Università degli studi di Bari

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

tesi di laurea

in

Algoritmi e strutture dati

Algoritmi di ricerca locale per l'apprendimento di Cutset Network

---

Relatore:
Dr. Nicola Di Mauro

Laureando:
Claudio Mastronardo

Anno Accademico 2015/2016

"Considerate la vostra semenza: fatti non foste a viver come bruti ma per seguir virtute e canoscenza"

Dante Alighieri, Inferno XXVI, 116-120

Indice

1 Introduzione

1.1 Contesto

Le tematiche di Intelligenza Artificiale (Artificial Intelligence, AI) e Apprendimento Automatico (Machine Learning, ML) sono delle aree dell'informatica studiate da diversi decenni. Nella speranza di creare macchine sempre più intelligenti, i ricercatori di tutto il mondo hanno creato e continuano a creare sempre nuovi e più sofisticati algoritmi. Molto spesso questi algoritmi sono destinati alla creazione di modelli che rappresentino, in maniera unitaria, come un programma debba effettuare un compito all'interno di una certa area di interesse. La creazione di questi modelli è basata spesso sulla somministrazione di un'insieme di dati d'esempio all'algoritmo. L'obiettivo centrale è quello di risolvere nuovi problemi addestrando le macchine secondo vari paradigmi e tecniche.

In Machine Learning si cerca di costruire programmi che, sfruttando una certa quantità di dati, siano in grado di apprendere autonomamente dei modelli in grado di descrivere e spiegare in maniera chiara suddetti dati e di predirne di nuovi.

Benché la quantità di informazioni, modelli ed algoritmi sia di notevole ampiezza, questa tesi ne analizza molto limitatamente una tipologia: quella dei *Modelli Grafici Probabilistici [6] (Probabilistic Graphical Model - PGM)*. Essi rappresentano un ricco framework in grado di codificare e descrivere distribuzioni di probabilità su domini complessi, in maniera grafica, sfruttando i concetti della teoria delle probabilità e della teoria dei grafi. Un punto di forza è sicuramente quello di descrivere il loro comportamento in maniera chiara (White Box) a differenza di altri modelli (es. Reti neurali [17]) che non permettono di capire i meccanismi e le relazioni interne alla struttura che permettano ad essi di fare inferenza (Black Box).

La famiglia dei Probabilistic Graphical Model è formata da diverse tipologie di modelli. Il lavoro svolto in questa tesi ha avuto a che fare con una nuova tipologia chiamata "Cutset Network"[2], che sfrutta modelli ed algoritmi già presentati in letteratura come gli alberi di Chow-Liu[1] e l'algoritmo di Kruskal [15]. Per la creazione di questi modelli è stato utilizzato un approccio presentato con successo in [3], che sfrutta la definizione ricorsiva delle Cutset Networks.

Questa tesi investiga l'utilizzo di algoritmi stocastici di ottimizzazione per migliorare le strutture grafiche apprese. In particolare si analizzano alcuni semplici algoritmi stocastici di ricerca locale [5] come Iterative Improvement, Randomised Iterative Improvement e GRASP (Greedy Randomised Adaptive Search Procedures). Essi sono stati applicati ed integrati al lavoro svolto

in [3].
In maniera congiunta all'applicazione degli algoritmi di ottimizzazione, è stato analizzato l'impatto di alcune modifiche all'algoritmo presentato in [3], come l'uso di foreste e l'uso di rumore nel processo di apprendimento.

L'implementazione ed analisi di codesti algoritmi hanno dato luogo a sperimentazioni numeriche riportate e descritte alla fine del manoscritto. Le sperimentazioni sono state fatte utilizzando dei datasets comunemente accettati dalla comunità scentifica come standard per il benchmarking di algoritmi di apprendimento di PGM. La parte finale del lavoro svolto è costituita dalla discussione dei risultati ottenuti.

1.2 Strutturazione del documento

I primi capitoli del documento contengono i concetti base e necessari per poter comprendere il lavoro svolto. Successivamente sono presentati gli algoritmi di ricerca utilizzati. In seguito è analizzato come essi sono stati applicati al problema specifico dell'apprendimento di Cutset Networks, ed infine sono analizzati, riportati e discussi i risultati derivati dalle sperimentazioni.
Il capitolo 2 contiene le nozioni legate al tema di Machine Learning, Density Estimation e modelli grafici probabilistici. Il capitolo fornisce definizioni ed esempi, presentando le Cutset Networks e analizzando l'algoritmo originale di apprendimento[2] e quello nuovo basato sulla decomponibilità[3].
Il capitolo 3 tratta delle nozioni legate alla ricerca stocastica locale descrivendo ad alto livello gli algoritmi applicati in questo lavoro.
Il capitolo 4 è dedicato alla descrizione di come gli algoritmi presentati nel capitolo 3 sono stati adattati al problema centrale. Inoltre sono presentate alcune modifiche all'algoritmo originale [3].
Il capitolo 5 riporta le sperimentazioni compiute sui datasets e la loro discussione.
Il capitolo 6 è dedicato alle conclusioni derivate dalle sperimentazioni.

1.3 Concetti preliminari

Nel contesto in cui è stata svolta questa tesi si è fatto uso di un insieme di dati detto "training set" o "learning set" che consiste in un insieme di osservazioni. Le osservazioni sono derivate dal problema che si è scelto di affrontare. Solitamente si assume che le osservazioni siano indipendenti tra di loro e che siano generate dalla stessa densità che non cambia nel tempo.

Formalmente, l'osservazione di un fenomeno può essere descritta tramite un insieme di k variabili aleatorie trattate in un certo ordine:

$$\boldsymbol{\mathcal{X}} = \{\mathcal{X}1, ..., \mathcal{X}k\}$$

Ogni variabile può essere categorica o numerica, continua o discreta.
La probabilità congiunta di $\boldsymbol{\mathcal{X}}$ è denotata come $\mathcal{P}(\boldsymbol{\mathcal{X}})$.
Quindi una osservazione corrisponde ad una tupla di k valori rappresentanti i valori assunti da altrettante variabili aleatorie nel momento in cui sono osservate. Formalmente :

$$Val(\boldsymbol{\mathcal{X}}) \ni \text{xD} = \text{x} = (\mathcal{X}_1 = x_1, ..., \mathcal{X}_p = x_p) \sim \mathcal{P}(\boldsymbol{X}).$$

con $Val(\boldsymbol{\mathcal{X}})$ l'insieme dei valori assumibili da $\boldsymbol{\mathcal{X}}$, ovvero il prodotto cartesiano di tutti gli insiemi di valori assumibili di tutte le variabili.

Il training set è utilizzato per la fase di apprendimento e generazione del modello con cui si decide di affrontare il problema scelto. Un algoritmo di apprendimento è un algoritmo il cui compito è quello di generare un *modello* che sia in grado di rappresentare e risolvere nuovi problemi all'interno di un determinato ambito di interesse.
Esso ha come input il training set, che utilizza per *apprendere* automaticamente particolari pattern e regole che permettano di trovare il modello ottimale capace di "capire" tutte le informazioni rilevanti e quindi *generalizzare*.

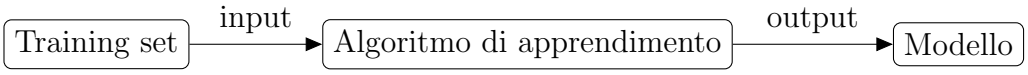


Figura 1: Un algoritmo di apprendimento prende in input un learning set e produce in output un modello.

La strategia intrapresa dall'algoritmo per scegliere il modello da fornire in output può essere di due tipi: deterministica o stocastica.
Nel primo caso l'algoritmo, dato un training set, darà in output sempre lo stesso modello. Al contrario nel secondo caso l'algoritmo darà in output modelli diversi per ogni esecuzione sullo stesso training set.
Alcune strategie stocastiche saranno discusse nel capitolo 3.

Il modello può essere informativo, predittivo o entrambi. Un modello informativo fornisce informazioni a proposito del problema, ad esempio alcune correlazioni tra variabili. Un modello predittivo fornisce informazioni su nuove realizzazioni del problema.

Una componente spesso fondamentale è rappresentata da un set chiamato "validation set". Il validation set è anch'esso una collezione di osservazioni rappresentate tramite l'avvalorazione delle k variabili aleatorie prese in considerazione. A differenza del training set, il validation set non viene utilizzato dall'algoritmo per "apprendere", ma viene utilizato durante la fase di apprendimento per fare la messa a punto dei parametri e della struttura del

modello finale. Alla fine della fase di apprendimento, quindi, ci si è costruiti un modello appreso grazie al training set e "validato" sul validation set.

Una volta appreso il modello si utilizza il cosiddetto "test set". Il test set è anch'esso una collezione di osservazioni rappresentate tramite l'avvalorazione delle k variabili aleatorie prese in considerazione. Ciò che lo distingue dal training set/validation set è il fatto che esso non viene somministrato all'algoritmo in fase di apprendimento. Esso è utilizzato a posteriori per valutare la performance del modello appreso.

2 Machine Learning e Probabilistic Graphical Models

2.1 Machine Learning

L'apprendimento automatico (in inglese *Machine Learning*) [18] è una branca dell'intelligenza artificiale che si occupa di studiare modelli ed algoritmi che permettono al software di apprendere automaticamente come risolvere nuovi problemi. Ciò è ottenuto spesso fornendo, agli algoritmi, dei dati rappresentanti osservazioni del fenomeno da studiare.
Questo insieme di dati è rappresentato dal training set e dal validation set.

2.1.1 Tipologie di apprendimento

Gli algoritmi di apprendimento automatico sono divisi in diverse tipologie. Di seguito si fa accenno solamente a due macro tipologie: apprendimento supervisionato (Supervised Learning) e apprendimento non supervisionato (Unsupervised Learning) [18].

L'apprendimento supervisionato consiste nel somministrare, all'algoritmo, un training set contenente le caratteristiche delle osservazioni e uno o più valori di output desiderato. L'output desiderato viene utilizzato per quantificare l'errore commesso dall'algoritmo e, in base ad esso, correggere e migliorare l'inferenza per le successive osservazioni.
Nell'apprendimento non supervisionato l'obiettivo dell'algoritmo è trovare pattern nascosti all'interno di un data set.

In questa tesi verranno trattati algoritmi appartenenti alla prima tipologia presentata.

2.1.2 Valutazione

Grazie ad un algoritmo di apprendimento ed un training set si è in grado, quindi, di generare un modello. Ma come capire se il modello generato è *effettivamente* adeguato? Risposta: effettuare una fase di valutazione atta a misurare il livelli di performance del modello appreso.

Una parte fondamentale dell'apprendimento automatico è rappresentata dalle metriche e tecniche di valutazione dei modelli appresi. La valutazione di un modello è quella fase in cui si misura il comportamento, in termini di accuratezza, su un set di dati non precedentemente "visti" . Per far ciò si utilizza il test set.

Una delle metriche più semplici e comuni è l'accuratezza, che misura la percentuale di osservazioni correttamente classificate da un modello di

classificazione. Altre misure in classificazione sono : precision, recall e specificity.

La misura dell'errore, generalmente, migliora rispetto alle osservazioni presenti nel training set. Una misura d'errore troppo "buona" sul training set potrebbe portare il modello ad avere comportamenti negativi su un set di osservazioni non ancora "viste". Questo problema è chiamato *overfitting*. Quindi l'obiettivo dell'apprendimento è quello di *generalizzare*, in modo tale da comportarsi in maniera ottima anche su osservazioni non ancora viste.
La figura 2 mostra, in un esempio, l'evoluzione delle misure di errore sul training e sul test set all'aumentare della complessità del modello. Si può notare come all'aumentare della complessità del modello l'errore sul training set diminuisca sempre più ma l'errore sul test set decresca fino ad un certo punto per poi riprendere ad aumentare. Dal punto in cui si peggiora l'errore sul test set ma si continua a migliorare quello sul training set inizia la fase di overfitting. L'obiettivo di un buon addestramento, quindi, è quello di trovare il giusto grado di complessità tale da minimizzare l'errore sul test set e non fare overfitting.
Nel caso di density estimation (presentato in 2.1.3) la complessità del modello aumenta con l'aumentare del numero di parametri indipendenti utilizzati per rappresentare il modello.

Altre tecniche atte all'evitare l'overfitting si occupano di utilizzare anche il validation set (introdotto prima). In particolare in situazioni computazionalmente abbastanza favorevoli è molto efficace l'uso del *K-Fold cross-validation*. Questa procedura prevede il partizionamento del set originale di osservazioni in K slices (con K deciso a priori). Per ogni slice si avvia una procedura di apprendimento utilizzando i restanti K-1 slices come training set, usando il corrente slice come validation set. La procedura viene fatta per tutti i K slices e la valutazione finale del modello è ricavata mediando tutte le K valutazioni.

2.1.3 Density Estimation

In statistica per stima della densità s'intende la costruzione di una stima di una funzione di densità di probabilità inosservabile. Questo viene fatto tramite l'utilizzo di un insieme di dati osservati.

Sfruttando le definizioni date nel capitolo 2.1, possiamo ricondurre l'insieme dei dati osservati utilizzati per fare density estimation come il training set $\boldsymbol{X}$ da cui costruire un modello che sia in grado di codificare questa stima.

Definendo con $\mathcal{P}(\boldsymbol{X})$ la distribuzione congiunta di $\mathcal{X}$ possiamo rappresentare la probabilità di un set $\boldsymbol{T}$ di N osservazioni come:

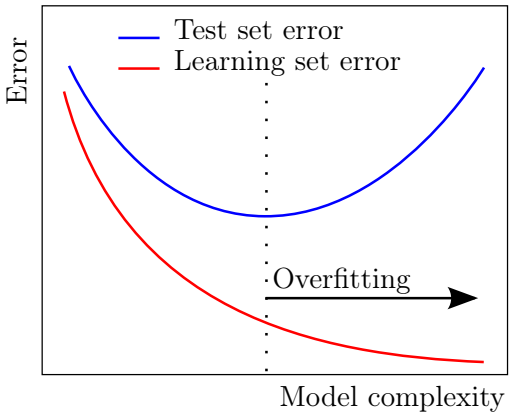


Figura 2: Cercando di minimizzare l'errore sul learning set si può superare un punto per cui l'errore sul training set continua a migliorare ma quello sul test set inizia a peggiorare, portando così all'overfitting.

$$\mathcal{P}(T) = \prod_{i=1}^{N} \mathcal{P}(\mathcal{X} = xd_i)$$

con xd_i la i-esima tupla di valori del training set.

Per quanto riguarda la valutazione, nel caso di density estimation, una misura di errore adeguata è rappresentata dal *loglikelihood*, che si occupa di comparare la bontà dell'adattamento del modello appreso rispetto alla densità obiettivo da codificare. Una misura nota è quella della divergenza di Kullback-Leibler (DKL) [16], anche chiamata information divergence o relative entropy:

$$\text{Err}(M\text{D}) = DKL\,(\mathcal{P}||M\text{D}) = \sum_x \mathcal{P}(\text{x})\,\log\frac{\mathcal{P}(x)}{MD(x)}$$

con MD il modello M appreso sul training set D.
La divergenza può essere interpretata come il numero di bit extra richiesti per codificare esempi dalla densità originale utilizzando codice basato su MD, al posto di $\mathcal{P}$. Questa misura si occupa di misurare la differenza tra due distribuzioni di probabilità. In questo caso una distribuzione è quella pura da stimare e l'altra è quella codificata dal modello appreso. Se la misura è 0 allora le due densità sono uguali.
Un'altra misura utilizzata è il negative log-likelihood:

$$\text{nlogll}(M\text{D,x}) = \text{-log}M\text{D(x)}$$

con nlogll(MD,x) = 0 se le due distribuzioni coincidono.
Queste misure non sono calcolabili in maniera esatta dato che la densità vera che si sta cercando di approssimare è ignota e anche perché di solito

il numero di variabili in gioco è molto alto causando il calcolo delle misure improponibile.

Quindi le misure sono calcolate in maniera stimata basandosi su un test set D di N osservazioni tramite:

$$\hat{Err}\text{D'}(M\text{D}) = \frac{1}{N}\sum_{i=1}^{N} Err(MD, xD'i)$$

In questa tesi saranno utilizzate alcune tecniche di machine learning per apprendere modelli che cercano di risolvere problemi di stima delle densità, sfruttando le formule e le definizioni ivi fornite.

2.2 Probabilistic Graphical Models

I Probabilistic Graphical Model (PGM)[6] sono una classe di modelli che sfruttano la teoria dei grafi e la teoria delle probabilità per descrivere in forma compatta distribuzioni di probabilità su cui è possibile fare inferenza. Attraverso una struttura grafica ed un insieme di parametri permettono di codificare una distribuzione di probabilità multivariata identificando le relazioni tra le variabili aleatorie prese in considerazione. Identificare e esplicitare i possibili valori da esse assumibili corrisponde a determinare i *parametri* del modello.

La potenza espressiva dei PGM permette loro di rappresentare facilmente le relazioni presenti tra le variabili trattando esse come nodi del grafo e gli archi tra i nodi come interazioni fra di esse. Quindi il modello grafico rappresenta strutturalmente la distribuzione esplicitandone le variabili e i parametri associati mostrando eventuali relazioni di indipendenza condizionata tra di esse.

Formalmente un PGM è formato da un set di parametri θ ed una struttura grafica $\mathcal{G} = (V, E)$ con V il set di vertici del grafo e $E \subset V$ x V il set di archi del grafo. Catturare ed interpretare il significato codificato dalla struttura di un PGM dipenderà dal particolare tipo di modello impiegato. Ad un primo livello gerarchico si distinguono i modelli *diretti* (directed) e i modelli *indiretti* (undirected), direttamente collegati alla tipologia di archi utilizzati.

2.2.1 Reti Bayesiane

Un esempio di PGM è rappresentato dalle reti bayesiane. Le reti bayesiane rappresentano un insieme di variabili aleatorie e le loro dipendenze attraverso un *grafo aciclico* ***diretto***. In questo tipo di rappresentazioni le variabili aleatorie vengono rappresentate tramite dei nodi e forniscono informazioni utili alla strutturazione del problema. Gli archi tra i nodi invece indicano delle condizioni di dipendenza. I parametri quantificano la distribuzione di probabilità di ogni variabile $\mathcal{X}i$ condizionata dai propri padri nel grafo.
Un esempio è riportato nella figura 3 .

Nella figura d'esempio possiamo notare la presenza di 11 variabili aleatorie identificate attraverso nodi del grafo. Ogni arco rappresenta una relazione indicante la dipendenza condizionale di una variabile verso un'altra.
Questa rete, per esempio, potrebbe essere usata sfruttando un insieme di osservazioni per derivare nuove informazioni riguardo la malattia.
Nella rete bayesiana possiamo notare come l'età, il sesso, peso, alcohol e il livello di ferro influiscano sulla malattia al fegato. Si può anche notare come non esistano dipendenze dirette tra età ed itterizia, ma bensì una dipenden-

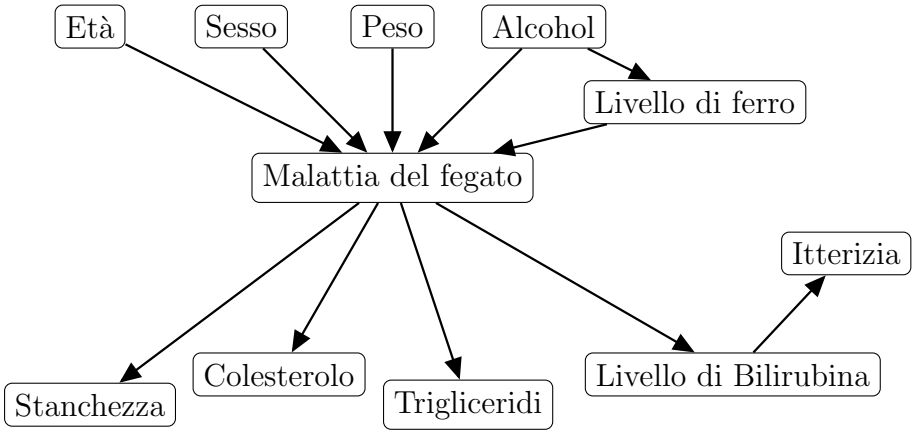


Figura 3: Rete bayesiana che modella malattie al fegato(in particolare la *cirrosi biliare primitva* e *steatosi epatica*). Esempio ispirato da [4]

za transitiva tramite malattia del fegato e livello di bilirubina. Grazie alle dipendenze condizionali espresse dal modello grafico, si può quindi codificare una fattorizzazione della probabilità congiunta tramite il prodotto delle distribuzioni condizionali:

$$\mathcal{P}(\mathcal{X}) = \prod_{i=1}^{P} \mathcal{P}(\mathcal{X}i|\mathcal{P}a(\mathcal{X}i))$$

Con $\mathcal{X}$ un array di possibili valori delle variabili aleatorie prese in considerazione, $\mathcal{X}i$ la i-esima variabile aleatoria e $\mathcal{P}(\mathcal{X}i|\mathcal{P}a(\mathcal{X}i))$ la probabilità della variabile $\mathcal{X}i$ condizionata alle variabili padre $\mathcal{P}a(\mathcal{X}i)$. L'utilizzo di una rete bayesiana, oltre a essere visualmente vantaggiosa, permette di visualizzare le probabilità e le relazioni in maniera sintetica rispetto ad una ipotetica tabella di contingenza (contenente la probabilità di ogni possibile configurazione di variabili).

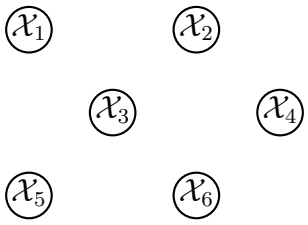
La struttura di una rete bayesiana può essere costruita grazie alla conoscenza di un esperto nell'area di appartenenza del problema che si sta cercando di risolvere oppure grazie ad un algoritmo di apprendimento automatico.
Nel caso in cui si utilizzi un algoritmo di apprendimento automatico, la costruzione della rete può essere utile ad evidenziare particolari relazioni di dipendenza e indipendenza che magari a priori non fossero note.

2.2.2 Alberi di Chow-Liu

La struttura "albero di Chow-Liu" è un modello *trattabile* ad albero che permette di approssimare una rete bayesiana. È un modello grafico presentato da Chow e Liu in [1]. Una assunzione principale in questa struttura è, quindi, che ogni nodo ha un solo padre e la radice è il padre diretto/indiretto di tutti i nodi e non ha padre a sua volta.
La struttura dell'albero è appresa sfruttando la cosiddetta *matrice delle mutue informazioni*. Questa matrice rappresenta sulle righe e sulle colonne tutte le variabili aleatorie. In corrispondenza della cella sulla riga i e colonna j si ha un valore che rappresenta la mutua informazione della coppia formata dalla i-esima e j-esima variabile aleatoria. La mutua informazione rappresenta una misura della mutua dipendenza tra le due variabili aleatorie. Date due variabili u,v la mutua informazione è così calcolata:

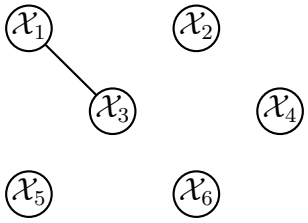
$$Iuv = \sum_{xuxv} \mathcal{P}uv(xu,vv) \log \frac{\mathcal{P}uv(xu,vv)}{\mathcal{P}u(xu)\mathcal{P}v(xv)} ,$$
$$u,v \in V,\ u \neq v,\ x_v \in Val(v),\ x_u \in Val(u)$$

Con V l'insieme delle variabili aleatorie e $Val(u)$ l'insieme dei valori assunti dalla variabile u.
Piu sarà alto il valore della mutua informazione tra due variabili, più sarà utile includere la loro relazione causale nell'albero finale. I valori di mutua informazione per ogni coppia di variabili è direttamente calcolato dal training set. Questo procedimento dà vita ad una matrice simmetrica come quella presente nella parte destra della figura 4.
Una volta calcolata la matrice delle mutue informazioni per tutte le coppie di variabili si procede con la costruzione dell'albero tramite un algoritmo di *Maximum Weight Spanning Tree* (MWST) usando le mutue informazioni come pesi per gli archi tra le variabili. L'obiettivo è quello di costruire un albero $G_T = (V, E_T)$ utilizzando tutte le variabili come nodi, tale che ogni arco $(u, v) \in (E_T)$ sia pesato da $I(u, v)$ e la sommatoria dei pesi degli archi finale sia la massima raggiungibile.
L'algoritmo utilizzato da Chow e Liu è l'algoritmo di Kruskal (esempio nella figura sottostante 4). L'algoritmo di Kruskal è di tipo greedy, tutti gli archi candidati sono ordinati in ordine decrescente di peso. Ad uno ad uno, partendo dall'arco con peso maggiore, vengono aggiunti all'albero (inizialmente avente l'insieme degli archi vuoto). Ad ogni passo si verifica che l'arco aggiunto non crei un ciclo. Nel caso in cui si venga a creare un ciclo l'arco viene scartato. Il processo va avanti fino a quando non sono state collegate tutte le n variabili aleatore all'albero avendo, quindi, *n - 1* archi.



