essere convenientemente utilizzate (per impedire che gli attributi che varino in un range più grande dominino gli attributi che varino in un range più piccolo). Se è avvenuta una precedente codifica come *OHE* lo scaling non si applica.

Model selection Nel caso in cui la dimensionalità dei campioni sia molto grande oppure si è applicato un *kernel* che si spera renda i campioni linearmente separabili si potrebbe sperare di usare con fiducia hard margin anche se come detto nell'appendice C è spesso consigliabile scegliere comunque soft margin. Si ha quindi che C è un parametro spesso presente anche con dati linearmente separabili. Inoltre si ha che prima di utilizzare un *kernel* adatto a un problema specifico si prova uno di quelli predefiniti e in questo caso la scelta ricade spesso sul *kernel* gaussiano perchè quello sigmoidale rispetta le condizioni di Mercer solo per alcuni valori dei parametri e si dimostra che per alcuni valori dei parametri diventa un sottocaso del *kernel gaussiano*, e quello polinomiale consta di due o tre parametri a dispetto di un parametro nel gaussiano, e anche perchè come detto il *kernel* gaussiano è automaticamente normalizzato e non presenta difficoltà numeriche e tra quelli predefiniti si è dimostrato esibire spesso performances migliori. Quindi il *kernel gaussiano* è di solito consigliato come prima scelta per poi passare a *kernel* "ad hoc" in caso di risultati insoddisfacenti. Un possibile modo di procedere è il seguente:

- **Validazione incrociata** Tipicamente si procede dividendo l'insieme di campioni processato (codifica o scaling) in un *un training set* e un *test set*. Si effettua la validazione incrociata, tipicamente 5-fold-validation o 10-fold-validation , sul *training set*. In questo modo si fa model selection ottenendo un classificatore (e quindi gli iperparametri) che esibisce una accuracy maggiore. Infine si fa **model evaluation** sul *test set* per ottenere una qualche misura e poi si riaddestra *SVM* sull'intero insieme di campioni iniziale
- **Grid-search** Nella procedura descritta immediatamente sopra è necessario scegliere un metodo per la selezione del classificatore migliore nello step di validazione incrociata. La possibilità più semplice è quella di

utilizzare grid-search tuttavia esistono tecniche avanzate che ritornano una stima dell'accuracy della *cross-validation*. Tuttavia quando è possibile (cioè sufficienti risorse computazionali) è preferibile sempre utilizzare grid-search che è una tecnica esaustiva che da una maggiore affidabilità. Inoltre si ha che la grid-search può essere parallelizzata per incrementare le prestazioni. Inoltre utilizzando il *kernel* gaussiano si hanno solo due parametri C e γ e quindi lo spazio di ricerca dei parametri non è eccessivamente ampio. Una scelta usuale per entrambi i parametri è di scegliere un passo esponenziale. Tipici intervalli sono $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$ e $\gamma = 2^{-15}, 2^{-13}, \ldots, 2^{3}$

D.5 Misure

Per un problema di classificazione binaria è possibile definire la matrice di confusione come in figura D.2 che stabilisce le seguenti quantità numeriche per un predittore binario:

TP(true positive) Campioni positivi predetti dal classificatore come positivi.

TN(true negative) Campioni negativi predetti dal classificatore come negativi.

FP(false positive) Campioni negativi predetti erroneamente dal classificatore come positivi.

FN(false negative) Campioni positivi predetti erroneamente dal classificatore come negativi.

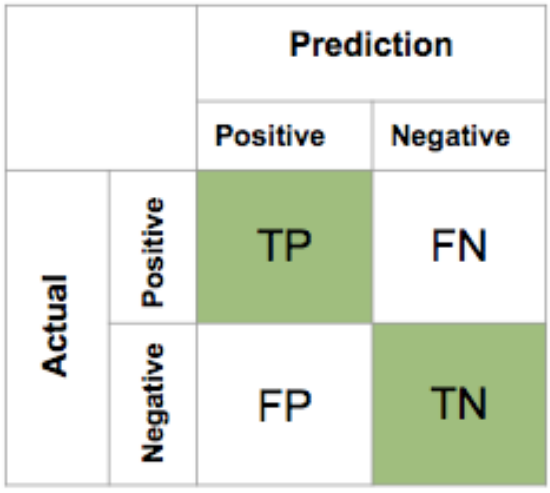


Figura D.2: *Matrice di confusione per un problema di classificazione binaria*

Allora si definisce l'**accuracy** come:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