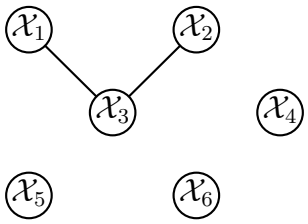
	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$	$\mathcal{X}_4$	$\mathcal{X}_5$	$\mathcal{X}_6$
$\mathcal{X}_1$		21	25	14	3	15
$\mathcal{X}_2$			23	17	5	8
$\mathcal{X}_3$				11	13	18
$\mathcal{X}_4$					7	19
$\mathcal{X}_5$						4

(a) Situazione iniziale



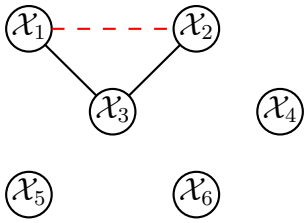
	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$	$\mathcal{X}_4$	$\mathcal{X}_5$	$\mathcal{X}_6$
$\mathcal{X}_1$		21	25	14	3	15
$\mathcal{X}_2$			23	17	5	8
$\mathcal{X}_3$				11	13	18
$\mathcal{X}_4$					7	19
$\mathcal{X}_5$						4

(b) Si sceglie l'arco con peso maggiore e lo si aggiunge all'albero



	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$	$\mathcal{X}_4$	$\mathcal{X}_5$	$\mathcal{X}_6$
$\mathcal{X}_1$		21	25	14	3	15
$\mathcal{X}_2$			23	17	5	8
$\mathcal{X}_3$				11	13	18
$\mathcal{X}_4$					7	19
$\mathcal{X}_5$						4

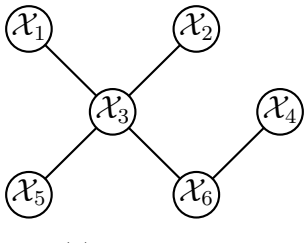
(c) Si sceglie l'arco con peso maggiore scartando quelli già aggiunti e lo si aggiunge all'albero controllando che non si vengano a creare cicli



	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$	$\mathcal{X}_4$	$\mathcal{X}_5$	$\mathcal{X}_6$
$\mathcal{X}_1$		21	25	14	3	15
$\mathcal{X}_2$			23	17	5	8
$\mathcal{X}_3$				11	13	18
$\mathcal{X}_4$					7	19
$\mathcal{X}_5$						4

(d) In questa situazione l'arco con peso maggiore se aggiunto all'albero creerebbe un ciclo, quindi viene scartato

Figura 4



	$\mathcal{X}_1$	$\mathcal{X}_2$	$\mathcal{X}_3$	$\mathcal{X}_4$	$\mathcal{X}_5$	$\mathcal{X}_6$
$\mathcal{X}_1$		21	25	14	3	15
$\mathcal{X}_2$			23	17	5	8
$\mathcal{X}_3$				11	13	18
$\mathcal{X}_4$					7	19
$\mathcal{X}_5$						4

Figura 5: Il processo va avanti fino a quando non si è collegati tutti i nodi mantenendo le proprietà di un albero.

Algorithm 1 Learn Chow-Liu tree($\mathcal{D}$,X)

Input: : il learning set $\mathcal{D}$,il set di variabili X
Output: : ($\mathcal{T}$,θ), un albero $\mathcal{T}$ avente θ come parametri

1: MI = 0$|\mathcal{X}|$ x $|\mathcal{X}|$ ▷ Inizializzazione matrice
2: Per ogni coppia Xu,$Xv \in \mathcal{X}$:
3: MIu,v = calcolaMutuaInformazione(Xu,Xv,$\mathcal{D}$)
4: T = MWST(MI)
5: $\mathcal{T}$ = attraversaAlbero(T) //Costruito scegliendo un nodo random come radice
6: θ = calcolaParametri($\mathcal{D}, \mathcal{T}$)
7: return ($\mathcal{T}, \theta$)

Una volta appresa la struttura dell'albero si procede scegliendo un nodo random come radice. Successivamente si attraversa l'albero definendo i padri e i figli. Una volta calcolati i relativi parametri è possibile sfruttare le relazioni, in essa codificate, per poter calcolare la distribuzione rappresentata dal nuovo albero tramite:

$$\mathcal{P}(\mathcal{X}) = \prod_{i=1}^{P} \mathcal{P}(\mathcal{X}i|\mathcal{P}a(\mathcal{X}i))$$

utilizzando la semplice probabilità non condizionata : $\mathcal{P}(\mathcal{X}\rangle)$, per la variabile posta come radice dall'algoritmo di apprendimento.

2.2.3 Misture di PGM

Il compito di questo capitolo è quello di dare un'infarinatura sulle misture di probabilistic graphical model.

Invece di utilizzare un'unica potenziale struttura, si sceglie di trattare il problema tramite una combinazione di semplici PGM, ognuno modellante una distribuzione di probabilità congiunta su tutte le stesse variabili.
Quindi è definita *Mistura (Mixture)*[8] un insieme di m PGM. Ad ogni PGM viene dato un peso w.

La densità di probabilità definita da una mistura è calcolata mediando quelle delle differenti densità codificate da ogni PGM:

$$\mathcal{P}(\mathcal{X}) = \sum_{i=1}^{m} wi\mathcal{P}i(\mathcal{X})$$

con

$$\sum_{i=1}^{m} wi = 1 \wedge \forall i : wi \geq 0$$

Questo approccio permette l'utilizzo di algoritmi di apprendimento semplici e sfrutta la media le predizioni di ogni PGM per rappresentare delle classi di densità più ricche.

Ogni modello può essere visto come una distribuzione alternativa e la media pesata può essere visto come un modo per trattare il problema in modo differente dove la mancanza di esempi non permette di discriminare.

Un modo di creare queste misture è quello di creare gli m PGM addestrati su altrettanti m versioni del training set. Ogni versione del training set è il risultato di una leggera alterazione applicata al training set originale. Questo permette di far avere alla mistura una visione più generale del problema. La creazione dei training sets alterativi può essere fatta attraverso il bagging[7]. Questo meta-algoritmo consiste nel creare un nuovo training set D' della stessa dimensione N del training set originale D scegliendo N numeri naturali, in maniera uniforme ed indipendente, $ri \in [1, N]$ e popolando D' tramite:

$$\mathrm{x}D'\mathrm{i} = \mathrm{x}D\mathrm{ri}\ \forall i \in [1, N']$$

con xDri la j-esima osservazione del dataset D.
Il processo è illustrato nella figura 6 presa da [8].

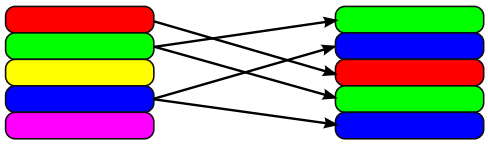


Figura 6: Il processo di bagging sceglie in maniera randomica delle osservazioni dal training set originale e le copia nel nuovo training set. Si può notare come nel nuovo training set c'è la possibilità di trovare una stessa osservazione più volte.

2.2.4 Alberi OR

Un albero OR è un albero utilizzato per rappresentare lo spazio di ricerca esplorato durante inferenza probabilistica condizionata [14]. Ogni nodo dell'albero rappresenta una variabile aleatoria e ne è etichettato di conseguenza. Ogni arco uscente da un nodo v rappresenta il condizionamento della variabile nodo v per un certo valore x_v ed è etichettata con la probabilità che la variabile v assuma valore x_v dato il cammino che parte dalla radice ed arriva al nodo. Ogni nodo quindi avrà un numero di archi uscenti pari alla dimensione del dominio dei valori assumibili dalla variabile da esso rappresentata. Quindi per ogni inferenza, per ogni nodo si sceglie l'arco uscente etichettato con il valore assunto dalla variabile e si procede per i successivi nodi. Arrivati alla foglia si calcola $\mathcal{P}(x)$ sfruttando il cammino percorso dalla radice. La distribuzione rappresentata da un albero OR T equivale a:

$$\mathcal{P}(x) = \prod_{(vi,vj)\in pathT(x)} w(v_i, v_j)$$

con $pathT(x)$ il cammino dalla radice alla foglia $l(x)$ corrispondente all'assegnazione x e $w(v_i, v_j)$ la probabilità con cui è etichettato l'arco tra il nodo v_i e il nodo v_j. La produttoria in congiunzione alla struttura dell'albero, rappresenta quindi la probabilità di osservare x calcolata tenendo conto delle dipendenze funzionali rappresentate dalla relazione padre $\rightarrow$ figlio.

2.2.5 Cutset Networks

Le *Cutset Networks* (CNets) sono delle strutture ibride contenenti alberi di Chow-Liu le cui radici sono rappresentate da nodi OR, con nodi OR utilizzati come nodi interni e alberi di Chow-Liu come foglie. Sono state presentate da Rahman et al. in [2].

Ogni nodo OR rappresenta una variabile aleatoria Xi ed ogni arco partente da esso rappresenta il condizionamento della variabile Xi per un certo valore $x^j i \in Val\ (Xi)$ pesato con la probabilità wi,j che la variabile Xi assuma il valore $x^j i$.

Una cutset network è rappresentata da una coppia $< \mathcal{G}, \xi >$.
Dove $\mathcal{G} = \mathcal{O} \cup \{\mathcal{T}\infty, ..., \mathcal{T}\mathcal{H}\}$, con $\mathcal{O}$ un albero OR e $\{\mathcal{T}\infty, ..., \mathcal{T}\mathcal{H}\}$ le sue foglie di alberi Chow-Liu.
Con $\xi = \boldsymbol{w} \cup \{\theta\infty, ..., \theta\mathcal{H}\}$; dove $\boldsymbol{w}$ corrisponde ai parametri dell'albero OR e $\{\theta\infty, ..., \theta\mathcal{H}\}$ corrisponde ai parametri degli alberi figli.
Di seguito ne viene data una definizione ricorsiva che ne descrive l'essenza in maniera elegante e sintetica.

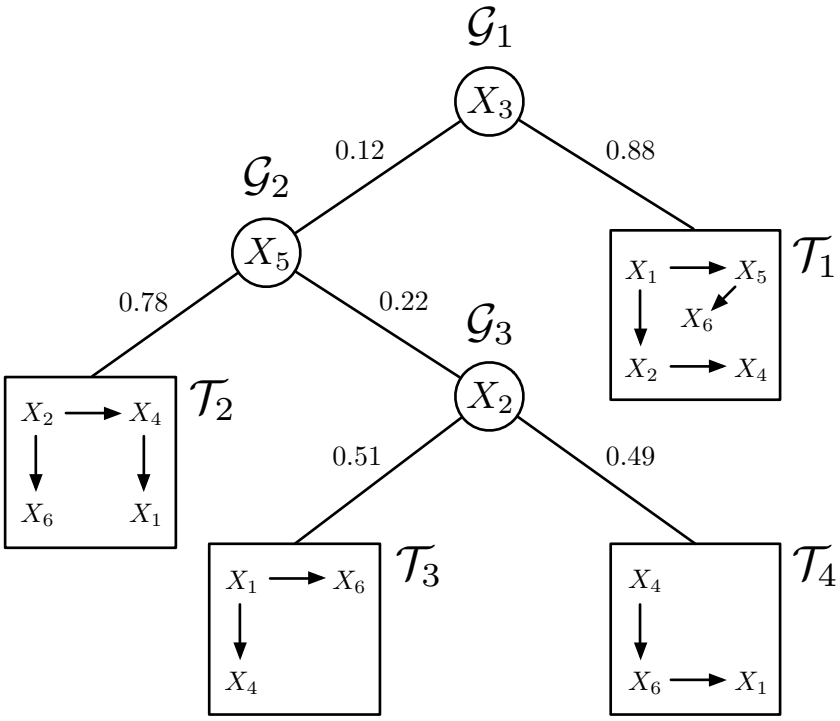


Figura 7: Una cutset network avente 6 features.*Figura presa da [3]*

Definizione: Cutset Network.
Sia $\mathcal{X}$ un insieme di variabili aleatorie discrete, una Cutset Network è:
1. un albero di Chow-Liu avente come scope $\mathcal{X}$

oppure

2. data $\mathcal{X}\rangle \in \mathcal{X}$ una variabile con |Dominio($\mathcal{X}\rangle$)|= k, condizionata graficamente da un nodo OR, una disgiunzione pesata di k cutset networks $\mathcal{G}\rangle$ aventi tutte lo scope ($\mathcal{X}$\i), $dovetuttiipesiwi, jconj = 1, ..., k, sisommanoad1eX\idenotal'insien$

Una cutset network d'esempio è riportata nella figura 7.

Nell'esempio riportato possiamo notare un $\mathcal{X} = \{\mathcal{X}\infty, ..., \mathcal{X}\}$. Ogni variabile ha uno scope la cui dimensione è 2. Sono presenti tre nodi OR, aventi due figli ciascuno, che rappresentano le variabili $\mathcal{X}\ni, \mathcal{X}\bigtriangledown, \mathcal{X}\in$, e quattro alberi di Chow-Liu come foglie. Si può notare come al primo livello di profondità della cutset network da un lato sia presente il nodo OR $\mathcal{X}\bigtriangledown$, dall'altro sia presente già un albero di Chow-Liu. Questo indica che nel caso in cui la variabile $\mathcal{X}\ni$ assuma il valore per cui si scenda verso il figlio sinistro, sia adeguato trattare l'osservazione discriminando subito sulla variabile $\mathcal{X}\bigtriangledown$; mentre nel caso in cui si scenda verso il figlio destro di $\mathcal{X}\ni$ sia adeguato continuare trattare l'osservazione tramite l'albero di Chow-Liu $\mathcal{T}1$.

La distribuzione rappresentata da una cutset network è data da:

$$\mathcal{P}(\mathcal{X}) = \left[\prod_{(vi,vj)\in pathO(x)} w(vi, vj) \right] \cdot \Big(Tl(x) \Big)$$

con $pathO(x)$ il cammino dalla radice all'unica foglia $l(x)$ e $Tl(x)$ l'albero di Chow-Liu foglia associato a $l(x)$. Nel lavoro originale [2] Rahman et al. utilizzano anche il concetto di mistura applicato alle Cutset networks che generalizza misture di alberi di Chow-Liu [13], cercando così di migliorare l'accuratezza.
In questa tesi non è stato fatto utilizzo di misture di CNets benchè disponibili nel progetto di Di Mauro et al. [3].

2.2.6 Apprendimento delle Cutset Networks

Rahman et al. in [2] hanno presentato il primo assoluto algoritmo per l'apprendimento di una cutset network.
L'algoritmo utilizzato sfrutta un approccio *divide et impera* per apprendere in maniera ricorsiva la rete da un dataset $\mathcal{D}$. Si basa sul selezionare una variabile $\mathcal{X}' \in \mathcal{X}$, alla volta, usando una determinata *euristica di splitting* ed

utilizzare la variabile $\mathcal{X}'$ come radice della cutset network. Da qui generare un arco per ogni valore del dominio dei valori della feature. Quindi ripetere l'approccio, in maniera ricorsiva, su ogni arco utilizzando per ogni arco le variabili aleatorie $\mathcal{X} \setminus \mathcal{X}'$ e le sole istanze del dataset il cui valore della feature padre (in questo caso $\mathcal{X}'$) sia quello dell'arco corrispondente.

Ogni volta viene verificato un certo *criterio di terminazione* e se soddisfatto si interrompe l'algoritmo di apprendimento della cutset network e si esegue l'algoritmo classico per l'apprendimento di un albero di Chow-Liu sulle features e istanze rimanenti in quell'arco, così generando una foglia.

L'*euristica di splitting* proposta da Rahman et al. consiste nel calcolare la riduzione attesa in entropia causata dal conoscere il valore di una data feature.
In particolare l'entropia di un dataset $\mathcal{D}$ definito su un set V di variabili è dato da:

$$\hat{\mathcal{H}}(D) = \frac{1}{|V|} \sum_{v \in V} \mathcal{H}\mathcal{D}(v)$$

con l'entropia di un dataset $\mathcal{D}$ relativa ad una variabile v :

$$\mathcal{H}\mathcal{D}(v) = - \sum_{xv \in \Delta v} \mathcal{P}(xv) log(\mathcal{P}(xv))$$

Calcolando, quindi, l'*information gain* condizionato ad una variabile v usando la seguente espressione:

$$Gain\mathcal{D}(v) = \hat{\mathcal{H}}(D) - \sum_{xv \in \Delta v} \frac{|\mathcal{D}xv|}{\mathcal{D}} \hat{\mathcal{H}}(\mathcal{D}xv)$$

con $\mathcal{D}xv = \{x^{\mathrm{i}} \in \mathcal{D} | x^{\mathrm{i}}v = vv\}$.
Quindi viene scelta la variabile che ha il più alto information gain.

Come *criterio di terminazione* si può pensare di stoppare lo splitting se il numero di istanze rimaste è inferiore ad un certo threshold oppure se l'entropia è troppo bassa.

2.2.7 Decomposability based Cutset Network learning

Un altro algoritmo di apprendimento per le cutset networks è stato presentato dal Di Mauro et al. in [3]. L'approccio utilizzato da questo algoritmo sfrutta la decomponibilità del likelihood di una cutset network. L'algoritmo riformula la ricerca nello spazio delle strutture come un task di ottimizzazione che massimizza il likelihood direttamente sui dati costruendo la cutset network in maniera ricorsiva.

Il primo passo dell'algoritmo è quello di costruire un singolo albero di Chow-Liu partendo dall'intero training set. Il passo successivo è quello di

determinare se sia possibile scegliere una variabile, da cui costruire un nodo OR e appendere ad esso degli alberi di Chow-Liu costruiti sulle restanti variabili, che permetta di ottenere una likelihood migliore rispetto a quella iniziale. Se viene trovata tale variabile allora si procede alla decomposizione, sostituzione della nuova struttura alla vecchia e applicazione dell'algoritmo in maniera ricorsiva sui figli del nodo OR appena creato.
La scelta di uno splitting viene fatta attraverso l'utilizzo del Bayesian Information Criterion (BIC) come in [9]. In particolare si misura la differenza di BIC fra la vecchia e la nuova struttura. Il BIC è un criterio per la selezione di un modello tra una classe di modelli e rappresenta una forma di regolarizzazione. Il valore BIC per una cutset network è:

$$\text{scoreBIC}(\langle \mathcal{G}, \gamma \rangle) = \log \mathcal{PD}(\langle \mathcal{G}, \gamma \rangle) - \frac{\log M}{2} Dim(G)$$

con Dim($\mathcal{G}$) il numero di nodi OR presenti in $\mathcal{G}$. L'utilizzo di un altro criterio come il BDe[10] è anche possibile

Quindi date due cutset networks $\mathcal{G}$ e $\mathcal{G}'$, con $\mathcal{G}'$ ottenuta da $\mathcal{G}$ sostituendo un albero foglia con una nuova cutset network avente come radice un nodo OR, :

$$\text{scoreBIC}(\langle \mathcal{G}', \gamma' \rangle) - \text{scoreBIC}(\langle \mathcal{G}, \gamma \rangle) =$$
$$l\mathcal{D}\ (\langle \mathcal{G}', \gamma' \rangle) - l\mathcal{D}(\langle \mathcal{G}, \gamma \rangle) - \frac{\log M}{2}(Dim(\mathcal{G}') - Dim(\mathcal{G})) =$$
$$l\mathcal{D}\ (\langle \mathcal{G}', \gamma' \rangle) - l\mathcal{D}(\langle \mathcal{G}, \gamma \rangle) - \frac{\log M}{2}(1) =$$
$$l\mathcal{D}\ (\langle \mathcal{G}', \gamma' \rangle) - l\mathcal{D}(\langle \mathcal{G}, \gamma \rangle) - \frac{\log M}{2}$$

$\mathcal{G}'$ è accettata se :

$$l\mathcal{D}\ (\langle \mathcal{G}', \gamma' \rangle) - l\mathcal{D}(\langle \mathcal{G}, \gamma \rangle) > \frac{\log M}{2}$$

Per tenere aggiornato il likelihood della cutset network sarà necessario solamente rivalutare il likelihood locale e sommarlo a quello della rete restante, sfruttando la decomponibilità del likelihood. La decomposizione di un albero $\mathcal{T}\updownarrow$ è indipendente da quella di un altro albero $\mathcal{T}\|, k \neq l$ dato che le loro contribuzioni al likelihood globale sono indipendenti; quindi non è importante l'ordine con cui decomporre le foglie.

Come criterio di terminazione si utilizza il numero minimo di istanze del training set rimaste su cui splittare e/o il numero minimo di features rimaste.

L'algoritmo in pseudocodice è riportato nell'algoritmo 2. La prima operazione è quindi l'apprendimento di un albero di Chow-Liu(riga 2). Successivamente si chiama l'algoritmo di decomposizione (algoritmo 3), passando in input l'albero appena creato.

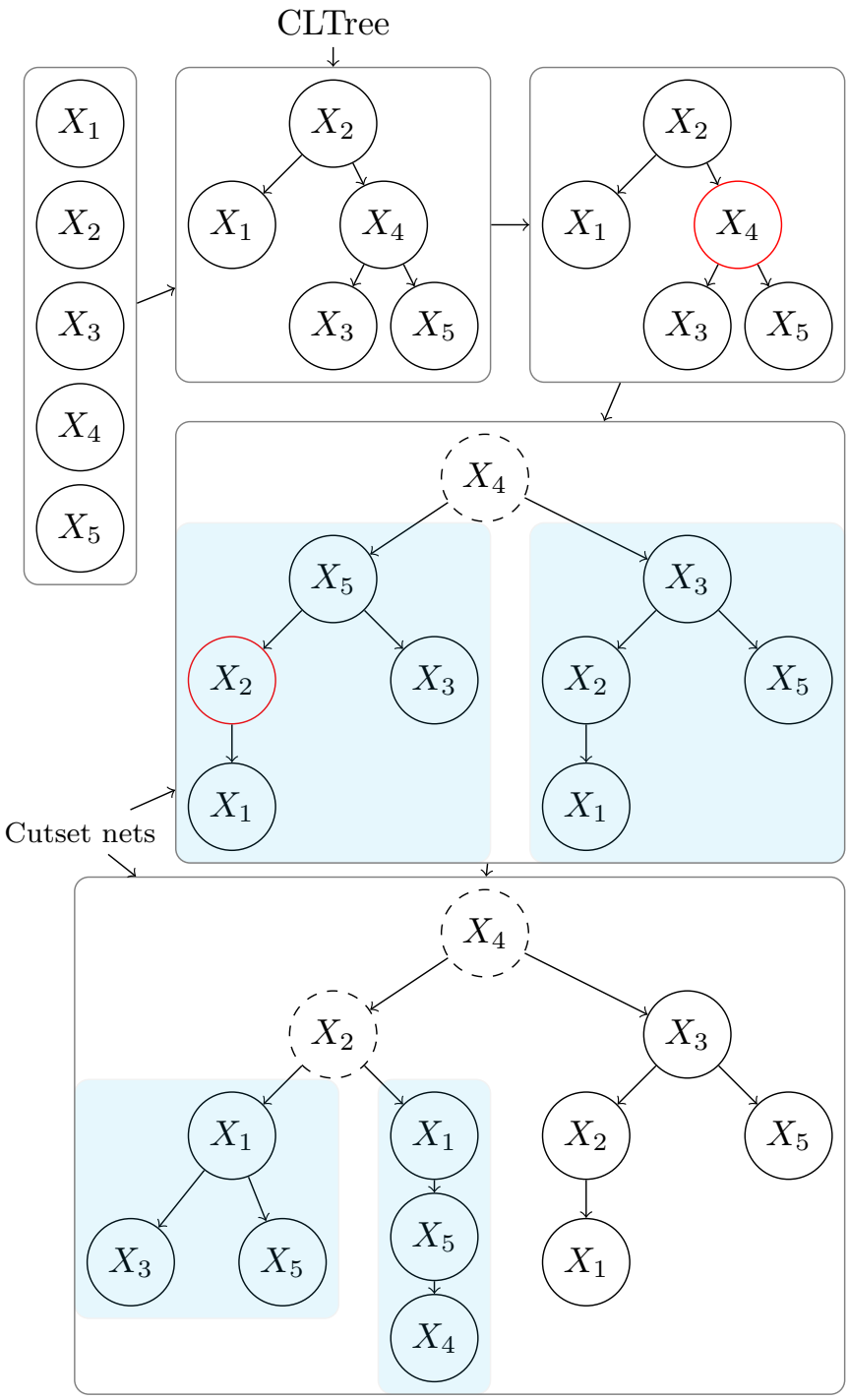


Figura 8: Alcuni passi di dcsn. I nodi rossi rappresentano la variabili per cui la decomposizione produce un aumento nel likelihood. I nodi tratteggiati rappresentano nodi OR. La cutset network finale, in questo caso, contiene 2 nodi OR (X2 e X4) e 3 alberi di Chow-Liu come foglie. Ogni variabile può assumere solo 2 valori.

Algorithm 2 dCSN($\mathcal{D}$,X,αf,δ, σ)

Input: : il learning set $\mathcal{D}$,il set di variabili X, $\alpha f \in [0,1]$, δ numero minimo di istanze su cui decomporre, σ minimo numero di features su cui decomporre

Output: : una cutset network $\langle \mathcal{G}, \gamma \rangle$ appresa su X attraverso $\mathcal{D}$

1: $\alpha = \alpha f |\mathcal{D}|$
2: $\langle \mathcal{T}, \theta \rangle$ = Learn Chow-Liu tree($\mathcal{D}$,X,α)
3: $w = 0$
4: $\langle \mathcal{G}, \gamma \rangle$ = decompose($\mathcal{D}$,X,$\alpha, \mathcal{T}, \theta, w, \delta, \sigma$)

Alla riga 1 vengono verificate le condizioni per poter tentare di decomporre la struttura; ovvero la presenza di almeno δ istanze e di σ features. In caso positivo si inizia a testare la decomposizione usando ogni variabile. Generato un nodo OR con la variabile Xi, per ogni valore del dominio di Xi si genera un arco e si apprende un albero di Chow-Liu con il rispettivo slice del training set. Dopo aver testato tutte le features si cerca di ottenere una decomposizione utilizzando la miglior variabile trovata che generi una nuova struttura avente log-likelihood migliore di quello corrente. In caso positivo (riga 14) si tenta di decomporre, in maniera ricosriva, i nuovi alberi di Chow-Liu della nuova cutset network (riga 19).
L'algoritmo termina quando almeno uno dei due criteri di terminazione è soddisfatto.

Algorithm 3 decompose($\mathcal{D}$,X,$\alpha, \mathcal{T}, \theta, w, \delta, \sigma$)

Input: : il learning set $\mathcal{D}$,il set di variabili X, $\alpha f \in [0,1], \delta$ numero minimo di istanze su cui decomporre, σ minimo numero di features su cui decomporre, $\mathcal{T}$ l'albero da decomporre e i suoi parametri θ

Output: : una cutset network $\langle \mathcal{G}, \gamma \rangle$ appresa su X attraverso $\mathcal{D}$

if $|\mathcal{D}| > \delta$ and $|X| > \sigma$ **then**
  $l$best = -∞
  for Xi$\in$X **do**
    $\mathcal{G}$i=0,wi=0, θi=0,Ci nodo OR associato a Xi
    for x^{j}i$\in$ Val(Xi) **do**
      $\mathcal{D}$j=$\{ \xi \in \mathcal{D} : \xi[Xs] = xs^{\text{j}} \}$
      $w$ij=$|\mathcal{D}|/|\mathcal{D}|$
      $\langle \mathcal{T}|, \theta ij \rangle$ = Learn Chow-Liu Tree($\mathcal{D}$|,X$\backslash s$,αwij)
      $\mathcal{G}\rangle$ = aggiungiSottoAlberi(Ci,$\mathcal{T}$|)
      $wi = wi \cup \{wij\}, \theta i = \theta i \cup \{\theta ij\}$
    $li = l$Di($\langle \mathcal{G} \rangle$,$wi \cup \theta i \rangle$)
    if $li > lbest$ and $li > l$Di($\langle \mathcal{T}, \theta \rangle$) **then**
      $lbest = li$, Xbest=Xi,$\mathcal{G}\lfloor\rceil \int \sqcup = \mathcal{G}\rangle, \theta \lfloor\rceil \int \sqcup = \theta\rangle$,$w$best=$w$i
  if $l$best - $l\mathcal{D}(\langle \mathcal{T}, \theta \rangle) > (log|\mathcal{D}|)/2$ **then**
    sostituisci $\mathcal{T}$ con $\mathcal{G}\rangle$
    $w = w \cup wbest$
    for x^{j}b$\in$Val(xbest) **do**
      $\mathcal{D}| = \{\xi \in \mathcal{D} : \xi[Xbest]\}$
      decompose($\mathcal{D}$|,X$\backslash best$,$\alpha wij, \mathcal{T}|, \theta j, w, \delta, \sigma$)

3 Tecniche di Stochastic Local Search

Gli argomenti cuore di questa tesi saranno descritti in questa sezione. L'obiettivo di questa sezione è quello di presentare algoritmi stocastici di ricerca locale (sez. 3.1 e 3.2). In particolare saranno presentati nel dettaglio: Iterative improvement (sez. 3.3), Randomised iterative improvement (sez. 3.4) e Greedy randomised 'adaptive' search procedure (GRASP) (sez. 3.5).