e quindi indica la percentuale di campioni predetti correttamente rispetto alla totalità. Non è rilevante l'insieme su cui si definiscono la matrice di confusione e le varie misure, ad esempio è possibile calcolare l'accuracy sia sul *training set* che sul *test set*. Inoltre è possibile calcolare la percentuale d'errore facendo:

$$Errore = 1 - Accuracy$$

L'accuracy però non è una misura idonea per due ragioni:

- Se l'insieme su cui si calcola l'accuracy non è bilanciato la misura d'accuracy può essere ingannevole. Ad esempio immaginiamo che il classificatore sia un *majority class* che predice tutti i campioni come positivi e che il *test set* abbia il 90% di campioni positivi e i restanti negativi. In questo caso avremo un'accuracy pari a 0.9 cioè del 90% nonostante il predittore sia pessimo.
- Anche se l'insieme fosse bilanciato, dall'accuracy non si evincerebbe se siano di più i FP oppure i FN, informazione che può essere rilevante.

Per porre rimedio a questi problemi si usano **Precision (positive prediction value)** e **Recall (sensitivity o positive prediction value)** definite come:

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \end{aligned}$$

Recall ci dice quanto è buono un classificatore nel predire i campioni positivi (in quanto avendo al denominatore tutti i dati positivi dell'insieme, Recall rappresenta la percentuale di campioni positivi predetta correttamente come tale). Un predittore *majority class* che predice sempre i campioni come positivi (quindi FN=0) può ingannarci massimizzando Recall e rendendolo pari a 1. Precision ci dice quanti dei dati predetti come positivi sono positivi veramente; un predittore può essere ingannevole massimizzando quest'ultima misura se classifica come positivi i campioni dei quali ha una maggiore fiducia che siano positivi e negativi i restanti. Queste due misure non risultano comunque sufficienti nei casi, come questo lavoro, di *IIR* da stringhe in cui risulta rilevante anche come vengono classificati i campioni negativi per cui si utilizzano altre due misure $Precision^{-}$(Negative prediction value) e $Recall^{-}$(Specificity o true negative rate) [55] :

$$\begin{aligned} Precision^{-} &= \frac{TN}{TN + FN} \\ Recall^{-} &= \frac{TN}{TN + FP} \end{aligned}$$

Queste ultime due misure hanno la stessa semantica di Precision e Recall ma sui campioni negativi. Si è accennato che le singole misure di per sè possono ancora essere ingannevoli tuttavia se combinate insieme tali problemi possono essere superati ad esempio se il predittore è un $majorityclass$ sempre positivo $Recall = 1$ ma

$Recall^- = 0$ (perchè TN=0) e quindi il problema può essere individuato. Inoltre utilizzare delle misure che combinano quelle definite sopra è necessario anche perchè per scegliere in maniera automatica un classificatore rispetto ad un altro è necessario avere un'unica misura. Uno dei criteri più noti e utilizzati è **F1-score (F-measure)** che effettua la media armonica tra Precisione e Recall quindi:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Questa misura può essere utilizzata in luogo dell'accuracy per insiemi non bilanciati tuttavia non fa ancora al caso nostro perchè non tiene conto dei campioni negativi che nel nostro scenario sono rilevanti. Una possibilità è quella di definire un F1-score anche sulla classe negativa usando la stessa equazione (D.5) con $Precision^-$ e $Recall^-$ e poi mediare le due quantità, tuttavia in questo caso è più adatto e semplice utilizzare un'altra misura **MCC(Mattehws correlation coeffient)** definita come:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Questa misura è ottima nel caso di dati non bilanciati e rappresenta bene la matrice di confusione ed è sicuramente più adatta di F1-score nel caso in cui sia i campioni positivi che quelli negativi sono rilevanti. Il valore ritornato varia tra -1 e 1 e 0 è equivalente ad un'accuracy del 50%. Ad esempio un MCC di 0.5 è comparabile ad un'accuracy del 75%.