3.1 Ricerca locale e problemi combinatoriali

Prima di descrivere ed affrontare la tipologia di problemi discussi in questo capitolo è utile descrivere un problema in termini di spazio di input, spazio delle soluzioni e spazio di output.
Si definisce lo *spazio di input* l'insieme dei valori assumibili dalle variabili di input del problema. Inoltre si definisce lo *spazio delle soluzioni* come l'insieme dei valori, per alcune variabili del problema, che costituiscano una soluzione del problema. Lo *spazio di output* costituisce l'insieme delle informazioni in uscita si vuole che il processo risolutivo produca.

In natura esistono diverse tipologie di problemi. Una tipologia di problemi per cui trovare la soluzione ottima non è un compito immediato è rappresentato dai *problemi combinatoriali*.
I problemi combinatoriali si presentano in molte aree come l'intelligenza artificiale, ricerca operativa, bioinformatica e commercio elettronico. La loro caratteristica comune è la crescita esponenziale del numero di soluzioni possibili, per una certa istanza di problema, all'aumentare della dimensione della specifica istanza. Questo fa sì che un approccio esaustivo nella ricerca di una soluzione accettabile non sia fattibile.

Un esempio di problema combinatoriale è quello di trovare il percorso più breve o meno costoso in un grafo (problema del commesso viaggiatore). Un altro è quello di trovare il minimo albero ricoprente di un grafo (Minimum Spanning Tree). Altri problemi combinatoriali si incontrano in pianificazione, scheduling, allocazione di risorse, code design, hardware design e nel sequenzionamento del genoma.
Questo tipo di problemi di solito richiede il raggruppamento, l'ordinamento o l'assegnazione di insiemi finiti di oggetti che soddisfano certe condizioni. Combinazioni di queste componenti formano potenziali soluzioni finali.

Spesso ci interessa trovare la miglior soluzione dello spazio delle soluzioni, spesso determinarla in maniera diretta è quasi impossibile. In questo tipo di problemi ci si limita a trovare una soluzione ragionevolmente buona cercando di ottimizzare soluzioni iniziali non ottime. Questo tipo di problema appartiene alla categoria dei problemi di *ottimizzazione*.

Nei problemi di ottimizzazione si utilizza una funzione detta **funzione obiettivo**, il cui compito è di associare ad un elemento dello spazio delle soluzioni un valore che è direttamente collegato alla qualità della soluzione. Quindi per ottimizzare la soluzione di un problema ci si pone l'obiettivo di utilizzare algoritmi che minimizzino o massimizzino il più possibile la funzione obiettivo partendo da una soluzione iniziale.

La crescente complessità dei problemi combinatoriali ci spinge ad assumere approcci risolutivi che possono essere caratterizzati come *algoritmi di ricerca*. L'idea fondamentale dietro l'approccio di ricerca è di generare e valutare iterativamente delle soluzioni candidate; dove con "valutare" s'intende stabilire il corrispettivo valore della funzione obiettivo. Questo è possibile grazie al fatto che, pur essendo i problemi combinatoriali di complessità elevata, la valutazione delle loro soluzioni candidate è spesso possibile più efficientemente in tempo polinomiale.
Quindi per "ricerca" s'intende la ricerca di una soluzione, secondo un certo criterio, nello *spazio delle possibili soluzioni*.

Si può immaginare lo spazio delle possibili soluzioni come un grafo in cui ogni nodo rappresenta una possibile soluzione ed ogni arco, collegante due soluzioni, rappresenti un "passo" nello spazio di ricerca. Si può immagi-

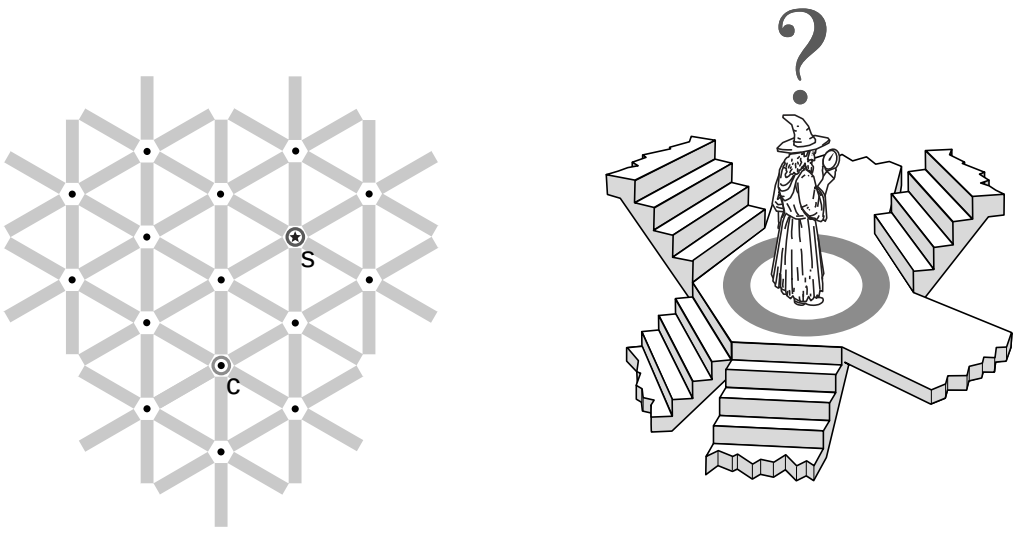


Figura 9: Presa da [5]

nare una soluzione candidata come il nodo etichettato con "C" nella figura 9. Il compito di un algoritmo di ricerca è quello di cercar di raggiungere o comunque avvicinarsi il più possibile alla soluzione ottimalmente globale, rappresentata dal nodo "S". Una volta posizionati in una soluzione non ancora ottima, gli algoritmi di ricerca adottano diverse tecniche per poter de-

cidere in che direzione andare. In base a queste tecniche essi sono suddivisi in diverse categorie: a seconda di come rappresentano le soluzioni, a seconda di come le generano e a seconda dell'uso che fanno delle informazioni della soluzione corrente.

3.1.1 Paradigmi di ricerca

Gli approcci di ricerca ricadono in diversi paradigmi, ovvero diversi sotto-approcci con cui gli algoritmi di ricerca generano soluzioni candidate.

Gli algoritmi di ricerca presenti in questa sezione appartengono al paradigma *perturbativo*. Questo paradigma sfrutta la proprietà delle soluzioni per cui, esse sono formate a loro volta da un insieme di *componenti*. Ad esempio una soluzione per il problema del commesso viaggiatore è composta da componenti che rappresentano i singoli archi che compongono il percorso finale.
Trattando le soluzioni come insiemi di componenti, gli algoritmi perturbativi generano facilmente nuove soluzioni modificando una o più componenti di una soluzione avente a disposizione.

Un altro paradigma è quello *generativo*, dove si sceglie di generare una soluzione componente per componente. Questo tipo di ricerca ha luogo nello spazio di ricerca includendo anche le soluzioni *parziali*, ovvero, soluzioni per cui alcune componenti sono mancanti.
La generazione di nuove soluzioni è quindi ottenuta estendendo iterativamente le soluzioni candidate migliorando la funzione obiettivo.

Altri due tipi di paradigmi sono quello *sistematico* e quello *locale*.
Gli algoritmi di ricerca sistematica sono algoritmi che attraversano lo spazio di ricerca in maniera completa, garantendo il ritrovamento della soluzione ottimale (se presente). Gli algoritmi di ricerca locale, d'altra parte, partono ad una determinata posizione dello spazio di ricerca e successivamente si muovono dalla posizione corrente verso posizioni vicine basandosi esclusivamente su conoscenza *locale*. Codesti algoritmi sono quindi incompleti, ovvero, non garantiscono il ritrovamento di una soluzione ottima (se presente) dato che possono non visitare tutto lo spazio di ricerca o addirittura visitare la stessa posizione più di una volta .

Quindi, perché utilizzare la ricerca locale? Un motivo è rappresentato dalla natura costruttiva di molti problemi. In queste situazioni è chiaro che l'obiettivo di un algoritmo risolutivo è quello di generare una soluzione e non cercarne una nello spazio di tutte le possibili soluzioni. Un altro motivo è rappresentato dal tempo di risoluzione. Quasi tutti i problemi del mondo reale sono legati indissolubilmente al tempo. La ricerca esaustiva attuata dagli

algoritmi sistematici è quasi sempre impraticabile dato che visitare l'intero spazio delle soluzioni per trovare una soluzione ottima impiegherebbe un tempo di gran lunga superiore a quello richiesto dalla tipologia di applicazione. Gli algoritmi di ricerca locale, invece, sono capaci di trovare comunque una soluzione in tempi brevi. La loro natura iterativa permette il ritrovamento di soluzioni man mano migliori, garantendo una risposta in tempi brevi al contrario degli algoritmi sistematici che potrebbero abortire la ricerca dopo il quanto di tempo avuto a disposizione. Idealmente algoritmi per problemi real-time dovrebbero essere in grado di dare una buona soluzione a qualsiasi punto della loro esecuzione.

3.2 Ricerca stocastica

Molti algoritmi di ricerca locale fanno uso di scelte randomiche nella generazione e selezione di soluzioni candidate. Questi algoritmi sono chiamati *algoritmi stocastici di ricerca locale (stochastic local search - SLS) [5].*

Come per gli algoritmi di ricerca locale normale, essi selezionano una soluzione iniziale e procedono iterativamente verso soluzioni "vicine" nello spazio delle soluzioni basandosi su conoscenza limitata e locale. La differenza chiave è che gli algoritmi stocastici fanno uso di randomicità, includendola nel processo di generazione iniziale della soluzione e nel processo di decisione della soluzione vicina verso cui muoversi.

Definizione: Algoritmo stocastico di ricerca locale. Dato un problema (combinatoriale) Π, un algoritmo stocastico di ricerca locale per la risoluzione di un'arbitraria istanza $\pi \in \Pi$ è definito dalle seguenti componenti:

- lo spazio di ricerca S(π) dell'istanza π, ovvero l'insieme di soluzioni candidate.
- un insieme di soluzioni attuabili S'(π)$\subseteq$ S(π)
- una funzione neighbourhood su S(π), N(π)$\subseteq$ S(π) x S(π), atta ad identificare le soluzioni "vicine" nello spazio delle soluzioni
- un insieme finito di stati di memoria M(π), per algoritmi facenti uso di uno storico degli stati
- una funzione di inizializzazione init(π): $0 \rightarrow \mathcal{D}$(S($\pi$) x M($\pi$)), che specifica una distribuzione di probabilità sulle posizioni iniziali

- una funzione scalino step(π): S(π) x M(π)$\rightarrow$ $\mathcal{D}$(S(π) x M(π)), che fa un mapping tra le posizioni e gli stati di memoria in una distribuzione di probabilità sulle posizioni vicine e sugli stati di memoria

- un predicato di terminazione
terminate(π): S(π) x M(π) $\rightarrow$ $\mathcal{D}$(true,false), che fa il mapping da ogni soluzione e stato di memoria a una distribuzione di probabilità sui valori booleani indicando la probabilità di terminazione della ricerca

Ogni relazione di vicinanza N(π) può essere trasformata equivalentemente in una funzione Neighbourhood: S(π) $\rightarrow$ $2^{S(\pi)}$ che si occupa di fare il mapping tra una soluzione candidata s$\in$S e il suo insieme di soluzioni direttamente "vicine" N(s) := {s'$\in$S | N(s,s')} $\subseteq$S.
L'insieme N(s) è chiamato *vicinato (Neighbourhood)* . In generale la scelta della relazione di vicinanza tra soluzioni è cruciale per la performance dell'algoritmo SLS. Fra le diverse relazioni più popolari si trova il *k-exchange neighbourhoods*, che afferma che due soluzioni candidate sono vicine se, e solo se, esse differiscono di almeno k componenti.

Avendo dato informazioni di base e sufficienti sugli algoritmi stocastici di ricerca locale, si procede con la descrizione degli algoritmi utilizzati nel lavoro di tesi.

3.3 Iterative improvement

Il più semplice algoritmo stocastico di ricerca locale di questa tesi è *Iterative improvement*. Dato uno spazio di ricerca S, una relazione di vicinato N e una funzione obiettivo g, Iterative Improvement parte da una soluzione random dello spazio di ricerca e cerca di migliorare la soluzione candidata corrente rispetto alla funzione g. La versione in pseudocodice è rappresentata dall'algoritmo 4.

Algorithm 4 Iterative improvement

1: Determina una soluzione candidata iniziale s
2: **while** s non è un ottimo locale **do**
3: Scegli una soluzione s' nel vicinato di s tale che g(s') sia migliore di g(s)
4: s:=s'

L'approccio utilizzato da Iterative Improvement prevede una scansione dei vicini di una soluzione e successivamente una valutazione della qualità

di ognuno di essi. Alla fase di valutazione segue la fase in cui si sceglie la soluzione vicina che abbia il più alto valore della funzione obiettivo. Infine ci si sposta su di essa se il suo valore della funzione obiettivo è migliore del valore della funzione obiettivo calcolata sulla soluzione corrente. Questo ciclo viene fatto fino a quando si hanno dei miglioramenti.
Per implementare efficientemente iterative improvement, i valori della funzione obiettivo sono tipicamente trattati usando *aggiornamenti incrementali* dopo ogni ciclo di ricerca. In molti problemi il valore della funzione obiettivo è formato dai contributi indipendenti delle componenti delle soluzioni. Quindi, per calcolare gli effetti della soluzione s' sulla funzione obiettivo, ci si limita a considerare il contributo delle componenti non in comune tra s e s'.

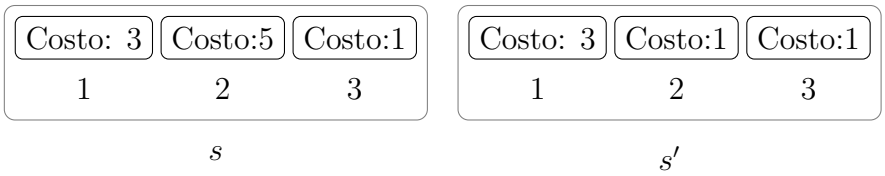


Figura 10: In questa figura le soluzioni s e s' differiscono solamente per la componente 2. Quindi per capire se conviene scegliere s' al posto di s è efficiente misurare solamente i costi delle componenti non in comune. In questo caso la componente 2 di s' ha un costo minore rispetto alla componente 2 di s; essendo le componenti indipendenti il cambiamento di costo della componente 2 non influenza quello delle altre componenti 1 e 3.

Quindi nella figura 10 dopo aver constatato che il costo della soluzione s' è minore rispetto a quello corrente, iterative improvement si sposta verso s'.

Iterative improvement può sfruttare due tipi di approcci nella scelta della posizione verso cui dirigersi nello spazio di ricerca. Questi due tipi differiscono soprattutto nell'efficienza.

Uno di questi è *Iterative Best Improvement* (anche chiamato Greedy Hill Climbing), basato sull'idea di scegliere il vicino che causi il maggior miglioramento della funzione obiettivo. Questa tipologia prevede, quindi, una completa valutazione di tutte le soluzioni del vicinato in ogni ciclo di ricerca e la scelta del miglior vicino.

L'altro è *First Improvement*, dove la selezione di una soluzione vicina tenta di evitare la complessità di valutare tutti i vicini scegliendo la *prima* soluzione che migliora il valore della funzione obiettivo.

Minimo locale e problemi annessi. In Iterative Improvement la strategia di scelta della soluzione verso cui muoversi non è ben adatta nei casi

in cui le soluzioni candidate non abbiano vicini che migliorino il valore della funzione obiettivo. Una soluzione candidata avente questa proprietà è chiamata *minimo locale*, e la sua definizione formale segue:

Definizione: Minimo locale. Sia S uno spazio di ricerca, S' $\subseteq$ S un insieme di soluzioni, N $\subseteq$ S x S una relazione di vicinato e g: S $\rightarrow \mathbb{R}$ una funzione obiettivo, una soluzione candidata s $\in$ S è un minimo locale se $\forall s' \in N(s), g(s) \leq g(s')$ (con $a \leq b$ rappresentante a soluzione migliore rispetto a b). Un minimo locale è detto *strettamente minimo locale* se $\forall s' \in N(s), g(s) < g(s')$.

Intuitivamente un minimo locale è una posizione nello spazio di ricerca dove nessun passo può portare al miglioramento della funzione obiettivo. In questi casi, quando un algoritmo incontra un minimo locale rispetto ad una certa funzione obiettivo, esso rimane "intrappolato".

In letteratura esistono diversi approcci atti ad evadere da queste situazioni. Molti si servono dell'utilizzo di randomicità, includendola nelle fasi di decisione e valutazione. Un algoritmo che sfrutta questo approccio e che è direttamente collegato ad iterative improvement è Randomised Iterative Improvement.

3.4 Randomised iterative improvement

Una delle più semplici vie per estendere iterative improvement per evadere da minimi locali è quello di includere randomicità nella scelta della prossima soluzione. Quindi, invece di scegliere un vicino che migliori la qualità della soluzione corrente, si sceglie un vicino random.

La frequenza con cui scegliere un vicino in maniera random può essere determinata dal programmatore. Quindi ad ogni ciclo di ricerca è preferibile determinare in maniera probabilistica se applicare un passo di iterative improvement oppure applicare un passo di scelta randomica.

Questo viene fatto introducendo un parametro $p \in [0, 1]$, chiamato *walk probability o noise parameter*, che corrisponde alla probabilità di effettuare una scelta random rispetto ad una scelta non-random.

L'introduzione di questa scelta dà vita a *Randomised Iterative Improvement (RII)*.

Quindi un effetto benefico della scelta random è che nel caso in cui si incorra in un ottimo locale, RII permette di evadere anche da ottimi locali aventi un grande "bacino di attrazione" nel senso che molti passi peggiorativi

Algorithm 5 Randomised Iterative improvement(π,s,p)

Input: π: l'istanza del problema, s una soluzione iniziale, p il noise parameter
Output: una soluzione candidata s'

```
1: u:= random([0,1])
2: if  u ≤ p then
3:     s':= scelta_random(π,s)
4: else
5:     s':= scelta_iterative_improvement(π,s)
   return return s'
```

possano essere richiesti per assicurare che i successivi miglioramenti abbiano la chance di portare al raggiungimento di un differente ottimo locale.

Inoltre è stato dimostrato che, quando la procedura di ricerca è eseguita tanto abbastanza, RII permette il raggiungimento di un'eventuale soluzione ottimale per ogni tipologia di problema e per arbitarie alte probabilità.

L'algoritmo 5 presenta una versione in pseudocodice di RII. Si può notare come l'algoritmo verifichi la possibilità di effettuare un passo random nello spazio di ricerca valutando il valore della variabile u e confrontandolo con p che rappresenta la probabilità di effettuare una scelta random. In caso contrario la soluzione parziale *s'* è scelta attraverso Iterative improvement.

3.5 Greedy Randomised Adaptive Search Procedures

Una famiglia di algoritmi stocastici di ricerca locale è rappresentata da tutti quegli algoritmi i cui approcci come risultato di combinazioni di approcci più semplici. La combinazione di due o più approcci caratterizza la famiglia degli *algoritmi stocastici di ricerca locale ibridi*.

Il comportamento e le performance degli algoritmi stocastici "semplici" possono essere spesso migliorati significativamente combinando diverse strategie.

Un approccio standard per trovare velocemente delle soluzioni di alta qualità è quello di applicare metodi greedy di ricerca alle fasi di costruzione degli algoritmi. Questi metodi, ad ogni step costruttivo, aggiungono una componente solutiva scelta come migliore in un insieme di altre componenti utilizzando una funzione euristica di selezione. Successivamente utilizzano ricerca locale perturbativa per migliorare la qualità della soluzione appena costruita.

Un difetto comune di questi metodi è che la fase costruttiva greedy genera solamente un numero limitato di soluzioni candidate.

Per evitare questo difetto le *Greedy Randomised Adaptive Search Procedures (GRASP)* includono la randomizzazione nel metodo di costruzione in modo tale da generare molte più potenziali buone soluzioni da poter poi passare ad un metodo di ricerca locale.

Un algoritmo di base di GRASP è riportato nell'algoritmo 6.

Algorithm 6 GRASP(π)

Input: π: l'istanza del problema
Output: una soluzione candidata s' oppure 0

```
 1: s := 0
 2: ŝ := s
 3: f(ŝ) := ∞
 4: while not (terminate(π,s)) do
 5:     s := costruisci(π)
 6:     s' := ricercaLocale(π,s)
 7:     if f(s') < f(s') then
 8:         ŝ := s'
 9: if ŝ ∈S' then
10:     return ŝ
11: else
12:     return 0
```

In GRASP in ogni iterata si genera una soluzione candidata tramite una procedura di ricerca costruttiva randomizzata (costruisci() - riga 5). Successivamente si applica un algoritmo di ricerca locale alla soluzione appena generata. Se alla fine della ricerca si è trovata una soluzione migliore rispetto alla soluzione corrente allora si aggiorna s e si ripete il ciclo fino a quando un criterio di terminazione viene verificato.

L'algoritmo di costruzione usato in GRASP, invece che scegliere man mano la miglior componente solutiva rispetto ad una determinata euristica, seleziona randomicamente una componente all'interno di un set di componenti aventi comunque alta qualità. Questo set di componenti è chiamato *Restricted Candidate List (RCL)*. Quindi, per ogni passo costruttivo GRASP genera una RCL di componenti e ne sceglie una in maniera random secondo una distribuzione uniforme di probabilità. La RCL può essere generata secondo due meccanismi: *restrizione di cardinalità o restrizione di valore*. Nel caso della restrizione di cardinalità la RCL contiene le migliori k componenti. Nel caso della restrizione di valore la RCL contiene le componenti la cui qualità supera un certo valore threshold.

Effettuate tutte le fasi costruttive, la procedura *costruisci* fornisce una soluzione che rappresenta la soluzione iniziale per l'algoritmo di ricerca locale

(riga 6). L'algoritmo di ricerca locale cerca di ottimizzare questa soluzione. Nella riga 7 si verifica se la soluzione appena trovata è migliore della miglior soluzione precedentemente trovata. In caso positivo si aggiorna $\hat{s}$.

GRASP è 'adattivo' perché di solito, nel passo costruttivo, il valore euristico di ogni componente solutiva dipende da quello delle componenti già facenti parte della soluzione parziale in costruzione.

In generale, comunque, non è garantito che le soluzioni candidate create dal passo costruttivo randomizzato siano degli ottimi locali rispetto a delle semplici soluzioni di un vicinato.

4 Stochastic Local Search per Cutset Network

Gli algoritmi presentati nella precedente sezione sono stati applicati per migliorare ed ottimizzare le soluzioni del problema dell'apprendimento di cutset networks. Questa sezione ha il compito di descrivere, per ogni algoritmo, la sua applicazione al problema specifico. Prima di descrivere gli algoritmi viene data una motivazione ed alcune osservazioni riguardo le modifiche all'algoritmo originale dCSN (sez. 4.1 e 4.2).

4.1 Apprendimento di foreste

Come descritto nella sezione 2.2.7, l'algoritmo di apprendimento di una cutset network, basato sulla decomponibilità, apprende degli alberi di Chow-Liu alle foglie della cutset network.

Una nuova modifica applicata all'algoritmo originale è stata quella di apprendere delle foreste invece che semplici alberi.
Una foresta non è altro che un insieme di alberi i cui insiemi di vertici sono disgiunti. Formalmente:

Definizione: Foresta di Chow-Liu
f è una foresta se:
- $f = \{\mathcal{T}\infty, .., \mathcal{T}\|\}$ con t albero di Chow-Liu $\forall t \in \{\mathcal{T}\infty, .., \mathcal{T}\|\}$
- $\forall t_i, t_j \in \{\mathcal{T}\infty, .., \mathcal{T}\|\}$, $V(t_i) \cap V(t_j) = \emptyset$, con $V(x)$ insieme dei nodi dell'albero x
- $\bigcup_{i=1}^{k} V(t_i) = V$ con V l'insieme originale delle variabili aleatorie

Una foresta avente 1 solo albero corrisponde ad un albero di Chow-Liu.

La decisione è nata dal problema che non sempre la distribuzione di probabilità che si sta cercando di approssimare è un albero come approssimata dell'algoritmo di Chow-Liu. Un esempio grafico di foresta è riportato nella figura 11 .

La figura 11 evidenzia il caso in cui la struttura migliore per approssimare una data distribuzione possa essere codificabile attraverso una foresta e non un albero. In particolare la scelta dell'albero obbliga la connessione di variabili che magari non siano veramente correlate. Se avessimo scelto di utilizzare un albero di Chow-Liu sicuramente la variabile X_7 sarebbe stata

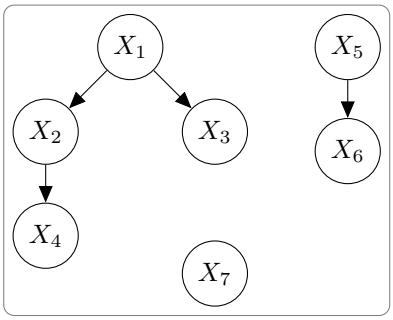


Figura 11: La figura rappresenta una foresta contenente 3 alberi. La struttura potrebbe evidenziare il fatto che la variabile X_7 non dipenda da nessun'altra variabile e quindi sia essa stessa un albero di Chow-Liu.

collegata ad un'altra variabile portando così la struttura ad approssimare male la distribuzione.