Bibliografia

[1] John M. Abela. «ETS Learning of Kernel Languages». Tesi di laurea mag. University of New Brunswick, nov. 2002 (cit. a p. 15).

[2] Dana Angluin. «Learning Regular Sets from Queries and Counterexamples». In: *Inf. Comput.* 75.2 (nov. 1987), pp. 87–106. ISSN: 0890-5401. DOI: 10.1016/0890-5401(87)90052-6 (cit. alle pp. ii, x, xi, 21, 26, 27, 36, 53).

[3] Dana Angluin. «Negative results for equivalence queries». In: *Machine Learning Journal* 5 (giu. 1990), pp. 121–150. ISSN: 0885–6125. DOI: 10.1007/BF00116034 (cit. a p. 20).

[4] Dana Angluin e Carl H. Smith. «Inductive Inference: Theory and Methods». In: *ACM Computing Surveys (CSUR) Journal* 15.3 (set. 1983), pp. 237–269. DOI: 10.1145/356914.356918 (cit. a p. 13).

[5] Austin Appleby. *MurmurHash3 hash function*. URL: https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp (cit. a p. 63).

[6] Tudor Balanescu et al. «Formal black box testing for partially specified deterministic finite state machines». In: 28 (gen. 2003) (cit. alle pp. 72, 73).

[7] Bernhard E. Boser, Isabelle M. Guyon e Vladimir N. Vapnik. «A training algorithm for optimal margin classifiers». In: COLT '92 Proceedings of the fifth annual workshop on Computational learning theory. Lug. 1992, pp. 144–152. DOI: 10.1145/130385.130401 (cit. a p. 114).

[8] Christopher J.C. Burges. «A Tutorial on Support Vector Machines for Pattern Recognition». In: *Data Mining and Knowledge Discovery* 2 (gen. 1998), pp. 121–167. ISSN: 1573-756X. DOI: 10.1023/A:1009715923555. URL: https://doi.org/10.1023/A:1009715923555 (cit. alle pp. 101, 103).

[9] Noam Chomsky. «On Certain Formal Properties of Grammars». In: *Information and Computation* II (giu. 1959), pp. 137–167. DOI: 10.1016/S0019-9958(59)90362-6 (cit. a p. 84).

[10] Tsun S. Chow. «Testing Software Design Modeled by Finite-State Machines». In: *IEEE Transactions on Software Engineering* SE-4.3 (1978), pp. 178–187. ISSN: 0098-5589. DOI: 10.1109/TSE.1978.231496 (cit. alle pp. 72, 73).

[11] Alexander Clark, Christophe Costa Florêncio e Chris Watkins. «Languages as hyperplanes: grammatical inference with string kernels». In: *Machine Learning* 82.3 (mar. 2011), pp. 351–373. ISSN: 1573-0565. DOI: 10.1007/s10994-010-5218-3. URL: https://doi.org/10.1007/s10994-010-5218-3 (cit. alle pp. 54, 59, 97).

[12] Alexander Clark et al. «Planar Languages and Learnability». In: *Grammatical Inference: Algorithms and Applications: 8th International Colloquium, ICGI 2006, Tokyo, Japan, September 20-22, 2006. Proceedings*. A cura di Yasubumi Sakakibara et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 148–160. ISBN: 978-3-540-45265-2. DOI: 10.1007/11872436_13. URL: https://doi.org/10.1007/11872436_13 (cit. alle pp. 54, 59, 97).

[13] David Combe, Colin De La Higuera e Jean-Christophe Janodet. «Zulu: an Interactive Learning Competition». In: *Journal of Machine Learning Research*. 2010, pp. 473–497. DOI: 10.1.1.370.7333. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.370.7333&rep=rep1&type=pdf (cit. a p. 54).

[14] Corinna Cortes e Vladimir Vapnik. «Support-Vector Networks». In: *Machine Learning* 20 (set. 1995), pp. 273–297. ISSN: 0885-6125. DOI: 10.1023/A:1022627411411. URL: https://doi.org/10.1023/A:1022627411411 (cit. a p. 111).