Per poter applicare gli algoritmi presentati nella sezione 3 c'è bisogno di inquadrare la tipologia di problema da affrontare e la definizione di "soluzione" del problema.

Il problema da affrontare nell'apprendimento di foreste di Chow-Liu è quello di approssimare il più possibile la distribuzione rappresentata dal training set, portando a classificare la tipologia di problema come problema di ottimizzazione.

Per il problema in questione si è definita "soluzione del problema", una foresta la cui unione degli insiemi di vertici degli alberi in essa contenuti è uguale all'insieme delle features presenti nello slices del learning set ad essa correlato. In particolare si è trattata una soluzione come insiemi di componenti, dove per componenti s'intendono gli archi in essa contenuti.
Come affermato nella sezione 3.1, due soluzioni sono differenti se differiscono per almeno una componente. Da questa definizione ne segue che, nel problema corrente, due soluzioni sono differenti se differiscono per almeno un arco. Si può quindi definire il "vicinato" N di una soluzione s come l'insieme delle foreste che differiscono di **un** solo arco rispetto ad s .

Come funzione obiettivo è stata scelta la likelihood (sez. 2.1.2), essendo direttamente collegata alla bontà di adattamento della foresta appresa rispetto alla densità obiettivo. Per il calcolo della likelihood è stato utilizzato un validation set, diverso quindi dal training set.

Tutti gli algoritmi sono stati applicati utilizzando come soluzione iniziale l'albero generato dall'algoritmo originale di Chow-Liu (sez. 2.2.2).

4.2 Aggiunta di rumore

Un'altra modifica applicata all'algoritmo originale dCSN è costituita dall'aggiunta di rumore alla matrice delle mutue informazioni.

In particolare si è fatto uso di rumore puramente additivo sulla matrice simmetrica dei valori delle informazioni mutue. Per evitare valori negativi nella matrice con aggiunta di rumore si è utilizzato il valore assoluto dei valori campionati dalla distribuzione normale standard.

Sia M la matrice delle mutue informazioni, N una matrice simmetrica avente in ogni cella un valore $|v| \in N(\mu, \sigma^2)$, MR è una matrice con aggiunta di rumore in valore assoluto se $MR = M + N$.

Inoltre sono stati utilizzati diversi valori di varianza (0.1, 0.5, 1,2) per gestire il grado di amplificazione della aggiunta di rumore.

4.3 Stochastic Local Search per l'apprendimento di Cutset Network

Gli algoritmi stocastici di ricerca locale sono stati utilizzati per migliorare gli alberi di Chow-Liu inizialmente appresi da dCSN. Una rivisitazione di iterative improvement è riportata in 7.

La variante utilizzata nel progetto è quella dell'Iterative Best Improvement, scegliendo tra le soluzioni vicine quella che migliora più di tutte la likelihood.

Parte del processo di miglioramento delle soluzioni di Iterative Improvement è quello di generazione del vicinato di soluzioni. Per il problema affrontato in questa tesi la funzione $generateNeighbourhood()$ ha avuto il compito di generare l'insieme dei vicini in termini di foreste. Data una soluzione s, s' appartiene al vicinato di s se $E(s) - E(s') = 1$ e s' è ottenuta da s "tagliando" un arco.
Se s è un albero di Chow-Liu allora s' è una foresta avente due alberi di Chow-Liu.
Se s è una foresta avente n alberi allora s' è una nuova foresta avente $n + 1$ alberi.
La foresta è, quindi, ottenuta eliminando un arco dall'albero originale. Così facendo si punta ad eliminare l'arco che maggiormente influenzava in maniera negativa il comportamento dell'albero originale.

Quindi in tutti gli algoritmi di ricerca locale applicati, la dimensione del vicinato di una soluzione è stata pari al numero di archi in essa contenuti. Ogni vicino di una soluzione s corrisponde ad un arco tagliato da s.

Successivamente iterative improvement misura, per ogni elemento del vicinato, il valore di likelihood raggiunto sul validation set. Seguendo il com-

Algorithm 7 Iterative improvement(cltree)

Input: cltree: albero di Chow-Liu
Output: una foresta o l'albero ricevuto in input

```
 1: improved = True
 2: best_ll = -∞
 3: best_forest= null
 4: current_best_solution = cltree
 5: current_best_ll = score(cltree)
 6: while improved = True do
 7:     improved = False
 8:     N= generateNeighbourhood(cltree)
 9:     for s' in N do
10:         log_likelihood= score(s')
11:         if log_likelihood > best_ll then
12:             best_ll = log_likelihood
13:             best_forest = s'
14:     if  best_ll > current_best_ll then
15:         current_best_solution = best_forest
16:         current_best_ll = best_ll
17: return current_best_solution
```

portamento di base dell'algoritmo di ricerca, si sceglie la miglior soluzione del vicinato.
Finito il ciclo si testa il miglior log-likelihood con il likelihood correntemente raggiunto dalla struttura. Se si sono avuti dei miglioramenti allora si sostituisce la nuova struttura con l'arco tagliato alla vecchia struttura. Il tutto viene ripetuto fino a quando si hanno dei miglioramenti.

Anche Randomised Iterative Improvement è stato rivisitato per adattarlo al problema specifico affrontato. L'algoritmo è riporato in 8. Le adattazioni attuate in Iterative Improvement sono state le stesse di Randomised Iterative Improvement.

Algorithm 8 Randomised Iterative improvement(cltree,probability,times)

Input: cltree: albero di Chow-Liu, probability: probabilità di scegliere il miglior vicino, times: numero di iterate (criterio di terminazione)
Output: una foresta o l'albero ricevuto in input

```
1: t = 0
2: best_ll = -∞
3: best_forest= null
4: current_best_solution = cltree
5: current_best_ll = score(cltree)
6: while t < times do
7:     N = generateNeighbourhood(cltree)
8:     r=random([0,1])
9:     if r > probability then
10:        s' = getRandom(N)
11:        best_ll = score(s')
12:        best_forest = s'
13:    else
14:        for s' in N do
15:            log_likelihood= score(s')
16:            if log_likelihood > best_ll then
17:                best_ll = log_likelihood
18:                best_forest = s'
19:    if  best_ll > current_best_ll then
20:        current_best_solution = best_forest
21:        current_best_ll = best_ll
22:        improved = True
23:    t = t + 1
24: return current_best_solution
```

Algorithm 9 GRASP(times, k,cltree)

Input: times: numero di iterate (criterio di terminazione), k: cardinalità della RCL, cltree: l'output dell'algoritmo di chow-liu

Output:

```
1: current_best_solution = cltree
2: current_best_ll = score(cltree)
3: t := 0
4: while t < times do
5:     //Inizio costruzione
6:     mst = minimum_spanning_tree(k) //versione modificata dell'algoritmo di kruskal
7:     s = attraversaAlbero(mst)
8:     //Fine costruzione
9:     //Inizio ricerca locale
10:    improved = True
11:    best_ll = -∞
12:    best_forest= null
13:    initial_ll = score(s)
14:    while improved = True do
15:        improved = False
16:        N= generateNeighbourhood(s)
17:        for s' in N do
18:            log_likelihood= score(s')
19:            if log_likelihood > best_ll then
20:                best_ll = log_likelihood
21:                best_forest = s'
22:        if best_ll > initial_ll then
23:            s = best_forest
24:            initial_ll = best_ll
25:    //Fine ricerca locale
26:    if initial_ll > current_best_ll then
27:        current_best_ll = initial_ll
28:        current_best_solution = s
29:    t = t + 1
```

L'algoritmo GRASP è riportato in pseudocodice nell'algoritmo 9. In GRASP si è utilizzata una versione modificata dell'algoritmo di kruskal. La differenza è che nell'algoritmo di kruskal in ogni ciclo si cerca di aggiungere l'arco con mutua informazione migliore all'albero in costruzione, mentre nella versione modificata si sceglie un arco *random* tra i migliori k archi (ordinati in base alla mutua informazione che rappresentano).

Questa modifica è stata fatta per permettere la creazione della RCL (Restricted Candidate List), componente fondamentale dell'approccio GRASP. In questo adattamento la RCL è composta dai migliori k archi, appartenendo quindi alla tipologia di RCL con restrizione di cardinalità.

GRASP, quindi, costruisce l'albero di Chow-Liu iniziale per una foglia tramite l'algoritmo modificato di kruskal. Dopo la fase di costruzione randomizzata, si dà inizio alla fase di ricerca locale. Benchè teoricamente in GRASP si possa utilizzare qualsiasi algoritmo di ricerca locale, nel progetto è stato utilizzato Iterative Best Improvement. Successivamente testa il likelihood nuovo con quello vecchio e, come in Iterative Improvement e Randomised Iterative Improvement, sostituisce la nuova struttura con la vecchia.

Come criterio di terminazione è stato scelto un parametro chiamato "times" rappresentante il numero di cicli costruzione-ricerca effettuati dall'algoritmo.

5 Sperimentazioni

Alla fase di adattamento ed implementazione degli algoritmi di ricerca al problema dell'apprendimento delle cutset networks è seguita una fase di sperimentazione. La fase di sperimentazione è stata utilizzata per valutare le eventuali migliorie prodotte dall'applicazione delle modifiche a dCSN e degli algoritmi presentati nella sezione 3.

In questa sezione saranno presentati i datasets utilizzati (sez. 5.1). Alcuni dettagli implementativi (sez. 5.2), i risultati (sez 5.3) e la loro relativa discussione (sez 5.4).

5.1 Datasets

Nella fase di sperimentazione sono stati utilizzati 7 datasets. Questi datasets sono alcuni dei datsets standard per il benchmark degli algoritmi di apprendimento di PGM presentati in [11] e [12]. I datasets hanno un numero di istanze di training che va da 9000 a 291326. Il numero di features in esse contenute cade in un range di 16 - 111 unità.

	$\|X\|$	$\|T_{train}\|$	$\|T_{val}\|$	$\|T_{test}\|$
NLTCS	16	16181	2157	3236
MSNBC	17	291326	38843	58265
Plants	69	17412	2321	3482
Audio	100	15000	2000	3000
Jester	100	9000	1000	4116
Netflix	100	15000	2000	3000
Accidents	111	12758	1700	2551

Tabella 1: Datasets utilizzati

Tutti i datasets contengono variabili binare, ovvero, variabili che possono assumere valori {0,1}.
Si utilizzano gli stessi split per training, validation (10%), e test set (15%) individuati originariamente dagli autori.
Da qui segue che ogni nodo OR delle strutture apprese ha avuto 2 nodi figli: uno per le istanze aventi la feature avvalorata a 0 e un altro per le istanze aventi la feature avvalorata a 1.

5.2 Dettagli implementativi

Il lavoro svolto da Di Mauro at al. in [3] è stato utilizzato come base per l'applicazione degli algoritmi e delle varianti precedentemente discusse. Il codice originale è disponibile su Github[1].

Gli algoritmi di ricerca, così come l'intero progetto originale, sono stati sviluppati in Python 3.5 con l'ausilio delle librerie SciPy, NumPy, Numba e sklearn. Per poter implementare la variante dell'algoritmo di kruskal presentata nella sezione 4.3 è stata utilizzata, come base, l'implementazione presente nella libreria SciPy.

L'intero progetto presentato in questa tesi è anch'esso disponibile su Github[2]. Gli script python sono stati parametrizzati per facilitare i vari test e per automatizzare le grid searches. Gli script hanno prodotto risultati in forma testuale e tabellare così come nel progetto originale.

5.3 Risultati

Ogni tipologia di algoritmo di ricerca è stata applicata ad ogni dataset, e per ogni dataset è stata fatta una grid search sui seguenti parametri:

- $\alpha f \in \{0.5, 1.0, 5.0, 10.0\}$
- $\delta \in \{10, 50, 100, 200, 500\}$
- $\sigma^2 \in \{0.1, 0.5, 1, 2\}$ solo per le varianti con aggiunta di rumore
- $\sigma \in \{3\}$

Per ogni tipologia di algoritmo è stata testata la versione normale e la versione con aggiunta di rumore alla matrice delle mutue informazioni.

I risultati numerici delle esecuzioni degli algoritmi sono stati prodotti dagli script python e salvati in alcuni file di log. Questi file di log sono stati trasformati in tabelle. Dalle tabelle prodotte sono state estratte le colonne contenenti i valori dei parametri passati in input. Queste tabelle sono di seguito riportate raggruppate per dataset.

[1] https://github.com/nicoladimauro/dcsn
[2] https://github.com/Rhuax/dcsn

5.3.1 nltcs

Tabella 2: dataset:nltcs, nessun algoritmo di SLS applicato, senza rumore

alpha	mnist	test_ll
0.50000	10	-6.49890
0.50000	50	-6.33067
0.50000	100	-6.21828
0.50000	200	-6.15773
0.50000	500	-6.08355
1.00000	10	-6.43014
1.00000	50	-6.30230
1.00000	100	-6.18094
1.00000	200	-6.13457
1.00000	500	-6.06773
5.00000	10	-6.30488
5.00000	50	-6.20013
5.00000	100	-6.14045
5.00000	200	-6.10698
5.00000	500	-6.05411
10.00000	10	-6.25948
10.00000	50	-6.18628
10.00000	100	-6.15030
10.00000	200	-6.10845
10.00000	500	-6.07792

Tabella 3: dataset: nltcs, nessun algoritmo di SLS applicato, varianza del rumore: 0.1

alpha	mnist	test_ll
0.50000	10	-6.45926
0.50000	50	-6.27106
0.50000	100	-6.20831
0.50000	200	-6.13424
0.50000	500	-6.09508
1.00000	10	-6.40590
1.00000	50	-6.23987
1.00000	100	-6.19141
1.00000	200	-6.12428
1.00000	500	-6.09071

5.00000	10	-6.25544
5.00000	50	-6.16408
5.00000	100	-6.14874
5.00000	200	-6.06335
5.00000	500	-6.07834
10.00000	10	-6.23529
10.00000	50	-6.14924
10.00000	100	-6.11765
10.00000	200	-6.10998
10.00000	500	-6.08494

Tabella 4: dataset: nltcs, nessun algoritmo di SLS applicato, varianza del rumore: 0.5

alpha	mnist	test_ll
0.50000	10	-6.39648
0.50000	50	-6.25413
0.50000	100	-6.16192
0.50000	200	-6.11764
0.50000	500	-6.10513
1.00000	10	-6.40243
1.00000	50	-6.20579
1.00000	100	-6.15182
1.00000	200	-6.12577
1.00000	500	-6.11664
5.00000	10	-6.23935
5.00000	50	-6.15090
5.00000	100	-6.12457
5.00000	200	-6.09997
5.00000	500	-6.11330
10.00000	10	-6.19467
10.00000	50	-6.13907
10.00000	100	-6.12610
10.00000	200	-6.10557
10.00000	500	-6.11241

Tabella 5: dataset: nltcs, nessun algoritmo di SLS applicato, varianza del rumore: 1

alpha	mnist	test_ll
0.50000	10	-6.40386
0.50000	50	-6.22129
0.50000	100	-6.17629
0.50000	200	-6.11294
0.50000	500	-6.09374
1.00000	10	-6.34833
1.00000	50	-6.20471
1.00000	100	-6.13001
1.00000	200	-6.12358
1.00000	500	-6.09309
5.00000	10	-6.22281
5.00000	50	-6.13921
5.00000	100	-6.11422
5.00000	200	-6.10392
5.00000	500	-6.09141
10.00000	10	-6.19747
10.00000	50	-6.14352
10.00000	100	-6.10899
10.00000	200	-6.10105
10.00000	500	-6.13633

Tabella 6: dataset: nltcs, nessun algoritmo di SLS applicato, varianza del rumore: 2

alpha	mnist	test_ll
0.50000	10	-6.36372
0.50000	50	-6.22910
0.50000	100	-6.18135
0.50000	200	-6.11208
0.50000	500	-6.08472
1.00000	10	-6.34566
1.00000	50	-6.21412
1.00000	100	-6.13242
1.00000	200	-6.10306
1.00000	500	-6.09442
5.00000	10	-6.22394

5.00000	50	-6.11743
5.00000	100	-6.11249
5.00000	200	-6.10344
5.00000	500	-6.09353
10.00000	10	-6.18307
10.00000	50	-6.13529
10.00000	100	-6.10228
10.00000	200	-6.09426
10.00000	500	-6.11280

Tabella 7: dataset: nltcs, nessun algoritmo di SLS applicato, varianza del rumore: 3

alpha	mnist	test_ll
0.50000	10	-6.35577
0.50000	50	-6.25925
0.50000	100	-6.17176
0.50000	200	-6.12072
0.50000	500	-6.09210
1.00000	10	-6.31135
1.00000	50	-6.23468
1.00000	100	-6.13348
1.00000	200	-6.08710
1.00000	500	-6.09475
5.00000	10	-6.19364
5.00000	50	-6.16082
5.00000	100	-6.11079
5.00000	200	-6.09327
5.00000	500	-6.10482
10.00000	10	-6.19684
10.00000	50	-6.12901
10.00000	100	-6.12471
10.00000	200	-6.09956
10.00000	500	-6.10248

Tabella 8: dataset: nltcs, Iterative Improvement senza rumore

minst	alpha	test_ll

10	0.50000	-6.46330
50	0.50000	-6.33055
100	0.50000	-6.21818
200	0.50000	-6.15824
500	0.50000	-6.08381
10	1.00000	-6.39304
50	1.00000	-6.30148
100	1.00000	-6.18007
200	1.00000	-6.13408
500	1.00000	-6.06773
10	5.00000	-6.29269
50	5.00000	-6.20016
100	5.00000	-6.14048
200	5.00000	-6.10736
500	5.00000	-6.05411
10	10.00000	-6.23944
50	10.00000	-6.18542
100	10.00000	-6.15031
200	10.00000	-6.10843
500	10.00000	-6.07792

Tabella 9: dataset: nltcs, Iterative Improvement senza rumore

minst	alpha	test_ll
10	0.50000	-6.36980
50	0.50000	-6.22196
100	0.50000	-6.17539
200	0.50000	-6.11439
500	0.50000	-6.09293
10	1.00000	-6.32592
50	1.00000	-6.20494
100	1.00000	-6.12984
200	1.00000	-6.12455
500	1.00000	-6.09284
10	5.00000	-6.21486
50	5.00000	-6.13556
100	5.00000	-6.11506
200	5.00000	-6.10635
500	5.00000	-6.09497

10	10.00000	-6.18216
50	10.00000	-6.14665
100	10.00000	-6.10864
200	10.00000	-6.10083
500	10.00000	-6.13594

Tabella 10: dataset: nltcs, Iterative Improvement con rumore avente varianza 2

minst	alpha	test_ll
10	0.50000	-6.35208
50	0.50000	-6.21991
100	0.50000	-6.14243
200	0.50000	-6.13501
500	0.50000	-6.11457
10	1.00000	-6.34224
50	1.00000	-6.19734
100	1.00000	-6.14516
200	1.00000	-6.10100
500	1.00000	-6.09123
10	5.00000	-6.19478
50	5.00000	-6.11170
100	5.00000	-6.10484
200	5.00000	-6.07638
500	5.00000	-6.09773
10	10.00000	-6.19689
50	10.00000	-6.14672
100	10.00000	-6.12733
200	10.00000	-6.10693
500	10.00000	-6.13927

Tabella 11: dataset: nltcs, Iterative Improvement con rumore avente varianza 0.1

minst	alpha	test_ll
10	0.50000	-6.42910
50	0.50000	-6.27202
100	0.50000	-6.20841

200	0.50000	-6.13534
500	0.50000	-6.09618
10	1.00000	-6.36876
50	1.00000	-6.23962
100	1.00000	-6.18934
200	1.00000	-6.12620
500	1.00000	-6.09036
10	5.00000	-6.24290
50	5.00000	-6.16578
100	5.00000	-6.14800
200	5.00000	-6.06638
500	5.00000	-6.07826
10	10.00000	-6.22805
50	10.00000	-6.15302
100	10.00000	-6.11755
200	10.00000	-6.10926
500	10.00000	-6.08803

Tabella 12: dataset: nltcs, Iterative Improvement con rumore avente varianza 0.5

minst	alpha	test_ll
10	0.50000	-6.37725
50	0.50000	-6.25252
100	0.50000	-6.16200
200	0.50000	-6.11771
500	0.50000	-6.10755
10	1.00000	-6.37504
50	1.00000	-6.20715
100	1.00000	-6.15266
200	1.00000	-6.12888
500	1.00000	-6.11613
10	5.00000	-6.23254
50	5.00000	-6.15083
100	5.00000	-6.12424
200	5.00000	-6.09984
500	5.00000	-6.11749
10	10.00000	-6.18815
50	10.00000	-6.14216

100	10.00000	-6.12766
200	10.00000	-6.10517
500	10.00000	-6.11693

Tabella 13: dataset: nltcs, Randomised Iterative Improvement senza rumore, con probabilità 0.6 e times = 10

alpha	mnist	test_ll
0.5	10	-6.4633
0.5	50	-6.33055
0.5	100	-6.21818
0.5	200	-6.15824
0.5	500	-6.08381
1	10	-6.39304
1	50	-6.30148
1	100	-6.18007
1	200	-6.13408
1	500	-6.06773
5	10	-6.29269
5	50	-6.20016
5	100	-6.14048
5	200	-6.10736
5	500	-6.05411
10	10	-6.23938
10	50	-6.18542
10	100	-6.15031
10	200	-6.10843
10	500	-6.07792

Tabella 14: dataset: nltcs, Randomised Iterative Improvement con rumore avente varianza 0.1, con probabilità 0.6 e times = 10

alpha	mnist	test_ll
0.5	10	-6.42902
0.5	50	-6.27202
0.5	100	-6.20841
0.5	200	-6.13535
0.5	500	-6.09618

1	10	-6.36847
1	50	-6.23962
1	100	-6.18934
1	200	-6.1262
1	500	-6.09036
5	10	-6.2429
5	50	-6.16578
5	100	-6.148
5	200	-6.06697
5	500	-6.07826
10	10	-6.22805
10	50	-6.15302
10	100	-6.11755
10	200	-6.10926
10	500	-6.08803

Tabella 15: dataset: nltcs, Randomised Iterative Improvement con rumore avente varianza di 0.5, con probabilità 0.6 e times = 10

alpha	mnist	test_ll
0.5	10	-6.37725
0.5	50	-6.25252
0.5	100	-6.162
0.5	200	-6.11771
0.5	500	-6.10755
1	10	-6.37506
1	50	-6.20715
1	100	-6.15266
1	200	-6.12856
1	500	-6.11613
5	10	-6.23252
5	50	-6.15083
5	100	-6.12424
5	200	-6.09984
5	500	-6.11749
10	10	-6.18815
10	50	-6.14216
10	100	-6.12766
10	200	-6.10543

10	500	-6.11693

Tabella 16: dataset: nltcs, Randomised Iterative Improvement con rumore avente varianza di 1, con probabilità 0.6 e times = 10

alpha	mnist	test_ll
0.5	10	-6.36976
0.5	50	-6.22196
0.5	100	-6.17539
0.5	200	-6.11439
0.5	500	-6.09293
1	10	-6.32593
1	50	-6.20494
1	100	-6.12984
1	200	-6.12455
1	500	-6.09284
5	10	-6.21487
5	50	-6.13556
5	100	-6.11506
5	200	-6.10635
5	500	-6.09497
10	10	-6.18216
10	50	-6.14665
10	100	-6.10864
10	200	-6.10083
10	500	-6.13594

Tabella 17: dataset: nltcs, Randomised Iterative Improvement con rumore avente varianza di 2, con probabilità 0.6 e times = 10

alpha	mnist	test_ll
0.5	10	-6.35208
0.5	50	-6.21991
0.5	100	-6.14243
0.5	200	-6.13501
0.5	500	-6.11457
1	10	-6.34222
1	50	-6.19734

1	100	-6.14516
1	200	-6.101
1	500	-6.09123
5	10	-6.19478
5	50	-6.1117
5	100	-6.10484
5	200	-6.07638
5	500	-6.09773
10	10	-6.19689
10	50	-6.14672
10	100	-6.12733
10	200	-6.10693
10	500	-6.13927

Tabella 18: dataset: nltcs, Randomised Iterative Improvement senza rumore, con probabilità 0.8 e times = 10

alpha	mnist	test_ll
0.5	10	-6.4633
0.5	50	-6.33055
0.5	100	-6.21818
0.5	200	-6.15824
0.5	500	-6.08381
1	10	-6.39304
1	50	-6.30148
1	100	-6.18007
1	200	-6.13408
1	500	-6.06773
5	10	-6.29269
5	50	-6.20016
5	100	-6.14048
5	200	-6.10736
5	500	-6.05411
10	10	-6.23944
10	50	-6.18542
10	100	-6.15031
10	200	-6.10843
10	500	-6.07792

Tabella 19: dataset: nltcs, Randomised Iterative Improvement con rumore avente varianza di 0.1, con probabilità 0.8 e times = 10

alpha	mnist	test_ll
0.5	10	-6.4291
0.5	50	-6.27202
0.5	100	-6.20841
0.5	200	-6.13534
0.5	500	-6.09618
1	10	-6.36876
1	50	-6.23962
1	100	-6.18934
1	200	-6.1262
1	500	-6.09036
5	10	-6.2429
5	50	-6.16578
5	100	-6.148
5	200	-6.06638
5	500	-6.07826
10	10	-6.22805
10	50	-6.15302
10	100	-6.11755
10	200	-6.10926
10	500	-6.08803

Tabella 20: dataset: nltcs, Randomised Iterative Improvement con rumore avente varianza di 0.5, con probabilità 0.8 e times = 10

alpha	mnist	test_ll
0.5	10	-6.37725
0.5	50	-6.25252
0.5	100	-6.162
0.5	200	-6.11771
0.5	500	-6.10755
1	10	-6.37504
1	50	-6.20715
1	100	-6.15266
1	200	-6.12888
1	500	-6.11613
5	10	-6.23254

5	50	-6.15083
5	100	-6.12424
5	200	-6.09984
5	500	-6.11749
10	10	-6.18815
10	50	-6.14216
10	100	-6.12766
10	200	-6.10517
10	500	-6.11693