[15] Pietro Cottone, Marco Ortolani e Gabriele Pergola. «Gl-learning: an optimized framework for grammatical inference». In: CompSysTech '16 Proceedings of the 17th International Conference on Computer Systems e Technologies 2016. Giu. 2016, pp. 339–346. DOI: 10.1145/2983468.2983502 (cit. alle pp. ii, x, 53).

[16] Thomas M. Cover. «Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition». In: *IEEE Transactions on Electronic Computers* 14 (giu. 1965), pp. 326–334. DOI: 10.1109/pgec.1965.264137. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.366.5645&rep=rep1&type=pdf (cit. a p. 118).

[17] Colin De La Higuera. *Grammatical Inference: Learning Automata and Grammars*. New York, NY, USA: Cambridge University Press, 2010. ISBN: 0521763169 9780521763165 (cit. alle pp. 13, 17, 18, 87, 121).

[18] Kaibo Duan, Sathiya S. Keerthi e Aun Neow Poo. «Evaluation of simple performance measures for tuning svm hyperparameters». In: *Neurocomputing* 51 (2003), pp. 41–59. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.107.1325 (cit. a p. 57).

[19] Pierre Dupont. «Regular grammatical inference from positive and negative samples by genetic search: the GIG method». In: *Grammatical Inference and Applications*. ICGI '94. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 236–245. DOI: 10.1007/3-540-58473-0_152. URL: https://doi.org/10.1007/3-540-58473-0_152 (cit. alle pp. ii, x, 68).

[20] Mikel L. Forcada. «Neural Networks: Automata and Formal Models of Computation». 2002. URL: https://www.dlsi.ua.es//~mlf/nnafmc/pbook.pdf (cit. a p. 54).

[21] Mark E. Gold. «Complexity of automaton identification from given data». In: *Information and Control* 37.3 (giu. 1978), pp. 302–320. DOI: 10.1016/S0019-9958(78)90562-4 (cit. a p. 15).

[22] Mark E. Gold. «Language identification in the limit». In: *Information and Control* 10 (mag. 1967), pp. 447–474. DOI: 10.1016/S0019-9958(67)91165-5 (cit. alle pp. i, ix, 12, 14).

[23] Trevor Hastie, Robert Tibshirani e Jerome Friedman. *The Elements of Statistical Learning*. New York, NY, USA: Cambridge University Press, 2009. ISBN: 978-0-387-84858-7 (cit. a p. 129).

[24] J.E. Hopcroft e R.M. Karp. *A Linear Algorithm for Testing Equivalence of Finite Automata*. Rapp. tecn. Cornell Unisersity, dic. 1971. URL: http://hdl.handle.net/1813/5958 (cit. a p. 28).

[25] J.E. Hopcroft, R Motwani e J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 2ª ed. Pearson/Addison Wesley, 2007, p. 153. ISBN: 9780201441246 (cit. a p. 29).

[26] Falk M. Howar. «Active learning of interface programs». Tesi di dott. Technischen Universitat Dortmund, 2012, pp. 9–23. DOI: 2003/29486 (cit. alle pp. xi, 30, 46, 47, 50, 54).

[27] Chih-wei Hsu, Chih-chung Chang e Chih-jen Lin. *A practical guide to support vector classification*. 2010. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.224.4115 (cit. a p. 136).

[28] Michael J. Kearns e Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. Cambridge, MA, USA: MIT Press, 1994. ISBN: 0-262-11193-4 (cit. alle pp. 30, 36).

[29] Leonid (Aryeh) Kontorovich, Corinna Cortes e Mehryar Mohri. «Kernel Methods for Learning Languages». In: *Theoretical Computer Science* 405.3 (ott. 2008), pp. 223–236. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2008.06.037. URL: http://dx.doi.org/10.1016/j.tcs.2008.06.037 (cit. a p. 97).

[30] Leonid (Aryeh) Kontorovich e Boaz Nadler. «Universal Kernel-Based Learning with Applications to Regular Languages». In: *The Journal of Machine Learning Research* 10 (giu. 2009), pp. 1095–1129. URL: http://dl.acm.org/citation.cfm?id=1577069.1577108 (cit. alle pp. 54, 59, 97).