Tabella 21: dataset: nltcs, Randomised Iterative Improvement con rumore avente varianza di 1, con probabilità 0.8 e times = 10

alpha	mnist	test_ll
0.5	10	-6.3698
0.5	50	-6.22196
0.5	100	-6.17539
0.5	200	-6.11439
0.5	500	-6.09293
1	10	-6.32592
1	50	-6.20494
1	100	-6.12984
1	200	-6.12455
1	500	-6.09284
5	10	-6.21486
5	50	-6.13556
5	100	-6.11506
5	200	-6.10635
5	500	-6.09497
10	10	-6.18216
10	50	-6.14665
10	100	-6.10864
10	200	-6.10083
10	500	-6.13594

Tabella 22: dataset: nltcs, Randomised Iterative Improvement con rumore avente varianza di 2, con probabilità 0.8 e times = 10

alpha	mnist	test_ll
0.5	10	-6.35208
0.5	50	-6.21991
0.5	100	-6.14243
0.5	200	-6.13501
0.5	500	-6.11457
1	10	-6.34224
1	50	-6.19734
1	100	-6.14516
1	200	-6.101
1	500	-6.09123
5	10	-6.19478
5	50	-6.1117
5	100	-6.10484
5	200	-6.07638
5	500	-6.09773
10	10	-6.19689
10	50	-6.14672
10	100	-6.12733
10	200	-6.10693
10	500	-6.13927

Tabella 23: dataset: nltcs. GRASP senza rumore, k=3 e times=20

mnist	alpha	test_ll
10	0.50000	-6.46470
50	0.50000	-6.33067
100	0.50000	-6.21818
200	0.50000	-6.15824
500	0.50000	-6.08381
10	1.00000	-6.39680
50	1.00000	-6.30174
100	1.00000	-6.18014
200	1.00000	-6.13409
500	1.00000	-6.06792
10	5.00000	-6.28905
50	5.00000	-6.20052
100	5.00000	-6.14047
200	5.00000	-6.10742

500	5.00000	-6.05411
10	10.00000	-6.23572
50	10.00000	-6.18467
100	10.00000	-6.15038
200	10.00000	-6.10842
500	10.00000	-6.07792

Tabella 24: dataset: nltcs. GRASP con rumore avente varianza di 0.1, k=3 e times=20

mnist	alpha	test_ll
10	0.50000	-6.43689
50	0.50000	-6.29316
100	0.50000	-6.23490
200	0.50000	-6.11907
500	0.50000	-6.08413
10	1.00000	-6.36846
50	1.00000	-6.25364
100	1.00000	-6.19728
200	1.00000	-6.14150
500	1.00000	-6.09926
10	5.00000	-6.25055
50	5.00000	-6.17462
100	5.00000	-6.12125
200	5.00000	-6.09377
500	5.00000	-6.08660
10	10.00000	-6.23684
50	10.00000	-6.17011
100	10.00000	-6.13354
200	10.00000	-6.12670
500	10.00000	-6.09542

Tabella 25: dataset: nltcs. GRASP con rumore avente varianza di 0.5, k=3 e times=20

mnist	alpha	test_ll
10	0.50000	-6.37624
50	0.50000	-6.19939

100	0.50000	-6.16802
200	0.50000	-6.13302
500	0.50000	-6.11454
10	1.00000	-6.33618
50	1.00000	-6.21600
100	1.00000	-6.16475
200	1.00000	-6.11245
500	1.00000	-6.08045
10	5.00000	-6.25346
50	5.00000	-6.15292
100	5.00000	-6.10763
200	5.00000	-6.09976
500	5.00000	-6.09662
10	10.00000	-6.20577
50	10.00000	-6.16111
100	10.00000	-6.13682
200	10.00000	-6.10911
500	10.00000	-6.10118

Tabella 26: dataset: nltcs. GRASP con rumore avente varianza di 1, k=3 e times=20

mnist	alpha	test_ll
10	0.50000	-6.33551
50	0.50000	-6.24323
100	0.50000	-6.14590
200	0.50000	-6.13673
500	0.50000	-6.09849
10	1.00000	-6.33970
50	1.00000	-6.22954
100	1.00000	-6.14952
200	1.00000	-6.14029
500	1.00000	-6.10862
10	5.00000	-6.22294
50	5.00000	-6.15214
100	5.00000	-6.12833
200	5.00000	-6.09818
500	5.00000	-6.09551
10	10.00000	-6.21465

50	10.00000	-6.14105
100	10.00000	-6.11040
200	10.00000	-6.10544
500	10.00000	-6.14092

Tabella 27: dataset: nltcs. GRASP con rumore avente varianza di 2, k=3 e times=20

mnist	alpha	test_ll
10	0.50000	-6.37199
50	0.50000	-6.27130
100	0.50000	-6.17216
200	0.50000	-6.12675
500	0.50000	-6.09528
10	1.00000	-6.30394
50	1.00000	-6.20170
100	1.00000	-6.15774
200	1.00000	-6.09597
500	1.00000	-6.08484
10	5.00000	-6.19903
50	5.00000	-6.14982
100	5.00000	-6.14671
200	5.00000	-6.09507
500	5.00000	-6.09641
10	10.00000	-6.18493
50	10.00000	-6.13897
100	10.00000	-6.15724
200	10.00000	-6.09958
500	10.00000	-6.12475

Tabella 28: dataset: nltcs. GRASP senza rumore, k=2 e times=20

mnist	alpha	test_ll
10	0.50000	-6.46330
50	0.50000	-6.33055
100	0.50000	-6.21818
200	0.50000	-6.15824
500	0.50000	-6.08381

10	1.00000	-6.39304
50	1.00000	-6.30148
100	1.00000	-6.18007
200	1.00000	-6.13408
500	1.00000	-6.06773
10	5.00000	-6.29269
50	5.00000	-6.20016
100	5.00000	-6.14048
200	5.00000	-6.10736
500	5.00000	-6.05411
10	10.00000	-6.23944
50	10.00000	-6.18542
100	10.00000	-6.15031
200	10.00000	-6.10843
500	10.00000	-6.07792

Tabella 29: dataset: nltcs. GRASP con rumore avente varianza di 0.1, k=2 e times=20

mnist	alpha	test_ll
10	0.50000	-6.42253
50	0.50000	-6.25328
100	0.50000	-6.19492
200	0.50000	-6.13664
500	0.50000	-6.07789
10	1.00000	-6.34030
50	1.00000	-6.26478
100	1.00000	-6.16324
200	1.00000	-6.11145
500	1.00000	-6.10299
10	5.00000	-6.23029
50	5.00000	-6.17457
100	5.00000	-6.14257
200	5.00000	-6.09716
500	5.00000	-6.10887
10	10.00000	-6.22688
50	10.00000	-6.17211
100	10.00000	-6.12514
200	10.00000	-6.10453

500	10.00000	-6.09702

Tabella 30: dataset: nltcs. GRASP con rumore avente varianza di 0.5, k=2 e times=20

mnist	alpha	test_ll
10	0.50000	-6.37711
50	0.50000	-6.26173
100	0.50000	-6.16558
200	0.50000	-6.13839
500	0.50000	-6.10185
10	1.00000	-6.33538
50	1.00000	-6.23083
100	1.00000	-6.13975
200	1.00000	-6.12828
500	1.00000	-6.11158
10	5.00000	-6.21304
50	5.00000	-6.16839
100	5.00000	-6.12180
200	5.00000	-6.10381
500	5.00000	-6.11475
10	10.00000	-6.22862
50	10.00000	-6.15134
100	10.00000	-6.11265
200	10.00000	-6.12942
500	10.00000	-6.10152

Tabella 31: dataset: nltcs. GRASP con rumore avente varianza di 1, k=2 e times=20

mnist	alpha	test_ll
10	0.50000	-6.31276
50	0.50000	-6.26739
100	0.50000	-6.17068
200	0.50000	-6.12701
500	0.50000	-6.10599
10	1.00000	-6.30525
50	1.00000	-6.21461

100	1.00000	-6.15654
200	1.00000	-6.11267
500	1.00000	-6.08817
10	5.00000	-6.21272
50	5.00000	-6.13990
100	5.00000	-6.09971
200	5.00000	-6.08174
500	5.00000	-6.08593
10	10.00000	-6.15607
50	10.00000	-6.15051
100	10.00000	-6.13046
200	10.00000	-6.10758
500	10.00000	-6.13638

Tabella 32: dataset: nltcs. GRASP con rumore avente varianza di 2, k=2 e times=20

mnist	alpha	test_ll
10	0.50000	-6.36306
50	0.50000	-6.25484
100	0.50000	-6.15950
200	0.50000	-6.12439
500	0.50000	-6.09978
10	1.00000	-6.32357
50	1.00000	-6.20452
100	1.00000	-6.14770
200	1.00000	-6.13809
500	1.00000	-6.10508
10	5.00000	-6.22282
50	5.00000	-6.16231
100	5.00000	-6.12614
200	5.00000	-6.16197
500	5.00000	-6.09882
10	10.00000	-6.17125
50	10.00000	-6.11007
100	10.00000	-6.11713
200	10.00000	-6.11669
500	10.00000	-6.13722

5.3.2 msnbc

Tabella 33: dataset: msnbc. Nessun algoritmo di SLS applicato

alpha	minst	test_ll
0.5	10	-6.1018
0.5	50	inf
0.5	100	-6.07966
0.5	200	inf
0.5	500	-6.06059
1	10	-6.09292
1	50	-6.08174
1	100	inf
1	200	-6.06477
1	500	-6.05681
5	10	-6.07298
5	50	-6.06575
5	100	-6.06108
5	200	-6.05478
5	500	-6.05071
10	10	-6.06688
10	50	-6.06139
10	100	inf
10	200	inf
10	500	-6.04823

Tabella 34: dataset: msnbc. Nessun algoritmo di SLS applicato,rumore applicato con varianza di 0.1

alpha	minst	test_ll
0.5	10	-6.09886
0.5	50	-6.08239
0.5	100	-6.07298
0.5	200	-6.06176
0.5	500	-6.05368
1	10	-6.09274
1	50	-6.07793
1	100	-6.06657
1	200	-6.05812
1	500	-6.05271

5	10	-6.07003
5	50	-6.05995
5	100	-6.05551
5	200	-6.05151
5	500	-6.04912
10	10	-6.06188
10	50	-6.05527
10	100	-6.0503
10	200	-6.04997
10	500	-6.0487

Tabella 35: dataset: msnbc. Nessun algoritmo di SLS applicato,rumore applicato con varianza di 0.5

alpha	minst	test_ll
0.5	10	-6.0972
0.5	50	-6.08376
0.5	100	-6.07216
0.5	200	-6.06044
0.5	500	-6.05297
1	10	-6.08701
1	50	-6.07463
1	100	-6.06328
1	200	-6.05618
1	500	-6.05208
5	10	-6.06685
5	50	-6.05719
5	100	-6.05412
5	200	-6.05006
5	500	-6.04893
10	10	-6.06238
10	50	-6.05576
10	100	-6.05054
10	200	-6.04581
10	500	-6.04681

Tabella 36: dataset: msnbc. Nessun algoritmo di SLS applicato,rumore applicato con varianza di 1

alpha	minst	test_ll
0.5	10	-6.09274
0.5	50	-6.07833
0.5	100	-6.06783
0.5	200	-6.06107
0.5	500	-6.05507
1	10	-6.08243
1	50	-6.07167
1	100	-6.06303
1	200	-6.05582
1	500	-6.05129
5	10	-6.06833
5	50	-6.05931
5	100	-6.05235
5	200	-6.04864
5	500	-6.04735
10	10	-6.05943
10	50	-6.05342
10	100	-6.0498
10	200	-6.04698
10	500	-6.04605

Tabella 37: dataset: msnbc. Nessun algoritmo di SLS applicato,rumore applicato con varianza di 2

alpha	minst	test_ll
0.5	10	-6.09138
0.5	50	-6.07544
0.5	100	-6.06719
0.5	200	-6.06099
0.5	500	-6.05473
1	10	-6.0853
1	50	-6.07152
1	100	-6.06323
1	200	-6.05515
1	500	-6.05456
5	10	-6.06864

5	50	-6.0561
5	100	-6.05451
5	200	-6.04725
5	500	-6.04916
10	10	-6.06131
10	50	-6.05342
10	100	-6.05042
10	200	-6.04709
10	500	-6.04758

Tabella 38: dataset: msnbc.Iterative Improvement senza rumore

alpha	minst	test_ll
0.5	10	-6.09493
0.5	50	-6.08666
0.5	100	-6.0794
0.5	200	-6.06953
0.5	500	-6.06056
1	10	-6.08689
1	50	-6.08127
1	100	-6.07352
1	200	-6.0645
1	500	-6.05669
5	10	-6.06885
5	50	-6.0654
5	100	-6.06166
5	200	-6.05475
5	500	-6.05071
10	10	-6.06344
10	50	-6.06084
10	100	-6.05697
10	200	-6.05163
10	500	-6.04827

Tabella 39: dataset: msnbc.Iterative Improvement con rumore avente varianza di 0.1

alpha	minst	test_ll

0.5	10	-6.09395
0.5	50	-6.08152
0.5	100	-6.07256
0.5	200	-6.06152
0.5	500	-6.05379
1	10	-6.08756
1	50	-6.07749
1	100	-6.06647
1	200	-6.05814
1	500	-6.0527
5	10	-6.06711
5	50	-6.05966
5	100	-6.05552
5	200	-6.05163
5	500	-6.04904
10	10	-6.05904
10	50	-6.05531
10	100	-6.05032
10	200	-6.04996
10	500	-6.04871

Tabella 40: dataset: msnbc.Iterative Improvement con rumore avente varianza di 0.5

alpha	minst	test_ll
0.5	10	-6.09274
0.5	50	-6.08328
0.5	100	-6.07211
0.5	200	-6.06014
0.5	500	-6.05293
1	10	-6.08388
1	50	-6.07409
1	100	-6.06299
1	200	-6.05605
1	500	-6.05199
5	10	-6.06495
5	50	-6.05685
5	100	-6.05424
5	200	-6.04987

5	500	-6.04889
10	10	-6.06049
10	50	-6.05561
10	100	-6.05041
10	200	-6.04582
10	500	-6.04687

Tabella 41: dataset: msnbc.Iterative Improvement con rumore avente varianza di 1

alpha	minst	test_ll
0.5	10	-6.08724
0.5	50	-6.07759
0.5	100	-6.0676
0.5	200	-6.06088
0.5	500	-6.05501
1	10	-6.07757
1	50	-6.07125
1	100	-6.06285
1	200	-6.05574
1	500	-6.05131
5	10	-6.066
5	50	-6.05903
5	100	-6.05238
5	200	-6.04863
5	500	-6.04754
10	10	-6.05782
10	50	-6.05321
10	100	-6.04983
10	200	-6.04695
10	500	-6.04605

Tabella 42: dataset: msnbc.Iterative Improvement con rumore avente varianza di 2

alpha	minst	test_ll
0.5	10	-6.08687
0.5	50	-6.07454

0.5	100	-6.06695
0.5	200	-6.06082
0.5	500	-6.05477
1	10	-6.08093
1	50	-6.07083
1	100	-6.06324
1	200	-6.05518
1	500	-6.0546
5	10	-6.06582
5	50	-6.05563
5	100	-6.05443
5	200	-6.04738
5	500	-6.04913
10	10	-6.06044
10	50	-6.05318
10	100	-6.05036
10	200	-6.04705
10	500	-6.04755

Tabella 43: dataset: msnbc. Randomised Iterative Improvement senza rumore, probabilità di 0.6, times=10

alpha	minst	test_ll
0.5	10	inf
0.5	50	inf
0.5	100	-6.07953
0.5	200	inf
0.5	500	-6.06056
1	10	inf
1	50	inf
1	100	-6.07351
1	200	-6.06451
1	500	-6.05669
5	10	-6.06886
5	50	-6.0654
5	100	-6.06092
5	200	-6.05483
5	500	-6.05071
10	10	-6.06375

10	50	-6.06083
10	100	-6.05703
10	200	inf
10	500	-6.04823

Tabella 44: dataset: msnbc. Randomised Iterative Improvement con rumore avente varianza di 0.1, probabilità di 0.6, times=10

alpha	minst	test_ll
0.5	10	-6.09396
0.5	50	-6.08152
0.5	100	-6.07256
0.5	200	-6.06152
0.5	500	-6.05379
1	10	-6.08763
1	50	-6.07749
1	100	-6.06647
1	200	-6.05814
1	500	-6.0527
5	10	-6.06711
5	50	-6.05966
5	100	-6.05552
5	200	-6.05163
5	500	-6.04904
10	10	-6.05904
10	50	-6.0553
10	100	-6.05032
10	200	-6.04996
10	500	-6.04871

Tabella 45: dataset: msnbc. Randomised Iterative Improvement con rumore avente varianza di 0.5, probabilità di 0.6, times=10

alpha	minst	test_ll
0.5	10	-6.09274
0.5	50	-6.08328
0.5	100	-6.07211
0.5	200	-6.06014

0.5	500	-6.05293
1	10	-6.08388
1	50	-6.07409
1	100	-6.06299
1	200	-6.05605
1	500	-6.05199
5	10	-6.06495
5	50	-6.05685
5	100	-6.05424
5	200	-6.04987
5	500	-6.04889
10	10	-6.06049
10	50	-6.05561
10	100	-6.05041
10	200	-6.04582
10	500	-6.04687

Tabella 46: dataset: msnbc. Randomised Iterative Improvement con rumore avente varianza di 1, probabilità di 0.6, times=10

alpha	minst	test_ll
0.5	10	-6.08724
0.5	50	-6.07759
0.5	100	-6.0676
0.5	200	-6.06088
0.5	500	-6.05501
1	10	-6.07757
1	50	-6.07125
1	100	-6.06285
1	200	-6.05574
1	500	-6.05131
5	10	-6.066
5	50	-6.05903
5	100	-6.05238
5	200	-6.04863
5	500	-6.04754
10	10	-6.05782
10	50	-6.05321
10	100	-6.04981

10	200	-6.04695
10	500	-6.04605

Tabella 47: dataset: msnbc. Randomised Iterative Improvement con rumore avente varianza di 2, probabilità di 0.6, times=10

alpha	minst	test_ll
0.5	10	-6.08687
0.5	50	-6.07454
0.5	100	-6.06695
0.5	200	-6.06082
0.5	500	-6.05477
1	10	-6.08093
1	50	-6.07083
1	100	-6.06324
1	200	-6.05518
1	500	-6.0546
5	10	-6.06582
5	50	-6.05563
5	100	-6.05443
5	200	-6.04738
5	500	-6.04913
10	10	-6.06044
10	50	-6.05318
10	100	-6.05036
10	200	-6.04705
10	500	-6.04755

Tabella 48: dataset: msnbc. GRASP senza rumore, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-6.09476
0.50000	50	-6.08666
0.50000	100	-6.07940
0.50000	200	-6.06953
0.50000	500	-6.06056
1.00000	10	-6.08690
1.00000	50	-6.08078

1.00000	100	-6.07351
1.00000	200	-6.06449
1.00000	500	-6.05670
5.00000	10	-6.06884
5.00000	50	-6.06545
5.00000	100	-6.06092
5.00000	200	-6.05525
5.00000	500	-6.05071
10.00000	10	inf
10.00000	50	-6.06097
10.00000	100	-6.05695
10.00000	200	-6.05163
10.00000	500	-6.04823

Tabella 49: dataset: msnbc. GRASP con rumore avente varianza di 0.1, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-6.09238
0.50000	50	-6.08325
0.50000	100	-6.07069
0.50000	200	-6.06160
0.50000	500	-6.05329
1.00000	10	-6.08707
1.00000	50	-6.07506
1.00000	100	-6.06739
1.00000	200	-6.05759
1.00000	500	-6.05438
5.00000	10	-6.06813
5.00000	50	-6.05959
5.00000	100	-6.05682
5.00000	200	-6.04910
5.00000	500	-6.04809
10.00000	10	-6.06130
10.00000	50	-6.05610
10.00000	100	-6.05295
10.00000	200	-6.04846
10.00000	500	-6.04604

Tabella 50: dataset: msnbc. GRASP con rumore avente varianza di 0.5, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-6.08897
0.50000	50	-6.07542
0.50000	100	-6.06913
0.50000	200	-6.06118
0.50000	500	-6.05478
1.00000	10	-6.08613
1.00000	50	-6.07488
1.00000	100	-6.06214
1.00000	200	-6.05710
1.00000	500	-6.05409
5.00000	10	-6.06637
5.00000	50	-6.05930
5.00000	100	-6.05647
5.00000	200	-6.04992
5.00000	500	-6.04978
10.00000	10	-6.05899
10.00000	50	-6.05585
10.00000	100	-6.04989
10.00000	200	-6.04542
10.00000	500	-6.04800

Tabella 51: dataset: msnbc. GRASP con rumore avente varianza di 1, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-6.08719
0.50000	50	-6.07822
0.50000	100	-6.06644
0.50000	200	-6.05603
0.50000	500	-6.05635
1.00000	10	-6.08040
1.00000	50	-6.07218
1.00000	100	-6.06226
1.00000	200	-6.05678
1.00000	500	-6.05241
5.00000	10	-6.06546

5.00000	50	-6.05786
5.00000	100	-6.05365
5.00000	200	-6.04866
5.00000	500	-6.05200
10.00000	10	-6.06130
10.00000	50	-6.05297
10.00000	100	-6.04815
10.00000	200	-6.04767
10.00000	500	-6.04674

Tabella 52: dataset: msnbc. GRASP con rumore avente varianza di 2, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-6.08746
0.50000	50	-6.07373
0.50000	100	-6.06740
0.50000	200	-6.06083
0.50000	500	-6.05353
1.00000	10	-6.07751
1.00000	50	-6.07044
1.00000	100	-6.06386
1.00000	200	-6.05509
1.00000	500	-6.05438
5.00000	10	-6.06289
5.00000	50	-6.05700
5.00000	100	-6.05271
5.00000	200	-6.05128
5.00000	500	-6.04989
10.00000	10	-6.05792
10.00000	50	-6.05265
10.00000	100	-6.04898
10.00000	200	-6.04638
10.00000	500	-6.04658

5.3.3 plants

Tabella 53: dataset: plants. Nessun algoritmo di SLS applicato

alpha	minst	test_ll
0.50000	10	-18.34506
0.50000	50	-15.54635
0.50000	100	-14.43864
0.50000	200	-13.74225
0.50000	500	-13.42109
1.00000	10	-17.56881
1.00000	50	-15.16388
1.00000	100	-14.14518
1.00000	200	-13.61191
1.00000	500	-13.37280
5.00000	10	-15.79672
5.00000	50	-14.13266
5.00000	100	-13.61289
5.00000	200	-13.38759
5.00000	500	-13.31657
10.00000	10	-15.12328
10.00000	50	-13.93696
10.00000	100	-13.50377
10.00000	200	-13.33208
10.00000	500	-13.34332

Tabella 54: dataset: plants. Nessun algoritmo di SLS applicato, rumore avente varianza di 0.1

alpha	minst	test_ll
0.50000	10	-18.22500
0.50000	50	-15.18123
0.50000	100	-14.49518
0.50000	200	-14.14441
0.50000	500	-14.14831
1.00000	10	-17.65165
1.00000	50	-14.88795
1.00000	100	-14.29635
1.00000	200	-14.12616
1.00000	500	-14.18353
5.00000	10	-16.25273
5.00000	50	-14.36497
5.00000	100	-14.06806

5.00000	200	-14.02948
5.00000	500	-14.26160
10.00000	10	-15.40876
10.00000	50	-14.24034
10.00000	100	-13.91597
10.00000	200	-13.95932
10.00000	500	-14.37871

Tabella 55: dataset: plants. Nessun algoritmo di SLS applicato, rumore avente varianza di 0.5

alpha	minst	test_ll
0.50000	10	-16.69839
0.50000	50	-14.92640
0.50000	100	-14.52405
0.50000	200	-14.41954
0.50000	500	-15.39397
1.00000	10	-16.15963
1.00000	50	-14.55360
1.00000	100	-14.27099
1.00000	200	-14.49579
1.00000	500	-15.38226
5.00000	10	-15.32557
5.00000	50	-14.38040
5.00000	100	-14.19167
5.00000	200	-14.38686
5.00000	500	-15.28087
10.00000	10	-14.79711
10.00000	50	-14.17515
10.00000	100	-14.37276
10.00000	200	-14.55493
10.00000	500	-15.39746

Tabella 56: dataset: plants. Nessun algoritmo di SLS applicato, rumore avente varianza di 1

alpha	minst	test_ll
0.50000	10	-16.17996

0.50000	50	-14.71248
0.50000	100	-14.52424
0.50000	200	-14.59403
0.50000	500	-15.25462
1.00000	10	-15.84489
1.00000	50	-14.58301
1.00000	100	-14.40907
1.00000	200	-14.47674
1.00000	500	-15.34071
5.00000	10	-14.81431
5.00000	50	-14.17908
5.00000	100	-14.19804
5.00000	200	-14.51679
5.00000	500	-15.24408
10.00000	10	-14.74814
10.00000	50	-14.11680
10.00000	100	-14.33496
10.00000	200	-14.64048
10.00000	500	-15.35965

Tabella 57: dataset: plants. Nessun algoritmo di SLS applicato, rumore avente varianza di 2

alpha	minst	test_ll
0.50000	10	-15.95725
0.50000	50	-14.51371
0.50000	100	-14.48357
0.50000	200	-14.66337
0.50000	500	-15.24563
1.00000	10	-15.76220
1.00000	50	-14.36964
1.00000	100	-14.31814
1.00000	200	-14.42664
1.00000	500	-15.40555
5.00000	10	-14.93321
5.00000	50	-14.19586
5.00000	100	-14.24181
5.00000	200	-14.39096
5.00000	500	-15.28972

10.00000	10	-14.72755
10.00000	50	-14.23618
10.00000	100	-14.25716
10.00000	200	-14.63343
10.00000	500	-15.59013

Tabella 58: dataset: plants. Iterative improvement senza rumore

alpha	minst	test_ll
0.50000	10	-18.07949
0.50000	50	-15.54599
0.50000	100	-14.43883
0.50000	200	-13.74234
0.50000	500	-13.42111
1.00000	10	-17.34970
1.00000	50	-15.16379
1.00000	100	-14.14551
1.00000	200	-13.61206
1.00000	500	-13.37280
5.00000	10	-15.67038
5.00000	50	-14.13396
5.00000	100	-13.61419
5.00000	200	-13.38810
5.00000	500	-13.31657
10.00000	10	-15.04656
10.00000	50	-13.93858
10.00000	100	-13.50459
10.00000	200	-13.33308
10.00000	500	-13.34332

Tabella 59: dataset: plants. Iterative improvement con rumore avente varianza di 0.1

alpha	minst	test_ll
0.50000	10	-17.94585
0.50000	50	-15.18334
0.50000	100	-14.49583
0.50000	200	-14.14443

0.50000	500	-14.14837
1.00000	10	-17.39636
1.00000	50	-14.88956
1.00000	100	-14.29663
1.00000	200	-14.12585
1.00000	500	-14.18353
5.00000	10	-16.15021
5.00000	50	-14.37039
5.00000	100	-14.06819
5.00000	200	-14.03037
5.00000	500	-14.26164
10.00000	10	-15.32935
10.00000	50	-14.24338
10.00000	100	-13.92070
10.00000	200	-13.95949
10.00000	500	-14.37927

Tabella 60: dataset: plants. Iterative improvement con rumore avente varianza di 0.5

alpha	minst	test_ll
0.50000	10	-16.59947
0.50000	50	-14.92737
0.50000	100	-14.52380
0.50000	200	-14.41984
0.50000	500	-15.39397
1.00000	10	-16.06660
1.00000	50	-14.54638
1.00000	100	-14.27031
1.00000	200	-14.49488
1.00000	500	-15.38227
5.00000	10	-15.27942
5.00000	50	-14.38140
5.00000	100	-14.19264
5.00000	200	-14.38723
5.00000	500	-15.28089
10.00000	10	-14.76145
10.00000	50	-14.17603
10.00000	100	-14.37370

10.00000	200	-14.55570
10.00000	500	-15.39762

Tabella 61: dataset: plants. Iterative improvement con rumore avente varianza di 1

alpha	minst	test_ll
0.50000	10	-16.08432
0.50000	50	-14.71175
0.50000	100	-14.52416
0.50000	200	-14.59392
0.50000	500	-15.25461
1.00000	10	-15.77395
1.00000	50	-14.58409
1.00000	100	-14.40922
1.00000	200	-14.47689
1.00000	500	-15.34074
5.00000	10	-14.75881
5.00000	50	-14.18051
5.00000	100	-14.19975
5.00000	200	-14.51665
5.00000	500	-15.24408
10.00000	10	-14.68411
10.00000	50	-14.11847
10.00000	100	-14.33542
10.00000	200	-14.64369
10.00000	500	-15.35965

Tabella 62: dataset: plants. Iterative improvement con rumore avente varianza di 2

alpha	minst	test_ll
0.50000	10	-15.87610
0.50000	50	-14.51356
0.50000	100	-14.48728
0.50000	200	-14.66342
0.50000	500	-15.24564
1.00000	10	-15.67484

1.00000	50	-14.36938
1.00000	100	-14.31769
1.00000	200	-14.42674
1.00000	500	-15.40555
5.00000	10	-14.89405
5.00000	50	-14.19666
5.00000	100	-14.24246
5.00000	200	-14.39173
5.00000	500	-15.28972
10.00000	10	-14.70370
10.00000	50	-14.23923
10.00000	100	-14.26126
10.00000	200	-14.63441
10.00000	500	-15.59006

Tabella 63: dataset: plants. Randomised Iterative Improvement senza rumore, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-18.12027
0.50000	50	-15.54599
0.50000	100	-14.43883
0.50000	200	-13.74234
0.50000	500	-13.42111
1.00000	10	-17.40515
1.00000	50	-15.16363
1.00000	100	-14.14550
1.00000	200	-13.61206
1.00000	500	-13.37280
5.00000	10	-15.69009
5.00000	50	-14.13380
5.00000	100	-13.61404
5.00000	200	-13.38810
5.00000	500	-13.31657
10.00000	10	-15.06204
10.00000	50	-13.93820
10.00000	100	-13.50433
10.00000	200	-13.33308
10.00000	500	-13.34332

Tabella 64: dataset: plants. Randomised Iterative Improvement con rumore avente varianza di 0.1, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-17.99325
0.50000	50	-15.18332
0.50000	100	-14.49586
0.50000	200	-14.14443
0.50000	500	-14.14837
1.00000	10	-17.43437
1.00000	50	-14.88944
1.00000	100	-14.29537
1.00000	200	-14.12585
1.00000	500	-14.18353
5.00000	10	-16.15608
5.00000	50	-14.36950
5.00000	100	-14.06819
5.00000	200	-14.03037
5.00000	500	-14.26164
10.00000	10	-15.33884
10.00000	50	-14.24338
10.00000	100	-13.92018
10.00000	200	-13.95949
10.00000	500	-14.37923

Tabella 65: dataset: plants. Randomised Iterative Improvement con rumore avente varianza di 0.5, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-16.62150
0.50000	50	-14.92741
0.50000	100	-14.52380
0.50000	200	-14.41973
0.50000	500	-15.39397
1.00000	10	-16.07981
1.00000	50	-14.54621
1.00000	100	-14.27029
1.00000	200	-14.49491
1.00000	500	-15.38227
5.00000	10	-15.28454

5.00000	50	-14.38137
5.00000	100	-14.19260
5.00000	200	-14.38735
5.00000	500	-15.28089
10.00000	10	-14.76749
10.00000	50	-14.17603
10.00000	100	-14.37370
10.00000	200	-14.55570
10.00000	500	-15.39762

Tabella 66: dataset: plants. Randomised Iterative Improvement con rumore avente varianza di 1, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-16.10199
0.50000	50	-14.71175
0.50000	100	-14.52415
0.50000	200	-14.59409
0.50000	500	-15.25460
1.00000	10	-15.78869
1.00000	50	-14.58302
1.00000	100	-14.40922
1.00000	200	-14.47689
1.00000	500	-15.34074
5.00000	10	-14.76605
5.00000	50	-14.18053
5.00000	100	-14.19961
5.00000	200	-14.51663
5.00000	500	-15.24408
10.00000	10	-14.69423
10.00000	50	-14.11842
10.00000	100	-14.33542
10.00000	200	-14.64336
10.00000	500	-15.35965

Tabella 67: dataset: plants. Randomised Iterative Improvement con rumore avente varianza di 2, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-15.89351
0.50000	50	-14.51393
0.50000	100	-14.48723
0.50000	200	-14.66342
0.50000	500	-15.24564
1.00000	10	-15.68885
1.00000	50	-14.36926
1.00000	100	-14.31771
1.00000	200	-14.42673
1.00000	500	-15.40555
5.00000	10	-14.89732
5.00000	50	-14.19666
5.00000	100	-14.24229
5.00000	200	-14.39173
5.00000	500	-15.28972
10.00000	10	-14.70927
10.00000	50	-14.23999
10.00000	100	-14.25951
10.00000	200	-14.63428
10.00000	500	-15.59006

Tabella 68: dataset: plants. GRASP senza rumore, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-18.07949
0.50000	50	-15.54599
0.50000	100	-14.43883
0.50000	200	-13.74234
0.50000	500	-13.42111
1.00000	10	-17.34970
1.00000	50	-15.16379
1.00000	100	-14.14551
1.00000	200	-13.61206
1.00000	500	-13.37280
5.00000	10	-15.67038
5.00000	50	-14.13396
5.00000	100	-13.61419
5.00000	200	-13.38810

5.00000	500	-13.31657
10.00000	10	-15.04656
10.00000	50	-13.93858
10.00000	100	-13.50459
10.00000	200	-13.33308
10.00000	500	-13.34332

Tabella 69: dataset: plants. GRASP con rumore avente varianza di 0.1, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-18.47828
0.50000	50	-15.18229
0.50000	100	-14.59109
0.50000	200	-14.16208
0.50000	500	-14.41231
1.00000	10	-17.23546
1.00000	50	-14.75868
1.00000	100	-14.18205
1.00000	200	-14.02953
1.00000	500	-14.28273
5.00000	10	-15.94011
5.00000	50	-14.25848
5.00000	100	-14.03509
5.00000	200	-13.95989
5.00000	500	-14.25904
10.00000	10	-15.40849
10.00000	50	-14.27418
10.00000	100	-13.99200
10.00000	200	-14.07465
10.00000	500	-14.36889

Tabella 70: dataset: plants. GRASP con rumore avente varianza di 0.5, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-16.41338
0.50000	50	-14.89816

0.50000	100	-14.52314
0.50000	200	-14.54890
0.50000	500	-15.31216
1.00000	10	-16.36803
1.00000	50	-14.49050
1.00000	100	-14.42055
1.00000	200	-14.43960
1.00000	500	-15.21043
5.00000	10	-15.05668
5.00000	50	-14.18184
5.00000	100	-14.18369
5.00000	200	-14.42763
5.00000	500	-15.19951
10.00000	10	-14.86253
10.00000	50	-14.30112
10.00000	100	-14.23049
10.00000	200	-14.70801
10.00000	500	-15.19930

Tabella 71: dataset: plants. GRASP con rumore avente varianza di 1, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-16.20078
0.50000	50	-14.94015
0.50000	100	-14.47259
0.50000	200	-14.61070
0.50000	500	-15.22049
1.00000	10	-15.63829
1.00000	50	-14.56272
1.00000	100	-14.41656
1.00000	200	-14.51440
1.00000	500	-15.15965
5.00000	10	-14.81668
5.00000	50	-14.17960
5.00000	100	-14.12578
5.00000	200	-14.51885
5.00000	500	-15.33101
10.00000	10	-14.63288

10.00000	50	-14.22588
10.00000	100	-14.39067
10.00000	200	-14.73953
10.00000	500	-15.44410

Tabella 72: dataset: plants. GRASP con rumore avente varianza di 2, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-16.12965
0.50000	50	-14.62373
0.50000	100	-14.37235
0.50000	200	-14.65704
0.50000	500	-15.25362
1.00000	10	-15.63459
1.00000	50	-14.37064
1.00000	100	-14.32341
1.00000	200	-14.51511
1.00000	500	-15.29370
5.00000	10	-14.68516
5.00000	50	-14.06141
5.00000	100	-14.17007
5.00000	200	-14.46313
5.00000	500	-15.16306
10.00000	10	-14.62365
10.00000	50	-14.14985
10.00000	100	-14.27446
10.00000	200	-14.72060
10.00000	500	-15.51529

5.3.4 audio

Tabella 73: dataset: audio. Nessun algoritmo di SLS applicato

alpha	minst	test_ll
0.5	10	-91.14669
0.5	50	-61.08795
0.5	100	-50.85409
0.5	200	-45.75492

0.5	500	-43.02708
1	10	-84.28958
1	50	-58.36294
1	100	-49.65196
1	200	-45.32049
1	500	-42.92807
5	10	-68.27392
5	50	-52.2251
5	100	-47.01272
5	200	-44.08191
5	500	-42.47781
10	10	-61.89317
10	50	-49.65934
10	100	-45.71415
10	200	-43.50611
10	500	-42.3591

Tabella 74: dataset: audio. Nessun algoritmo di SLS applicato, rumore avente varianza di 0.1

alpha	minst	test_ll
0.5	10	-93.25399
0.5	50	-66.79338
0.5	100	-58.76185
0.5	200	-50.47885
0.5	500	-44.00786
1	10	-84.91422
1	50	-62.4118
1	100	-56.09414
1	200	-48.44995
1	500	-44.09925
5	10	-68.59961
5	50	-54.88423
5	100	-51.09852
5	200	-46.0104
5	500	-43.20257
10	10	-61.19806
10	50	-52.37637
10	100	-48.66114

10	200	-44.36606
10	500	-43.23819

Tabella 75: dataset: audio. Nessun algoritmo di SLS applicato, rumore avente varianza di 0.1

alpha	minst	test_ll
0.5	10	-73.62908
0.5	50	-55.61807
0.5	100	-48.19462
0.5	200	-44.53124
0.5	500	-43.40571
1	10	-68.21539
1	50	-52.50207
1	100	-46.19494
1	200	-43.96729
1	500	-43.24955
5	10	-57.97143
5	50	-48.83281
5	100	-44.58327
5	200	-43.40442
5	500	-43.11888
10	10	-53.18342
10	50	-46.14378
10	100	-44.01881
10	200	-43.21577
10	500	-43.20435

Tabella 76: dataset: audio. Nessun algoritmo di SLS applicato, rumore avente varianza di 1

alpha	minst	test_ll
0.5	10	-68.64829
0.5	50	-50.47777
0.5	100	-46.03299
0.5	200	-44.05689
0.5	500	-43.38003
1	10	-63.26593

1	50	-49.21369
1	100	-45.58622
1	200	-43.60246
1	500	-43.34447
5	10	-54.78296
5	50	-46.55168
5	100	-44.09825
5	200	-43.12464
5	500	-43.20069
10	10	-51.49676
10	50	-45.44291
10	100	-43.82776
10	200	-42.99448
10	500	-43.06078

Tabella 77: dataset: audio. Nessun algoritmo di SLS applicato, rumore avente varianza di 2

alpha	minst	test_ll
0.5	10	-64.28529
0.5	50	-49.24767
0.5	100	-45.48895
0.5	200	-43.7569
0.5	500	-43.4345
1	10	-61.80687
1	50	-48.73591
1	100	-45.33313
1	200	-43.72603
1	500	-43.41807
5	10	-53.5221
5	50	-46.17549
5	100	-43.938
5	200	-43.05493
5	500	-43.09917
10	10	-50.62946
10	50	-45.01344
10	100	-43.67545
10	200	-43.11812
10	500	-43.22579

Tabella 78: dataset: audio. Iterative Improvement senza rumore

alpha	minst	test_ll
0.5	10	-86.53359
0.5	50	-61.07992
0.5	100	-50.85409
0.5	200	-45.75492
0.5	500	-43.02708
1	10	-80.3469
1	50	-58.35761
1	100	-49.65196
1	200	-45.32049
1	500	-42.92807
5	10	-65.87959
5	50	-52.22314
5	100	-47.01272
5	200	-44.08191
5	500	-42.47781
10	10	-60.17661
10	50	-49.65631
10	100	-45.71415
10	200	-43.50611
10	500	-42.3591

Tabella 79: dataset: audio. Iterative Improvement con rumore avente varianza di 0.1

alpha	minst	test_ll
0.5	10	-88.93752
0.5	50	-66.79422
0.5	100	-58.76185
0.5	200	-50.47885
0.5	500	-44.00786
1	10	-81.29596
1	50	-62.40237
1	100	-56.09414
1	200	-48.44995
1	500	-44.09925
5	10	-66.51819
5	50	-54.88106

5	100	-51.09852
5	200	-46.0104
5	500	-43.20257
10	10	-59.71859
10	50	-52.3738
10	100	-48.66114
10	200	-44.36606
10	500	-43.23819

Tabella 80: dataset: audio. Iterative Improvement con rumore avente varianza di 0.5

alpha	minst	test_ll
0.5	10	-72.40827
0.5	50	-55.5884
0.5	100	-48.19462
0.5	200	-44.53124
0.5	500	-43.40571
1	10	-67.02668
1	50	-52.49503
1	100	-46.19494
1	200	-43.96729
1	500	-43.24955
5	10	-57.41615
5	50	-48.82937
5	100	-44.58327
5	200	-43.40442
5	500	-43.11888
10	10	-52.69957
10	50	-46.14339
10	100	-44.01881
10	200	-43.21577
10	500	-43.20435

Tabella 81: dataset: audio. Iterative Improvement con rumore avente varianza di 1

alpha	minst	test_ll

0.5	10	-67.21437
0.5	50	-50.47272
0.5	100	-46.03299
0.5	200	-44.05689
0.5	500	-43.38003
1	10	-61.93876
1	50	-49.20654
1	100	-45.58622
1	200	-43.60246
1	500	-43.34447
5	10	-54.09819
5	50	-46.54914
5	100	-44.09825
5	200	-43.12464
5	500	-43.20069
10	10	-50.94421
10	50	-45.43967
10	100	-43.82776
10	200	-42.99448
10	500	-43.06078

Tabella 82: dataset: audio. Iterative Improvement con rumore avente varianza di 2

alpha	minst	test_ll
0.5	10	-62.95375
0.5	50	-49.23908
0.5	100	-45.48895
0.5	200	-43.7569
0.5	500	-43.4345
1	10	-60.6537
1	50	-48.7289
1	100	-45.33313
1	200	-43.72603
1	500	-43.41807
5	10	-52.84854
5	50	-46.17535
5	100	-43.938
5	200	-43.05493

5	500	-43.09917
10	10	-50.07419
10	50	-45.01224
10	100	-43.67545
10	200	-43.11812
10	500	-43.22579

Tabella 83: dataset: audio. Randomised Iterative Improvement senza rumore, con probabilità di 0.6, times=10

alpha	minst	test_ll
0.50000	10	-89.49295
0.50000	50	-61.08580
0.50000	100	-50.85409
0.50000	200	-45.75492
0.50000	500	-43.02708
1.00000	10	-82.90760
1.00000	50	-58.36284
1.00000	100	-49.65196
1.00000	200	-45.32049
1.00000	500	-42.92807
5.00000	10	-67.43674
5.00000	50	-52.22478
5.00000	100	-47.01272
5.00000	200	-44.08191
5.00000	500	-42.47781
10.00000	10	-61.27329
10.00000	50	-49.65832
10.00000	100	-45.71415
10.00000	200	-43.50611
10.00000	500	-42.35910

Tabella 84: dataset: audio. Randomised Iterative Improvement con rumore avente varianza di 0.1, con probabilità di 0.6, times=10

alpha	minst	test_ll
0.50000	10	-91.63904
0.50000	50	-66.79385

0.50000	100	-58.76185
0.50000	200	-50.47885
0.50000	500	-44.00786
1.00000	10	-83.53287
1.00000	50	-62.41378
1.00000	100	-56.09414
1.00000	200	-48.44995
1.00000	500	-44.09925
5.00000	10	-67.77280
5.00000	50	-54.88209
5.00000	100	-51.09852
5.00000	200	-46.01040
5.00000	500	-43.20257
10.00000	10	-60.63103
10.00000	50	-52.37320
10.00000	100	-48.66114
10.00000	200	-44.36606
10.00000	500	-43.23819

Tabella 85: dataset: audio. Randomised Iterative Improvement con rumore avente varianza di 0.5, con probabilità di 0.6, times=10

alpha	minst	test_ll
0.50000	10	-73.17010
0.50000	50	-55.60261
0.50000	100	-48.19462
0.50000	200	-44.53124
0.50000	500	-43.40571
1.00000	10	-67.72980
1.00000	50	-52.49980
1.00000	100	-46.19494
1.00000	200	-43.96729
1.00000	500	-43.24955
5.00000	10	-57.74201
5.00000	50	-48.82967
5.00000	100	-44.58327
5.00000	200	-43.40442
5.00000	500	-43.11888
10.00000	10	-52.95476

10.00000	50	-46.14351
10.00000	100	-44.01881
10.00000	200	-43.21577
10.00000	500	-43.20435

Tabella 86: dataset: audio. Randomised Iterative Improvement con rumore avente varianza di 1, con probabilità di 0.6, times=10

alpha	minst	test_ll
0.50000	10	-68.08467
0.50000	50	-50.47476
0.50000	100	-46.03299
0.50000	200	-44.05689
0.50000	500	-43.38003
1.00000	10	-62.73354
1.00000	50	-49.21161
1.00000	100	-45.58622
1.00000	200	-43.60246
1.00000	500	-43.34447
5.00000	10	-54.50358
5.00000	50	-46.54870
5.00000	100	-44.09825
5.00000	200	-43.12464
5.00000	500	-43.20069
10.00000	10	-51.24390
10.00000	50	-45.44091
10.00000	100	-43.82776
10.00000	200	-42.99448
10.00000	500	-43.06078

Tabella 87: dataset: audio. Randomised Iterative Improvement con rumore avente varianza di 2, con probabilità di 0.6, times=10

alpha	minst	test_ll
0.50000	10	-63.81357
0.50000	50	-49.24532
0.50000	100	-45.48895
0.50000	200	-43.75690

0.50000	500	-43.43450
1.00000	10	-61.38592
1.00000	50	-48.73387
1.00000	100	-45.33313
1.00000	200	-43.72603
1.00000	500	-43.41807
5.00000	10	-53.25954
5.00000	50	-46.17514
5.00000	100	-43.93800
5.00000	200	-43.05493
5.00000	500	-43.09917
10.00000	10	-50.37972
10.00000	50	-45.01229
10.00000	100	-43.67545
10.00000	200	-43.11812
10.00000	500	-43.22579

Tabella 88: dataset: audio. GRASP senza rumore, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-86.53359
0.50000	50	-61.07992
0.50000	100	-50.85409
0.50000	200	-45.75492
0.50000	500	-43.02708
1.00000	10	-80.34690
1.00000	50	-58.35761
1.00000	100	-49.65196
1.00000	200	-45.32049
1.00000	500	-42.92807
5.00000	10	-65.87959
5.00000	50	-52.22314
5.00000	100	-47.01272
5.00000	200	-44.08191
5.00000	500	-42.47781
10.00000	10	-60.17661
10.00000	50	-49.65631
10.00000	100	-45.71415
10.00000	200	-43.50611

10.00000	500	-42.35910

Tabella 89: dataset: audio. GRASP con rumore avente varianza di 0.1, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-89.24376
0.50000	50	-65.86100
0.50000	100	-59.61444
0.50000	200	-49.98266
0.50000	500	-43.54181
1.00000	10	-81.61063
1.00000	50	-63.14966
1.00000	100	-56.47197
1.00000	200	-48.11819
1.00000	500	-43.76387
5.00000	10	-66.54191
5.00000	50	-54.79697
5.00000	100	-50.14578
5.00000	200	-45.66688
5.00000	500	-43.27520
10.00000	10	-60.79241
10.00000	50	-52.07687
10.00000	100	-48.53165
10.00000	200	-45.19603
10.00000	500	-43.26450

Tabella 90: dataset: audio. GRASP con rumore avente varianza di 0.5, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-71.71523
0.50000	50	-55.95491
0.50000	100	-47.02566
0.50000	200	-44.37196
0.50000	500	-43.49786
1.00000	10	-68.11237
1.00000	50	-53.76134

1.00000	100	-46.53680
1.00000	200	-44.11199
1.00000	500	-43.36671
5.00000	10	-57.14880
5.00000	50	-48.26350
5.00000	100	-44.64247
5.00000	200	-43.44105
5.00000	500	-43.13550
10.00000	10	-53.01980
10.00000	50	-46.68603
10.00000	100	-44.00530
10.00000	200	-43.13181
10.00000	500	-43.14111

Tabella 91: dataset: audio. GRASP con rumore avente varianza di 1, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-65.83947
0.50000	50	-50.91896
0.50000	100	-46.05856
0.50000	200	-43.82429
0.50000	500	-43.43185
1.00000	10	-62.08980
1.00000	50	-49.40018
1.00000	100	-45.65623
1.00000	200	-43.96511
1.00000	500	-43.25371
5.00000	10	-54.13160
5.00000	50	-46.51167
5.00000	100	-44.02634
5.00000	200	-43.26453
5.00000	500	-43.16043
10.00000	10	-51.08469
10.00000	50	-45.22951
10.00000	100	-43.82062
10.00000	200	-43.10559
10.00000	500	-43.09334

Tabella 92: dataset: audio. GRASP con rumore avente varianza di 2, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-63.87992
0.50000	50	-49.57349
0.50000	100	-45.50533
0.50000	200	-43.93398
0.50000	500	-43.28222
1.00000	10	-60.32249
1.00000	50	-48.70213
1.00000	100	-45.39570
1.00000	200	-43.84778
1.00000	500	-43.31314
5.00000	10	-53.38256
5.00000	50	-46.01039
5.00000	100	-44.19040
5.00000	200	-43.17796
5.00000	500	-43.24012
10.00000	10	-50.47683
10.00000	50	-44.92644
10.00000	100	-43.74475
10.00000	200	-43.24427
10.00000	500	-43.22367

5.3.5 jester

Tabella 93: dataset: jester. Nessun algoritmo di SLS applicato

alpha	minst	test_ll
0.50000	10	-125.47147
0.50000	50	-77.60039
0.50000	100	-64.07435
0.50000	200	-58.14350
0.50000	500	-55.66913
1.00000	10	-115.37348
1.00000	50	-74.78563
1.00000	100	-63.12024
1.00000	200	-57.88698
1.00000	500	-55.63026

5.00000	10	-92.69368
5.00000	50	-67.95320
5.00000	100	-60.18760
5.00000	200	-57.07277
5.00000	500	-55.51808
10.00000	10	-82.25151
10.00000	50	-64.53820
10.00000	100	-59.55639
10.00000	200	-56.68866
10.00000	500	-55.36343

Tabella 94: dataset: jester. Nessun algoritmo di SLS applicato, con rumore avente varianza di 0.1

alpha	minst	test_ll
0.50000	10	-123.96596
0.50000	50	-84.74470
0.50000	100	-74.23206
0.50000	200	-60.87688
0.50000	500	-56.48170
1.00000	10	-111.48674
1.00000	50	-81.91541
1.00000	100	-70.38324
1.00000	200	-60.52336
1.00000	500	-56.53947
5.00000	10	-90.29219
5.00000	50	-72.25335
5.00000	100	-65.10541
5.00000	200	-58.63544
5.00000	500	-56.55605
10.00000	10	-80.16234
10.00000	50	-68.64548
10.00000	100	-63.17137
10.00000	200	-57.18403
10.00000	500	-56.30876

Tabella 95: dataset: jester. Nessun algoritmo di SLS applicato, con rumore avente varianza di 0.5

alpha	minst	test_ll
0.50000	10	-95.45710
0.50000	50	-70.89194
0.50000	100	-59.63240
0.50000	200	-56.55942
0.50000	500	-56.47966
1.00000	10	-88.33916
1.00000	50	-68.73705
1.00000	100	-58.84856
1.00000	200	-56.44747
1.00000	500	-56.42325
5.00000	10	-74.82425
5.00000	50	-63.30852
5.00000	100	-57.67745
5.00000	200	-56.27756
5.00000	500	-56.31981
10.00000	10	-68.64321
10.00000	50	-60.80596
10.00000	100	-57.04721
10.00000	200	-56.18231
10.00000	500	-56.31387

Tabella 96: dataset: jester. Nessun algoritmo di SLS applicato, con rumore avente varianza di 1

alpha	minst	test_ll
0.50000	10	-86.95881
0.50000	50	-64.29676
0.50000	100	-58.11172
0.50000	200	-56.38537
0.50000	500	-56.31009
1.00000	10	-82.26095
1.00000	50	-63.08243
1.00000	100	-57.49093
1.00000	200	-56.43339
1.00000	500	-56.46230
5.00000	10	-71.09183