[31] Neeraj Kumar. 2014. URL: https://neerajkumar.org/writings/svm/ (cit. a p. 136).

[32] Kevin J. Lang, Barak A. Pearlmutter e Rodney A. Price. «Results of the Abbadingo One DFA Learning Competition and a New Evidence Driven State Merging Algorithm». In: Proceedings of the 4th International Colloquium on Grammatical Inference, ICGI 1998. London, UK: Springer-Verlag, mag. 1998, pp. 1–12. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.7621 (cit. alle pp. 17, 18).

[33] Willem J.M. Levelt. *Formal Grammars in Linguistics and Psycholinguistics*. Vol. I. John Benjamins Publishing Company, 2008. ISBN: 9789027290069. DOI: 10.1075/z.144 (cit. a p. 86).

[34] Ryszard Si Michalski. «Concept learning». In: *Encyclopedia of Artificial Intelligence*. A cura di Stuart C. Shapiro. Vol. I. Wiley–Interscience Publications, gen. 1986, pp. 185–194 (cit. alle pp. x, 3, 5, 10, 11).

[35] Ryszard Si Michalski. «Inferential theory of learning: Developing fondations for multistrategy learning». In: *Machine Learning: a Multistrategy Approach*. Vol. IV. Morgan Kauffman Inc., 1993 (cit. alle pp. 6, 7).

[36] Ryszard Si Michalski. «Understanding the nature of learning: Issue and research direction». In: *Machine Learning, an Artificial Intelligence Approach*. A cura di S. Ryszard Michalski, G. Jaime Carbonell e M. Tom Mitchell. Vol. II. Morgan Kaufmann, 1986. ISBN: 0-934613-00-1 (cit. a p. 2).

[37] Anil Nerode. «Linear Automaton Transformations». In: vol. 9. Proceedings of the American Mathematical Society. Ago. 1958, pp. 541–544. DOI: 10.1090/S0002-9939-1958-0135681-9 (cit. a p. 91).

[38] Andrew Ng. «Support Vector Machines» (cit. a p. 103).

[39] Daphne Norton. «Algorithms for testing equivalence of finite automata, with a grading tool for JFLAP». Tesi di laurea mag. Rochester Institute of Technology, mar. 2009, pp. 19–31. URL: http://scholarworks.rit.edu/theses/6939/ (cit. a p. 28).

[40] Jose Oncina e Pedro Garcia. «Identifying Regular Languages In Polynomial Time». In: *Advances in structural and Syntactic Pattern Recognition*. Vol. 5. Machine Perception and artificial intelligence. World Scientific, 1992, pp. 99–108. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.3126 (cit. alle pp. 17, 18).

[41] Edgar E. Osuna, Robert Freund e Federico Girosi. *Support Vector Machines: Training and Applications*. Rapp. tecn. Cambridge, MA, USA, mar. 1997. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.41.418&rep=rep1&type=pdf (cit. a p. 103).

[42] T. Pao e Carr J.W. «A solution of the syntactical induction-inference problem for regular languages». In: *Computer Languages* 3.1 (giu. 1978), pp. 53–64. DOI: 10.1016/0096-0551(78)90006-1 (cit. a p. 16).

[43] Ronald L. Rivest e Robert E. Schapire. «Inference of Finite Automata Using Homing Sequences». In: *Information and Computation* 103 (apr. 1993), pp. 299–347. ISSN: 0890-5401. DOI: 10.1006/inco.1993.1021 (cit. alle pp. 30, 36, 38).

[44] M. Raphael Robinson. «Primitive recursive functions». In: *Bulletin of the American Mathematical Society* 53 (1947), pp. 925–942. DOI: 10.1090/S0002-9904-1947-08911-4 (cit. a p. 86).

[45] José Sempere e Pedro Garcia, cur. *Grammatical Inference: Theoretical Results and Applications*. Proceedings of the 10th International Colloquium on Grammatical Inference, ICGI 2010. Set. 2010 (cit. a p. 17).