5.00000	50	-59.39682
5.00000	100	-57.01868
5.00000	200	-56.25458
5.00000	500	-56.32458
10.00000	10	-67.01140
10.00000	50	-58.71985
10.00000	100	-56.75743
10.00000	200	-56.17670
10.00000	500	-56.35394

Tabella 97: dataset: jester. Nessun algoritmo di SLS applicato, con rumore avente varianza di 2

alpha	minst	test_ll
0.50000	10	-84.54932
0.50000	50	-62.71570
0.50000	100	-57.72094
0.50000	200	-56.43870
0.50000	500	-56.38441
1.00000	10	-79.28506
1.00000	50	-61.56495
1.00000	100	-57.42628
1.00000	200	-56.36929
1.00000	500	-56.37674
5.00000	10	-69.47289
5.00000	50	-59.71290
5.00000	100	-56.95557
5.00000	200	-56.18646
5.00000	500	-56.49626
10.00000	10	-65.84738
10.00000	50	-58.50574
10.00000	100	-56.68976
10.00000	200	-56.24749
10.00000	500	-56.44242

Tabella 98: dataset: jester. Iterative Improvement senza rumore

alpha	minst	test_ll

0.50000	10	-116.32133
0.50000	50	-77.56757
0.50000	100	-64.07435
0.50000	200	-58.14350
0.50000	500	-55.66913
1.00000	10	-107.46736
1.00000	50	-74.76065
1.00000	100	-63.12024
1.00000	200	-57.88698
1.00000	500	-55.63026
5.00000	10	-87.68040
5.00000	50	-67.93596
5.00000	100	-60.18760
5.00000	200	-57.07277
5.00000	500	-55.51808
10.00000	10	-78.75370
10.00000	50	-64.50606
10.00000	100	-59.55639
10.00000	200	-56.68866
10.00000	500	-55.36343

Tabella 99: dataset: jester. Iterative Improvement con rumore avente varianza di 0.1

alpha	minst	test_ll
0.50000	10	-116.67330
0.50000	50	-84.69322
0.50000	100	-74.23084
0.50000	200	-60.87688
0.50000	500	-56.48170
1.00000	10	-104.89657
1.00000	50	-81.89254
1.00000	100	-70.38287
1.00000	200	-60.52336
1.00000	500	-56.53947
5.00000	10	-86.64506
5.00000	50	-72.24410
5.00000	100	-65.10541
5.00000	200	-58.63544

5.00000	500	-56.55605
10.00000	10	-77.73502
10.00000	50	-68.63968
10.00000	100	-63.17137
10.00000	200	-57.18403
10.00000	500	-56.30876

Tabella 100: dataset: jester. Iterative Improvement con rumore avente varianza di 0.5

alpha	minst	test_ll
0.50000	10	-93.11098
0.50000	50	-70.87476
0.50000	100	-59.63240
0.50000	200	-56.55942
0.50000	500	-56.47966
1.00000	10	-86.60999
1.00000	50	-68.72542
1.00000	100	-58.84856
1.00000	200	-56.44747
1.00000	500	-56.42325
5.00000	10	-73.82940
5.00000	50	-63.30679
5.00000	100	-57.67745
5.00000	200	-56.27756
5.00000	500	-56.31981
10.00000	10	-67.80168
10.00000	50	-60.80318
10.00000	100	-57.04721
10.00000	200	-56.18231
10.00000	500	-56.31387

Tabella 101: dataset: jester. Iterative Improvement con rumore avente varianza di 1

alpha	minst	test_ll
0.50000	10	-84.48067
0.50000	50	-64.27972

0.50000	100	-58.11172
0.50000	200	-56.38537
0.50000	500	-56.31009
1.00000	10	-80.36819
1.00000	50	-63.07313
1.00000	100	-57.49093
1.00000	200	-56.43339
1.00000	500	-56.46230
5.00000	10	-70.09661
5.00000	50	-59.39682
5.00000	100	-57.01868
5.00000	200	-56.25458
5.00000	500	-56.32458
10.00000	10	-66.19346
10.00000	50	-58.71940
10.00000	100	-56.75743
10.00000	200	-56.17670
10.00000	500	-56.35394

Tabella 102: dataset: jester. Iterative Improvement con rumore avente varianza di 2

alpha	minst	test_ll
0.50000	10	-82.26377
0.50000	50	-62.71403
0.50000	100	-57.72094
0.50000	200	-56.43870
0.50000	500	-56.38441
1.00000	10	-77.16463
1.00000	50	-61.55057
1.00000	100	-57.42628
1.00000	200	-56.36929
1.00000	500	-56.37674
5.00000	10	-68.37132
5.00000	50	-59.70823
5.00000	100	-56.95557
5.00000	200	-56.18646
5.00000	500	-56.49626
10.00000	10	-65.03417

10.00000	50	-58.50098
10.00000	100	-56.68976
10.00000	200	-56.24749
10.00000	500	-56.44242

Tabella 103: dataset: jester. Randomised Iterative Improvement senza rumore, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-123.20239
0.50000	50	-77.58687
0.50000	100	-64.07435
0.50000	200	-58.14350
0.50000	500	-55.66913
1.00000	10	-113.48656
1.00000	50	-74.77720
1.00000	100	-63.12024
1.00000	200	-57.88698
1.00000	500	-55.63026
5.00000	10	-91.41547
5.00000	50	-67.95082
5.00000	100	-60.18760
5.00000	200	-57.07277
5.00000	500	-55.51808
10.00000	10	-81.39364
10.00000	50	-64.53512
10.00000	100	-59.55639
10.00000	200	-56.68866
10.00000	500	-55.36343

Tabella 104: dataset: jester. Randomised Iterative Improvement con rumore avente varianza di 0.1, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-121.74964
0.50000	50	-84.73066
0.50000	100	-74.23188
0.50000	200	-60.87688

0.50000	500	-56.48170
1.00000	10	-109.55391
1.00000	50	-81.91289
1.00000	100	-70.38241
1.00000	200	-60.52336
1.00000	500	-56.53947
5.00000	10	-89.16471
5.00000	50	-72.25139
5.00000	100	-65.10541
5.00000	200	-58.63544
5.00000	500	-56.55605
10.00000	10	-79.34948
10.00000	50	-68.64256
10.00000	100	-63.17137
10.00000	200	-57.18403
10.00000	500	-56.30876

Tabella 105: dataset: jester. Randomised Iterative Improvement con rumore avente varianza di 0.5, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-94.61148
0.50000	50	-70.88654
0.50000	100	-59.63240
0.50000	200	-56.55942
0.50000	500	-56.47966
1.00000	10	-87.64976
1.00000	50	-68.73406
1.00000	100	-58.84856
1.00000	200	-56.44747
1.00000	500	-56.42325
5.00000	10	-74.39909
5.00000	50	-63.30869
5.00000	100	-57.67745
5.00000	200	-56.27756
5.00000	500	-56.31981
10.00000	10	-68.27235
10.00000	50	-60.80655
10.00000	100	-57.04721

10.00000	200	-56.18231
10.00000	500	-56.31387

Tabella 106: dataset: jester. Randomised Iterative Improvement con rumore avente varianza di 1, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-86.05713
0.50000	50	-64.28896
0.50000	100	-58.11172
0.50000	200	-56.38537
0.50000	500	-56.31009
1.00000	10	-81.53795
1.00000	50	-63.08068
1.00000	100	-57.49093
1.00000	200	-56.43339
1.00000	500	-56.46230
5.00000	10	-70.68207
5.00000	50	-59.39682
5.00000	100	-57.01868
5.00000	200	-56.25458
5.00000	500	-56.32458
10.00000	10	-66.66028
10.00000	50	-58.71972
10.00000	100	-56.75743
10.00000	200	-56.17670
10.00000	500	-56.35394

Tabella 107: dataset: jester. Randomised Iterative Improvement con rumore avente varianza di 2, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-83.75635
0.50000	50	-62.71507
0.50000	100	-57.72094
0.50000	200	-56.43870
0.50000	500	-56.38441
1.00000	10	-78.46292

1.00000	50	-61.56375
1.00000	100	-57.42628
1.00000	200	-56.36929
1.00000	500	-56.37674
5.00000	10	-69.01856
5.00000	50	-59.71177
5.00000	100	-56.95557
5.00000	200	-56.18646
5.00000	500	-56.49626
10.00000	10	-65.50718
10.00000	50	-58.50331
10.00000	100	-56.68976
10.00000	200	-56.24749
10.00000	500	-56.44242

Tabella 108: dataset: jester. GRASP senza rumore, k=2 e times=3

alpha	minst	test_ll
0.50000	10	-116.32133
0.50000	50	-77.56757
0.50000	100	-64.07435
0.50000	200	-58.14350
0.50000	500	-55.66913
1.00000	10	-107.46736
1.00000	50	-74.76065
1.00000	100	-63.12024
1.00000	200	-57.88698
1.00000	500	-55.63026
5.00000	10	-87.68040
5.00000	50	-67.93596
5.00000	100	-60.18760
5.00000	200	-57.07277
5.00000	500	-55.51808
10.00000	10	-78.75370
10.00000	50	-64.50606
10.00000	100	-59.55639
10.00000	200	-56.68866
10.00000	500	-55.36343

Tabella 109: dataset: jester. GRASP con rumore avente varianza di 0.1, k=2 e times=3

alpha	minst	test_ll
0.50000	10	-117.10172
0.50000	50	-87.07865
0.50000	100	-73.89528
0.50000	200	-61.22163
0.50000	500	-56.36799
1.00000	10	-108.89930
1.00000	50	-81.59752
1.00000	100	-70.55376
1.00000	200	-60.03375
1.00000	500	-56.46136
5.00000	10	-87.65345
5.00000	50	-72.31113
5.00000	100	-64.67077
5.00000	200	-58.25021
5.00000	500	-56.49393
10.00000	10	-77.79056
10.00000	50	-68.45883
10.00000	100	-62.33121
10.00000	200	-57.46528
10.00000	500	-56.28735

Tabella 110: dataset: jester. GRASP con rumore avente varianza di 0.5, k=2 e times=3

alpha	minst	test_ll
0.50000	10	-92.81448
0.50000	50	-71.13757
0.50000	100	-59.64419
0.50000	200	-56.54086
0.50000	500	-56.42753
1.00000	10	-87.12938
1.00000	50	-70.16060
1.00000	100	-59.12976
1.00000	200	-56.51973
1.00000	500	-56.38371
5.00000	10	-73.11728

5.00000	50	-63.04958
5.00000	100	-57.92319
5.00000	200	-56.22606
5.00000	500	-56.28537
10.00000	10	-68.55255
10.00000	50	-59.86038
10.00000	100	-57.24748
10.00000	200	-56.14986
10.00000	500	-56.35530

Tabella 111: dataset: jester. GRASP con rumore avente varianza di 1, k=2 e times=3

alpha	minst	test_ll
0.50000	10	-82.70918
0.50000	50	-63.58495
0.50000	100	-58.36002
0.50000	200	-56.35401
0.50000	500	-56.50172
1.00000	10	-80.25192
1.00000	50	-63.41697
1.00000	100	-57.79841
1.00000	200	-56.50708
1.00000	500	-56.46392
5.00000	10	-69.65531
5.00000	50	-59.85005
5.00000	100	-57.24598
5.00000	200	-56.22508
5.00000	500	-56.39747
10.00000	10	-65.73489
10.00000	50	-58.74664
10.00000	100	-56.72399
10.00000	200	-56.13675
10.00000	500	-56.22782

Tabella 112: dataset: jester. GRASP con rumore avente varianza di 2, k=2 e times=3

alpha	minst	test_ll
0.50000	10	-80.63601
0.50000	50	-62.54936
0.50000	100	-57.83377
0.50000	200	-56.54435
0.50000	500	-56.40681
1.00000	10	-77.47129
1.00000	50	-61.98214
1.00000	100	-57.53426
1.00000	200	-56.36805
1.00000	500	-56.46247
5.00000	10	-67.76990
5.00000	50	-59.16688
5.00000	100	-57.01176
5.00000	200	-56.23598
5.00000	500	-56.38628
10.00000	10	-64.95266
10.00000	50	-58.34480
10.00000	100	-56.56797
10.00000	200	-56.15862
10.00000	500	-56.25609

5.3.6 accidents

Tabella 113: dataset: accidents. Nessun algoritmo di SLS applicato

alpha	minst	test_ll
0.5	10	-62.15722
0.5	50	-41.48072
0.5	100	-34.8115
0.5	200	-31.66529
0.5	500	-30.07461
1	10	-57.77949
1	50	-40.02941
1	100	-34.13259
1	200	-31.37831
1	500	-29.99737
5	10	-47.56462
5	50	-36.11655

5	100	-32.39425
5	200	-30.58739
5	500	-29.86769
10	10	-44.0421
10	50	-34.65678
10	100	-31.74568
10	200	-30.40374
10	500	-29.87294

Tabella 114: dataset: accidents. Nessun algoritmo di SLS applicato, rumore con varianza di 0.1

alpha	minst	test_ll
0.5	10	-63.24715
0.5	50	-42.83256
0.5	100	-36.76239
0.5	200	-32.63634
0.5	500	-32.08839
1	10	-57.64453
1	50	-40.30993
1	100	-35.31977
1	200	-32.99985
1	500	-32.22128
5	10	-47.17704
5	50	-36.70782
5	100	-34.22309
5	200	-32.39974
5	500	-32.16439
10	10	-40.61994
10	50	-35.6842
10	100	-33.35076
10	200	-32.34451
10	500	-32.32546

Tabella 115: dataset: accidents. Nessun algoritmo di SLS applicato, rumore con varianza di 0.5

alpha	minst	test_ll

0.5	10	-48.92708
0.5	50	-38.82337
0.5	100	-37.0477
0.5	200	-36.81411
0.5	500	-36.57223
1	10	-43.76265
1	50	-37.88438
1	100	-36.70673
1	200	-37.22731
1	500	-37.03811
5	10	-40.7394
5	50	-37.74227
5	100	-38.57633
5	200	-37.23902
5	500	-36.61287
10	10	-38.85897
10	50	-41.63151
10	100	-38.23747
10	200	-35.87211
10	500	-38.46156

Tabella 116: dataset: accidents. Nessun algoritmo di SLS applicato, rumore con varianza di 1

alpha	minst	test_ll
0.5	10	-45.89425
0.5	50	-40.87472
0.5	100	-36.99941
0.5	200	-36.80616
0.5	500	-37.64417
1	10	-45.50123
1	50	-38.30319
1	100	-36.81792
1	200	-36.14623
1	500	-36.94757
5	10	-39.49768
5	50	-36.83786
5	100	-38.45275
5	200	-36.63342

5	500	-36.98186
10	10	-39.79604
10	50	-36.35389
10	100	-36.68787
10	200	-36.51541
10	500	-39.22734

Tabella 117: dataset: accidents. Nessun algoritmo di SLS applicato, rumore con varianza di 2

alpha	minst	test_ll
0.5	10	-46.95516
0.5	50	-38.15991
0.5	100	-36.80423
0.5	200	-36.24913
0.5	500	-36.8532
1	10	-42.19005
1	50	-37.86626
1	100	-36.5232
1	200	-36.38709
1	500	-36.81791
5	10	-40.3583
5	50	-36.60563
5	100	-39.77466
5	200	-36.11985
5	500	-37.39663
10	10	-38.93062
10	50	-36.13608
10	100	-36.02575
10	200	-36.78881
10	500	-37.05795

Tabella 118: dataset: accidents. Iterative Improvement senza rumore.

alpha	minst	test_ll
0.5	10	-59.55557
0.5	50	-41.43517
0.5	100	-34.8115

0.5	200	-31.66529
0.5	500	-30.07461
1	10	-55.57526
1	50	-39.99336
1	100	-34.13259
1	200	-31.37831
1	500	-29.99737
5	10	-46.25857
5	50	-36.11375
5	100	-32.39425
5	200	-30.58739
5	500	-29.86769
10	10	-43.00602
10	50	-34.65808
10	100	-31.74568
10	200	-30.40374
10	500	-29.87294

Tabella 119: dataset: accidents. Iterative Improvement con rumore avente varianza di 0.1.

alpha	minst	test_ll
0.5	10	-61.42059
0.5	50	-42.82839
0.5	100	-36.76239
0.5	200	-32.6298
0.5	500	-32.08839
1	10	-55.92635
1	50	-40.31144
1	100	-35.31977
1	200	-32.99985
1	500	-32.22128
5	10	-46.30341
5	50	-36.69938
5	100	-34.22222
5	200	-32.39974
5	500	-32.16439
10	10	-40.17745
10	50	-35.68809

10	100	-33.35076
10	200	-32.34451
10	500	-32.33111

Tabella 120: dataset: accidents. Iterative Improvement con rumore avente varianza di 0.5.

alpha	minst	test_ll
0.5	10	-48.23628
0.5	50	-38.82527
0.5	100	-37.04328
0.5	200	-36.80303
0.5	500	-36.56383
1	10	-43.28092
1	50	-37.88003
1	100	-36.69973
1	200	-37.2429
1	500	-37.03816
5	10	-40.52503
5	50	-37.74671
5	100	-38.57674
5	200	-37.24174
5	500	-36.61234
10	10	-38.81693
10	50	-41.6321
10	100	-38.23583
10	200	-35.87213
10	500	-38.47142

Tabella 121: dataset: accidents. Iterative Improvement con rumore avente varianza di 1.

alpha	minst	test_ll
0.5	10	-45.28576
0.5	50	-40.88067
0.5	100	-36.99741
0.5	200	-36.80462
0.5	500	-37.64433

1	10	-44.90638
1	50	-38.30141
1	100	-36.82177
1	200	-36.1385
1	500	-36.94913
5	10	-39.30069
5	50	-36.83263
5	100	-38.45021
5	200	-36.63443
5	500	-36.98315
10	10	-39.61138
10	50	-36.34962
10	100	-36.68793
10	200	-36.51427
10	500	-39.23051

Tabella 122: dataset: accidents. Iterative Improvement con rumore avente varianza di 2.

alpha	minst	test_ll
0.5	10	-46.23398
0.5	50	-38.15193
0.5	100	-36.80296
0.5	200	-36.24463
0.5	500	-36.85057
1	10	-41.84649
1	50	-37.86544
1	100	-36.5232
1	200	-36.38577
1	500	-36.81791
5	10	-40.10899
5	50	-36.60217
5	100	-39.77575
5	200	-36.12266
5	500	-37.39628
10	10	-38.68157
10	50	-36.13497
10	100	-36.02559
10	200	-36.79904

10	500	-37.05795

Tabella 123: dataset: accidents. Randomised Iterative Improvement senza rumore, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-60.83934
0.50000	50	-41.46811
0.50000	100	-34.81150
0.50000	200	-31.66529
0.50000	500	-30.07461
1.00000	10	-56.61832
1.00000	50	-40.01901
1.00000	100	-34.13259
1.00000	200	-31.37831
1.00000	500	-29.99737
5.00000	10	-46.90489
5.00000	50	-36.11715
5.00000	100	-32.39425
5.00000	200	-30.58739
5.00000	500	-29.86769
10.00000	10	-43.51441
10.00000	50	-34.65816
10.00000	100	-31.74568
10.00000	200	-30.40374
10.00000	500	-29.87294

Tabella 124: dataset: accidents. Randomised Iterative Improvement con rumore avente 0.1 di varianza, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-62.24658
0.50000	50	-42.83205
0.50000	100	-36.76239
0.50000	200	-32.62977
0.50000	500	-32.08839
1.00000	10	-56.69594
1.00000	50	-40.31181

1.00000	100	-35.31977
1.00000	200	-32.99985
1.00000	500	-32.22128
5.00000	10	-46.72341
5.00000	50	-36.71141
5.00000	100	-34.22311
5.00000	200	-32.39974
5.00000	500	-32.16439
10.00000	10	-40.36175
10.00000	50	-35.69038
10.00000	100	-33.35076
10.00000	200	-32.34451
10.00000	500	-32.32418

Tabella 125: dataset: accidents. Randomised Iterative Improvement con rumore avente 0.5 di varianza, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-48.50305
0.50000	50	-38.82681
0.50000	100	-37.04483
0.50000	200	-36.80748
0.50000	500	-36.56031
1.00000	10	-43.49345
1.00000	50	-37.88454
1.00000	100	-36.70702
1.00000	200	-37.23132
1.00000	500	-37.03785
5.00000	10	-40.62816
5.00000	50	-37.74194
5.00000	100	-38.57267
5.00000	200	-37.23877
5.00000	500	-36.61554
10.00000	10	-38.83921
10.00000	50	-41.63274
10.00000	100	-38.23947
10.00000	200	-35.87381
10.00000	500	-38.46426

Tabella 126: dataset: accidents. Randomised Iterative Improvement con rumore avente 1 di varianza, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-45.53200
0.50000	50	-40.87799
0.50000	100	-36.99684
0.50000	200	-36.80544
0.50000	500	-37.64447
1.00000	10	-45.09297
1.00000	50	-38.30121
1.00000	100	-36.81693
1.00000	200	-36.14076
1.00000	500	-36.94905
5.00000	10	-39.40048
5.00000	50	-36.83562
5.00000	100	-38.45222
5.00000	200	-36.63333
5.00000	500	-36.98179
10.00000	10	-39.69391
10.00000	50	-36.35079
10.00000	100	-36.68908
10.00000	200	-36.51637
10.00000	500	-39.23095

Tabella 127: dataset: accidents. Randomised Iterative Improvement con rumore avente 2 di varianza, probabilità di 0.6 e times=10

alpha	minst	test_ll
0.50000	10	-46.53237
0.50000	50	-38.15916
0.50000	100	-36.80448
0.50000	200	-36.24695
0.50000	500	-36.85022
1.00000	10	-41.98584
1.00000	50	-37.86619
1.00000	100	-36.52320
1.00000	200	-36.38285
1.00000	500	-36.81791
5.00000	10	-40.20916

5.00000	50	-36.60366
5.00000	100	-39.77520
5.00000	200	-36.12183
5.00000	500	-37.39627
10.00000	10	-38.78225
10.00000	50	-36.13419
10.00000	100	-36.02577
10.00000	200	-36.78771
10.00000	500	-37.05795

Tabella 128: dataset: accidents. GRASP senza rumore, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-59.55557
0.50000	50	-41.43517
0.50000	100	-34.81150
0.50000	200	-31.66529
0.50000	500	-30.07461
1.00000	10	-55.57526
1.00000	50	-39.99336
1.00000	100	-34.13259
1.00000	200	-31.37831
1.00000	500	-29.99737
5.00000	10	-46.25857
5.00000	50	-36.11375
5.00000	100	-32.39425
5.00000	200	-30.58739
5.00000	500	-29.86769
10.00000	10	-43.00602
10.00000	50	-34.65808
10.00000	100	-31.74568
10.00000	200	-30.40374
10.00000	500	-29.87294

Tabella 129: dataset: accidents. GRASP con rumore avente varianza di 0.1, k=2 e times=5

alpha	minst	test_ll

0.50000	10	-60.19237
0.50000	50	-42.19728
0.50000	100	-36.02921
0.50000	200	-33.44082
0.50000	500	-32.03720
1.00000	10	-53.05862
1.00000	50	-40.69411
1.00000	100	-35.71858
1.00000	200	-32.73453
1.00000	500	-31.96624
5.00000	10	-46.54484
5.00000	50	-37.49317
5.00000	100	-33.90308
5.00000	200	-32.28769
5.00000	500	-32.45413
10.00000	10	-42.34078
10.00000	50	-35.20659
10.00000	100	-33.12773
10.00000	200	-32.34099
10.00000	500	-32.17970

Tabella 130: dataset: accidents. GRASP con rumore avente varianza di 0.5, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-43.23682
0.50000	50	-39.00616
0.50000	100	-37.04375
0.50000	200	-36.29167
0.50000	500	-36.39501
1.00000	10	-40.89193
1.00000	50	-38.27758
1.00000	100	-37.71308
1.00000	200	-36.17050
1.00000	500	-37.53471
5.00000	10	-40.01585
5.00000	50	-37.36379
5.00000	100	-36.05226
5.00000	200	-37.22279

5.00000	500	-36.68760
10.00000	10	-38.35159
10.00000	50	-36.65296
10.00000	100	-36.50834
10.00000	200	-36.38754
10.00000	500	-37.63893

Tabella 131: dataset: accidents. GRASP con rumore avente varianza di 1, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-46.69148
0.50000	50	-39.09587
0.50000	100	-36.91035
0.50000	200	-36.47041
0.50000	500	-37.17706
1.00000	10	-45.45079
1.00000	50	-38.04120
1.00000	100	-36.47830
1.00000	200	-36.34505
1.00000	500	-37.14244
5.00000	10	-41.04134
5.00000	50	-40.24065
5.00000	100	-37.59176
5.00000	200	-35.70818
5.00000	500	-37.15097
10.00000	10	-39.38945
10.00000	50	-38.38552
10.00000	100	-37.20100
10.00000	200	-36.84360
10.00000	500	-37.17689

Tabella 132: dataset: accidents. GRASP con rumore avente varianza di 2, k=2 e times=5

alpha	minst	test_ll
0.50000	10	-46.38060
0.50000	50	-38.60307

0.50000	100	-38.18711
0.50000	200	-36.35354
0.50000	500	-36.87568
1.00000	10	-44.39325
1.00000	50	-37.54959
1.00000	100	-36.45150
1.00000	200	-36.02602
1.00000	500	-36.84540
5.00000	10	-40.62676
5.00000	50	-36.63745
5.00000	100	-35.80353
5.00000	200	-36.54389
5.00000	500	-36.56929
10.00000	10	-39.07415
10.00000	50	-36.28810
10.00000	100	-40.96060
10.00000	200	-36.19343
10.00000	500	-37.88981

5.4 Discussione

Questo capitolo racchiude tutte le considerazioni e le discussioni riguardo i risultati precedentemente inseriti. Ogni sotto-capitolo contiene la discussione dei risultati per ogni algoritmo.

5.4.1 Versione originale di base

Prima di applicare gli algoritmi di ricerca al problema sono state eseguite delle grid searches con l'algoritmo originale.
I risultati ottenuti sono stati utilizzati come base con cui comparare eventuali miglioramenti prodotti dagli algoritmi di ricerca e dalle modifiche presentate in 4.1 e 4.2.

Si è subito notato come l'aggiunta di rumore alla matrice delle mutue informazioni abbia avuto il medesimo impatto per ogni dataset testato. In particolare si è notato come l'aggiunta di rumore causi un miglioramento della log-likelihood solamente quando si utilizza un valore di δ piccolo; ovvero quando si termina la ricerca di un nodo, su cui splittare, a causa di un numero piccolo di istanze rimanenti.