[46] John Shawe-Taylor e Nello Cristianini. *Kernel Methods for Pattern Analysis*. New York, NY, USA: Cambridge University Press, 2004. ISBN: 0521813972 (cit. a p. 121).

[47] Herbert A. Simon. «Why should machine learn?» In: *Machine Learning, an Artificial Intelligence Approach*. A cura di Mitchell, Michalski e Carbonel. Vol. I. Tioga Publishing Company, 1983, pp. 25–37 (cit. a p. 1).

[48] Bernhard Steffen, Falk M. Howar e Malte Isberner. «The TTT Algorithm: A Redundancy Free Approach to Active Automata Learning». In: *Runtime Verification*. A cura di Bonakdarpour e Smolka. Vol. 8734. Lecture Notes in Computer Science. Springer, 2014, pp. 307–322. ISBN: 978-3-319-11163-6. DOI: 10.1007/978-3-319-11164-3 (cit. a p. 78).

[49] Bernhard Steffen, Falk M. Howar e Mike Merten. «Introduction to Active Automata Learning from a Practical Perspective». In: *Formal Methods for Eternal Networked Software Systems*. A cura di Bernanrdo e Issarny. Vol. 6659. Lecture Notes in Computer Science. Springer Verlag, 2011, pp. 256–259. ISBN: 978-3-642-21455-4. DOI: 10.1007/978-3-642-21455-4_8 (cit. alle pp. 46, 51, 91).

[50] Bernhard Steffen e Malte Isberner. «An Abstract Framework for Counterexample Analysis in Active Automata Learning». In: a cura di A. Clark, M. Kanazawa e R. Yoshinaka. Vol. 34. Proceedings of the International Conference on Grammatical Inference, ICGI 2014. Kyoto, Japan, set. 2014, pp. 79–93. URL: http://jmlr.org/proceedings/papers/v34/isberner14a.pdf (cit. alle pp. 36, 38, 93).

[51] Masaru Tomita. «Dynamic Construction of Finite Automata from examples using hill-climbing». In: *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*. Vol. 4. 1982, pp. 105–108 (cit. alle pp. ii, x, 68).

[52] L.G. Valiant. «A theory of the learnable». In: *Information and Control* 27.11 (nov. 1984), pp. 1134–1142. DOI: 10.1145/1968.1972 (cit. alle pp. 12, 60).

[53] Vladimir N. Vapnik. *Estimation of Dependences Based on Empirical Data*. New York, NY, USA: Springer-Verlag, 1982. ISBN: 0387907335 (cit. a p. 101).

[54] Vladimir N. Vapnik. *The nature of statistical learning theory*. A cura di M. Jordan e Steffen L. Lauritzen. New York, NY, USA: Springer-Verlag, 1995. ISBN: 0-387-94559-8 (cit. alle pp. 98, 99).

[55] Neil Walkinshaw, Kirill Bogdanov e Ken Johnson. «Evaluation and Comparison of Inferred Regular Grammars». In: *Proceedings of the 9th International Colloquium on Grammatical Inference: Algorithms and Applications*. ICGI '08. Saint-Malo, France: Springer-Verlag, 2008, pp. 252–265. ISBN: 978-3-540-88008-0. DOI: 10.1007/978-3-540-88009-7_20. URL: http://dx.doi.org/10.1007/978-3-540-88009-7_20 (cit. a p. 139).

[56] Neil Walkinshaw et al. «A Framework for the Competitive Evaluation of Model Inference Techniques». In: *Proceedings of the First International Workshop on Model Inference In Testing*. MIIT '10. Trento, Italy: ACM, 2010, pp. 1–9. ISBN: 978-1-4503-0147-3. DOI: 10.1145/1868044.1868045. URL: http://doi.acm.org/10.1145/1868044.1868045 (cit. alle pp. 67, 71, 74).

[57] Dong Zhuang et al. «Efficient Text Classification by Weighted Proximal SVM». In: ICDM '05 Proceedings of the Fifth IEEE International Conference on Data Mining. Nov. 2005, pp. 538–545. DOI: 10.1109/ICDM.2005.56 (cit. a p. 124).

Indice analitico