L'aggiunta di rumore peggiora la likelihood rispetto al modello senza aggiunta di rumore quando si utilizza un δ grande. Ciò è probabilmente

dovuto al fatto che avere un δ grande porta il modello a "vedere" meno istanze e quindi ad adattarsi in maniera peggiore alla distribuzione campione rappresentata dal training set. Avere un δ minore porta il modello a fare un pò di overfitting e quindi aggiungere del rumore lo aiuta ad evadere da un minimo locale.

La figura 12 rappresenta l'andamento del log-likelihood in funzione del parametro alpha. Ogni grafico rappresenta l'andamento del likelihood per δ uguale a 10,200,500 rispettivamente. Ogni grafico visualizza le serie di likelihood in base alla varianza scelta per il rumore avente diversi valori (0.1,0.5,1,2,3).
Si può notare come per mnist piccolo l'aggiunta del rumore abbia un impatto positivo nel likelihood. Mentre nel caso di mnist grande (500) l'aggiunta del rumore provochi una diminuzione del likelihood.
In tutti i casi l'aumentare della varianza contribuisce all'aumento del likelihood nel caso di mnist piccolo e alla diminuzione del likelihood nel caso di mnist grande.

In generale si può notare come un alpha grande permetta di avere un likelihood migliore rispetto ad alpha piccoli.

Si è notato inoltre come in caso di assenza di rumore il parametro δ influisca direttamente sull'apprendimento del modello. In particolare confrontando i vari log-likelihood misurati sul training set, validation set e test set si è potuto notare come l'utilizzo di δ grandi permetta al modello di non incorrere in overfitting.

La figura 13 rappresenta l'andamento del likelihood sulle varie partizioni del dataset nltcs, non aggiungendo rumore e non utilizzando nessun algoritmo di ricerca.
Le linee tratteggiate indicano il likelihood misurato sul training set, quelle di tipo tratto-punto quello misurato sul validation set e quelle continue quello misurato sul test set.
Ogni colore rappresenta un valore di δ:

- arancio: delta=10
- celeste: delta=50
- rosso: delta=100
- verde: delta=200
- nero: delta=500

Dal grafico emerge come l'aumentare di δ permetta al likelihood sul trai-

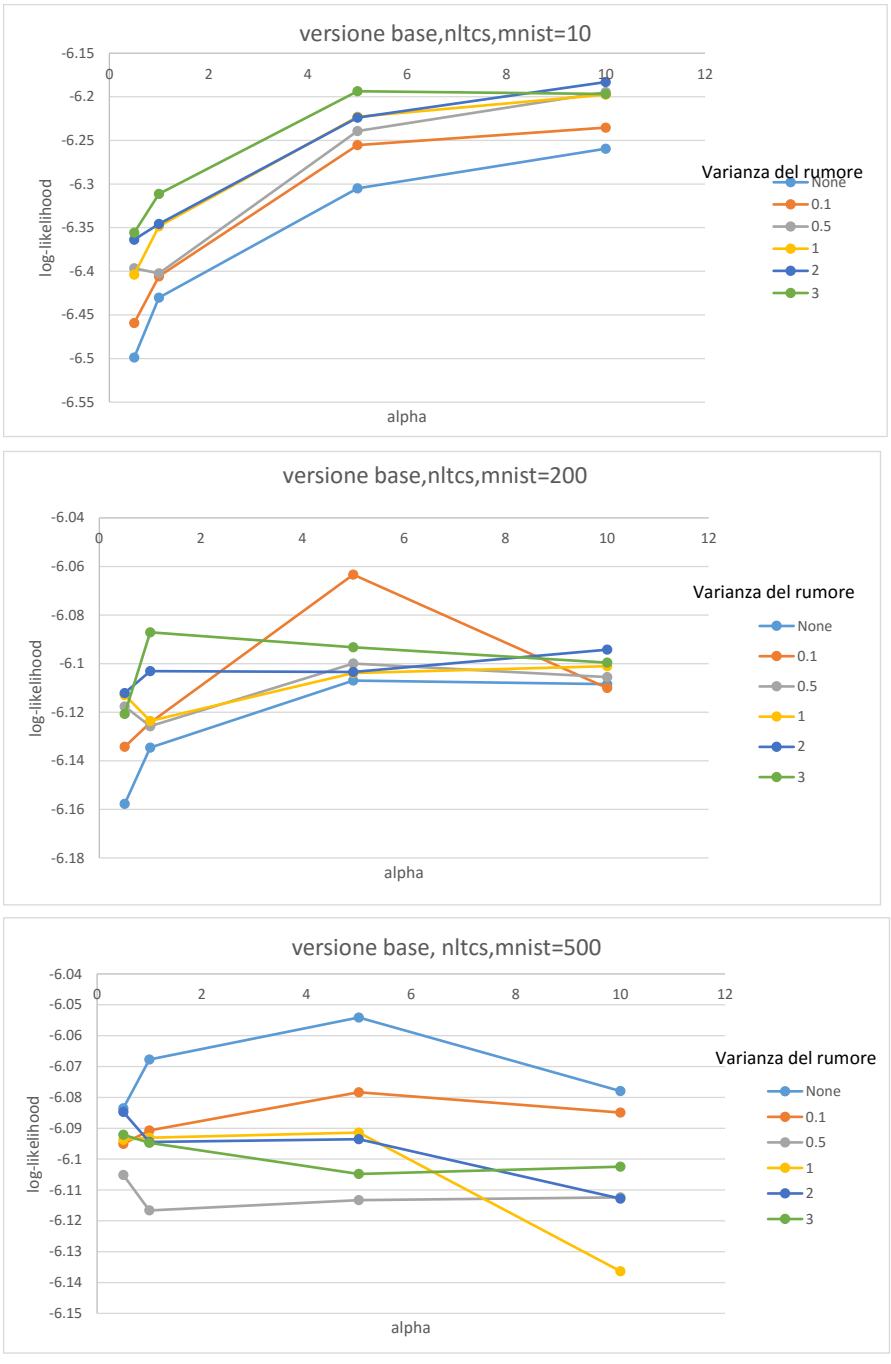

versione base,nltcs,mnist=10
log-likelihood
alpha
Varianza del rumore
None
0.1
0.5
1
2
3
versione base,nltcs,mnist=200
log-likelihood
alpha
Varianza del rumore
None
0.1
0.5
1
2
3
versione base, nltcs,mnist=500
log-likelihood
alpha
Varianza del rumore
None
0.1
0.5
1
2
3

Figura 12

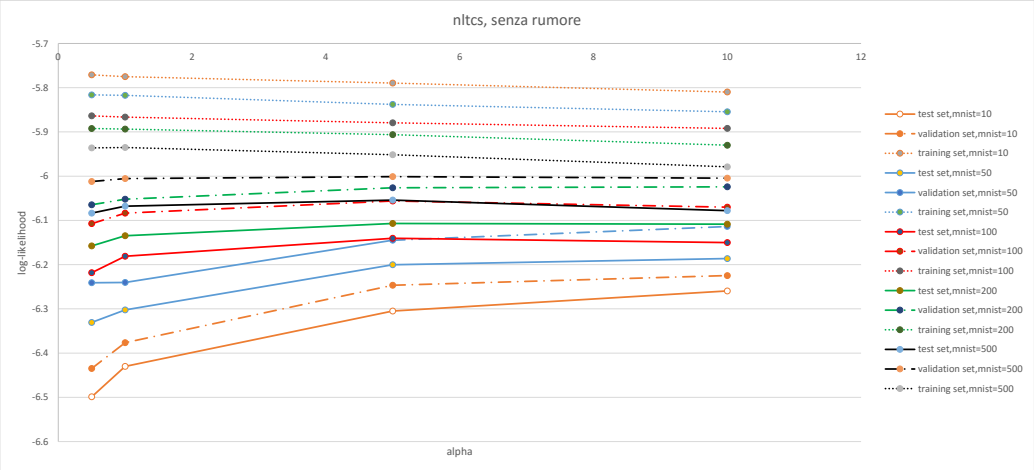


Figura 13

ning set di diminuire in favore dell'aumento dei likelihood su validation e test set. Quindi utilizzando un δ piccolo si tende a fare overfitting performando bene sul training set ma male su validation e test set.

È emerso che utilizzando un δ piccolo porta il modello a fare overfitting. Si può ovviare a questo problema aggiungendo del rumore.
Se si utilizza un δ grande allora conviene non aggiungere rumore.
Se si utilizza un delta medio allora conviene aggiungere rumore utilizzando una varianza né troppo alta e né troppo bassa.

Un altro esempio è rappresentato nella figura 14, dove il dataset plants ha prodotto il medesimo comportamento.

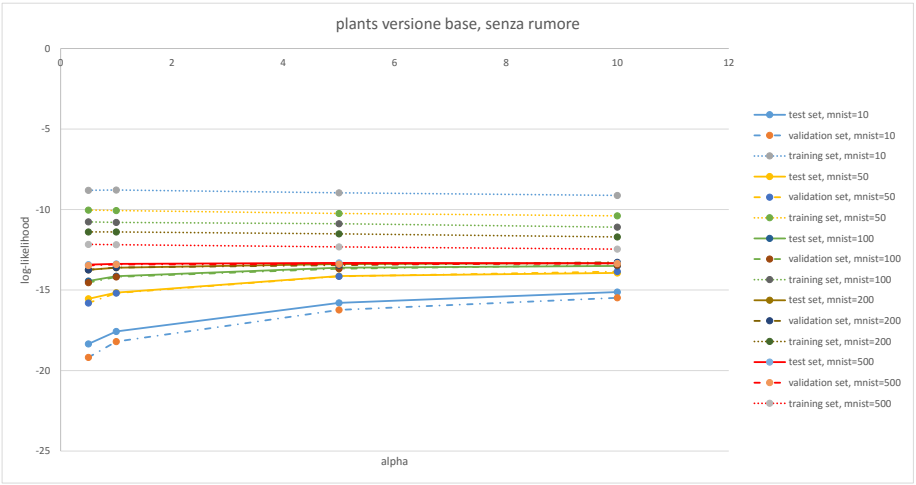


Figura 14

5.4.2 Iterative improvement

L'applicazione di Iterative Improvement al processo di apprendimento ha avuto risultati discreti. In particolare si è notato che iterative improvement riesce a migliorare la likelihood rispetto alla versione di base solamente quando δ è basso.
In generale per ogni dataset iterative improvement riesce a migliorare l'apprendimento quando δ è uguale a 10. Per valori di δ crescenti la versione base e la versione con iterative improvement raggiungono gli stessi valori di log-likelihood.

La figura 15 mostra l'andamento della likelihood, sul dataset "audio", comparando la versione di base con la versione a cui è stato applicato iterative improvement. Si può notare come utilizzando un δ piccolo, iterative improvement raggiunga risultati migliori per ogni valore di alpha. Utilizzando un δ maggiore o uguale a 100 fa si che le due serie diventino identiche, evidenziando il fatto che iterative improvement non riesce ad evadere dal minimo locale raggiunto in principio dalla versione base.

Lo stesso accade per gli altri datasets. Nella figura 16 sono riportati anche i grafici del dataset "accidents".

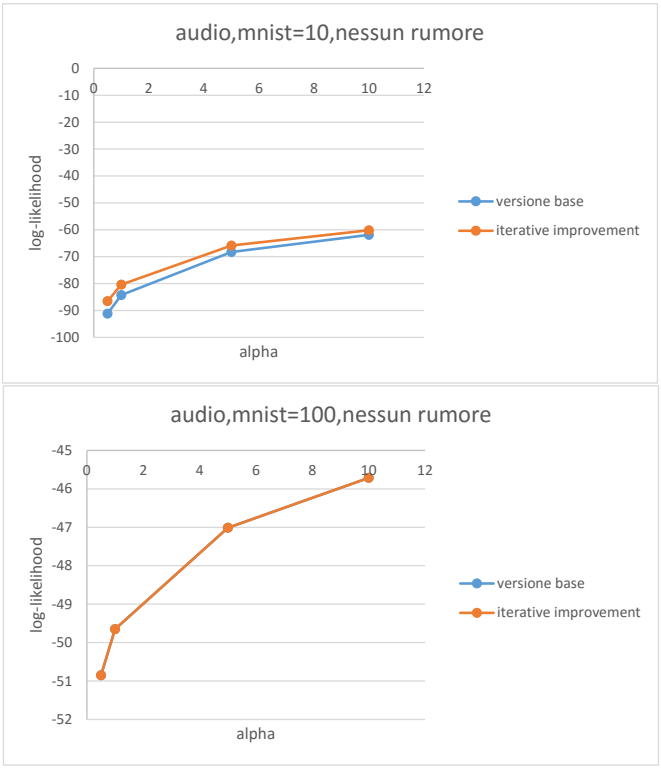
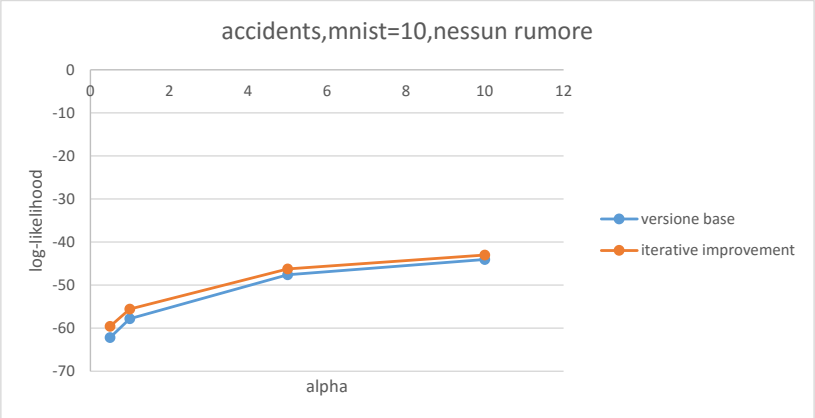

audio,mnist=10,nessun rumore
log-likelihood
alpha
versione base
iterative improvement
audio,mnist=100,nessun rumore
log-likelihood
alpha
versione base
iterative improvement

Figura 15


accidents,mnist=10,nessun rumore
0
-10
-20
-30
-40
-50
-60
-70
0
2
4
6
8
10
12
log-likelihood
alpha
versione base
iterative improvement

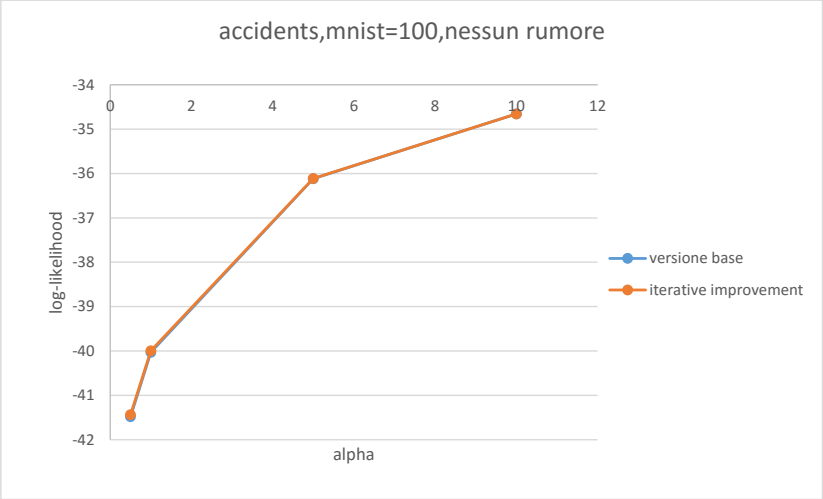

accidents,mnist=100,nessun rumore
-34
-35
-36
-37
-38
-39
-40
-41
-42
0
2
4
6
8
10
12
log-likelihood
alpha
versione base
iterative improvement

Figura 16

5.4.3 Randomised iterative improvement

Randomised Iterative improvement è stato applicato a tutti i datasets. Come parametro p, rappresentante la probabilità di scegliere una componente random dalla Restricted Candidate List, sono stati utilizzati i valori 0.6 e 0.8. Come parametro t rappresentante il numero di iterazioni massime da fare è stato utilizzato il valore 10. Tutti gli altri parametri sono stati soggetto a grid search come anticipatamente descritto.

Questa variante ha presentato gli stessi risultati dell'applicazione di Iterative Improvement. RII migliora l'adattamento del modello all'insieme di dati quando δ è basso (10-100) . Utilizziando valori di $\delta > 100$ RII raggiunge gli stessi risultati di Iterative Improvement e della versione base.

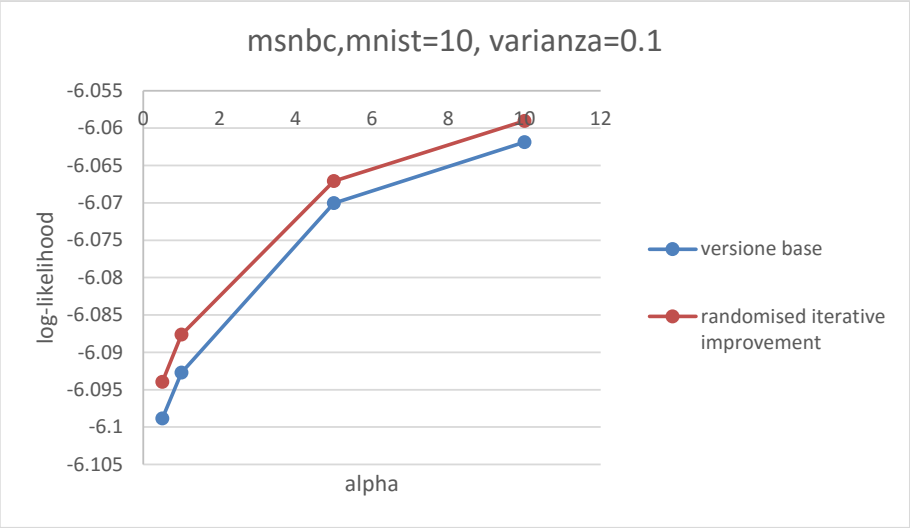


Figura 17: Grafico derivato dal dataset "msnbc"

Nella figura 17 è riportato l'andamento del likelihood con mnist piccolo e rumore avente varianza 0.1. Con mnist piccolo RII aiuta il modello ad adattarsi meglio, ma al crescere di mnist la likelihood è esattamente uguale a quella raggiunta dalla versione di base.

5.4.4 GRASP

Anche GRASP ha avuto un comportamento molto simile a quello degli altri algoritmi. Per quanto riguarda GRASP come valore di t sono stati utilizzati i seguenti valori $\{3, 5, 20\}$; essi sono stati decisi in base alla dimensione del dataset. Ad esempio sul dataset "nltcs" è stato potuto utilizzare $t = 20$, mentre per datasets più grandi sono stati utilizzati 3 e 5.

Il parametro k, rappresentante la dimensione della RCL contenente le migliori k componenti da cui scegliere, è stato scelto dall'insieme $\{2, 3\}$. Per datasets piccoli sono stati utilizzati tutti e due. Per datasets grandi come "plants" è stato utilizzato solamente il valore 3.

Anche con GRASP utilizzare un valore di δ grande ha implicato avere valori di log-likelihood non migliori della versione originale. Tuttavia l'utilizzo di valori di δ tra 10 e 100 ha evidenziato che GRASP riesce comunque ad avere risultati migliori della versione base con gli stessi parametri. Questi risultati migliori si allineano, bene o male, con i valori ottenuti dalle altre varianti algoritmiche.

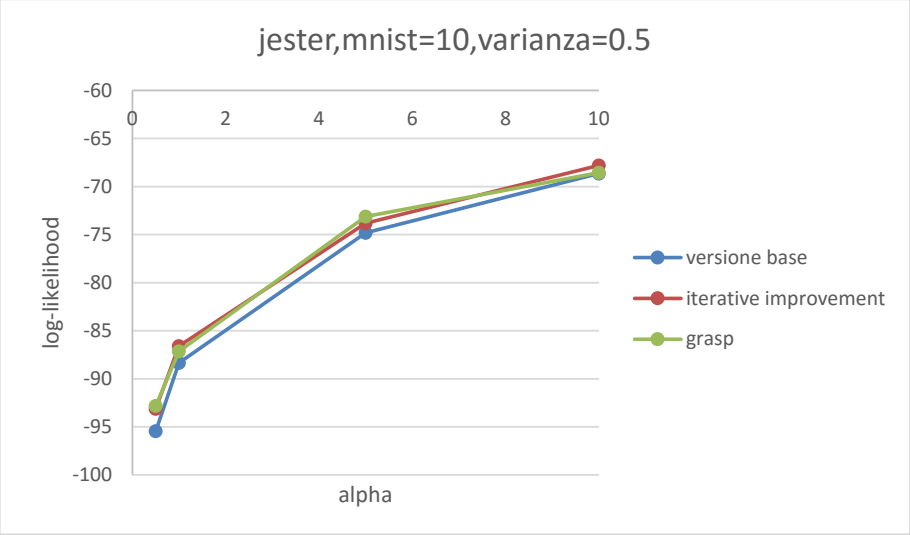


Figura 18: Grafico derivato dal dataset "jester"

La figura 18 mostra l'andamento del likelihood in funzione del parametro alpha sul dataset jester utilizzando un $\delta = 10$ e applicando rumore avente

varianza 0.5.
In quasi tutti i punti GRASP ottiene dei valori leggermente migliori rispetto alla versione base. Codesti miglioramenti sono nell'ordine di 0.5 - 1.0.

6 Conclusioni

Il compito degli algoritmi di apprendimento descritti in questa tesi è stato quello di apprendere modelli grafici probabilistici per fare density estimation sfruttando risultati consolidati negli anni come l'apprendimento di alberi di Chow-Liu.
Questa tesi ha avuto il compito di esplorare l'applicazione di algoritmi stocastici di ricerca locale all'apprendimento di Cutset Network. In particolare gli algoritmi sono stati applicati all'interno dell'algoritmo dCSN presentato in [3].

Il primo step della tesi è stato quello di presentare alcuni modelli grafici probabilistici come le reti bayesiane, gli alberi di Chow-Liu, le Cutset Networks e la tecnica delle misture.

Successivamente sono stati introdotti gli algoritmi di ricerca locale stocastica ed in particolare Iterative Improvement, Randomised Iterative Improvement e GRASP.

Poi sono state presentate alcune modifiche all'algoritmo originale di dCSN come l'aggiunta di rumore alla matrice delle mutue informazioni e l'utilizzo di foreste al posto degli alberi di Chow-Liu nelle Cutset Networks.

Dopo aver presentato i datasets su cui è stata fatta la sperimentazione, sono stati riportati i risultati in forma tabellare così come prodotti dagli script scritti in python.

Dai risultati è emerso come l'applicazione degli algoritmi di ricerca locale migliori le prestazioni dell'algoritmo originale solamente per particolari combinazioni di parametri. Per altre combinazioni di parametri è stato riscontrato che l'applicazione di algoritmi di ricerca non migliori né peggiori il log-likelihood misurato sulla partizione dedicata al test set di ogni dataset provato.
L'aggiunta di rumore alla matrice delle mutue informazioni ha avuto un riscontro positivo nel caso in cui vengano utilizzate molte osservazioni del training set per l'apprendimento della struttura. Il valore della varianza del rumore è stato direttamente proporzionale al miglioramento o peggioramento riscontrato nei risultati numerici.

Guardando in maniera generale i risultati numerici, gli algoritmi di ricerca locale non sono riusciti ad avere log-likelihood complessivamente migliori rispetto ai log-likelihood raggiunti dalla versione base di dCSN.

Ringraziamenti

Ringrazio il mio relatore, il Dr. Nicola Di Mauro. Ringrazio anche il Dr. Antonio Vergari per il continuo supporto fornitomi. Ringrazio i miei genitori e il mio cane per la compagnia fornitami durante la stesura della tesi. Ringrazio i Carbon based lifeforms, Boards of Canada, Solar Fields, Connect.Ohm e Lamb of God per aver fatto da sottofondo durante i miei studi.

Riferimenti bibliografici

[1] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory, 14−462 467, 1968

[2] Rahman, T., Kothalkar, P., Gogate, V.: Cutset networks: A simple, tractable,and scalable approach for improving the accuracy of Chow-Liu trees. In: Machine Learning and Knowledge Discovery in Databases, LNCS, vol. 8725, pp. 630−645. Springer (2014)

[3] Di Mauro, N.; Vergari, A.; and Esposito, F. 2015. Learning accurate cutset networks by exploiting decomposability. In AI*IA 2015 Advances in Artificial Intelligence, 221−232

[4] A. Onisko, M. Druzdzel, and H. Wasyluk. A Bayesian network model for diagnosis of liver disorders. In Proceedings of the 11th Conference on Biocybernetics and Biomedical Engineering, pages 842−846, 1999

[5] Hoos, H.H. and Stützle. Stochastic Local Search: Foundations and Applications, isbn 9780080498249, The Morgan Kaufmann Series in Artificial Intelligence. Elsevier Science (2004)

[6] Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)

[7] L. Breiman. Arcing classifiers. Technical report, Dept. of Statistics, University of California, 1996.

[8] François Schnitzler. Mixtures of Tree-Structured Probabilistic Graphical Models for Density Estimation in High Dimensional Spaces

[9] Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Machine learning 29(2-3), 131−163 (1997)

[10] Heckerman, D., Geiger, D., Chickering, D.: Learning bayesian networks: the combination of knowledge and statistical data. Machine learning 20, 197−243 (1995)

[11] Lowd, D., Davis, J.: Learning Markov network structure with decision trees. In: Proceedings of the 10th IEEE International Conference on Data Mining. pp. 334−343. IEEE Computer Society Press (2010)

[12] Haaren, J.V., Davis, J.: Markov network structure learning: A randomized feature generation approach. In: Proceedings of the 26th Conference on Articial Intelligence. AAAI Press (2012)

[13] Meila, M., Jordan, M.: Learning with mixtures of trees. Journal of Machine Learning Research 1, 1−48 (2000)

[14] Dechter, R., Mateescu, R.: AND/OR search spaces for graphical models. Artificial Intelligence 171(2), 73−106 (2007)

[15] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society, 7(1):48−50, 1956.

[16] S. Kullback and R. Leibler. On information and sufficiency. The Annals of Mathematical Statistics, 22(1):79−86, 1951.

[17] Christopher Bishop. Neural networks for pattern recognition. Clarendon Press-Oxford, 1995.

[18] Christopher Bishop. Pattern Recognition and Machine Learning. Springer Science-Business Media, LLC, 2006