Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Magistrale in Informatica

DTCU: Alberi Decisionali per la Classificazione di Dati Incerti

Tesi di Laurea in Basi di Dati

Relatore:
Chiar.mo Prof.
DANILO MONTESI

Presentata da:
SAMUELE CRESCENTI

Correlatore:
Dott.
MATTEO MAGNANI

III Sessione
Anno Accademico 2009-2010

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Oggi più che mai è fondamentale essere in grado di estrarre informazioni rilevanti e conoscenza dal grande numero di dati che ci possono arrivare da svariati contesti, come database collegati a satelliti e sensori automatici, repository generati dagli utenti come Flickr, YouTube e Facebook e data warehouse di grandi compagnie.

Una delle sfide attuali riguarda lo sviluppo di tecniche di data mining per la gestione dell'incertezza. L'interesse nella gestione di dati incerti è notevolmente aumentato negli ultimi anni, a causa del grande numero di ambiti in cui vengono usati questo tipo di dati. Questo settore è già stato studiato nella letteratura dei tradizionali database, ma è tornato a richiamare l'attenzione per via dei nuovi modi di collezionare ed elaborare i dati. La gestione di dati incerti porta ad una serie di sfide in termini di raggruppamento, modellazione, rappresentazione, indicizzazione, interrogazione ed estrazione dei dati. Si può inoltre notare che molti di questi problemi sono correlati e non possono semplicemente essere studiati in maniera indipendente.

L'incertezza è quasi ovunque ed è spesso causata dagli strumenti limitati che si hanno a disposizione per raccogliere dati: sensori termici, meccanici,

ottici e di altri tipi sono tutti affetti da sensibilità, che varia anche a seconda del valore della misura effettuata. In alcuni casi sono gli stessi utenti a fornire i dati e non sempre lo fanno con accuratezza: immaginiamo per esempio la pagina di registrazione ad un sito web che richiede informazioni anagrafiche, tra cui la provincia di residenza, tramite un menù a tendina. Svariati visitatori, per pigrizia o svogliatezza, selezionano semplicemente la prima opzione, che nel caso italiano è "Agrigento". Bisogna quindi gestire i dati raccolti in maniera opportuna, tenendo conto che non sempre l'informazione è totalmente affidabile: se ad esempio il gestore del sito avesse intenzione di aprire un negozio nella provincia che registra il maggior numero di utenti, farebbe bene a trattare in maniera opportuna il caso in cui la scelta vada a ricadere su Agrigento o su un'altra delle prime voci. Un altro caso applicativo reale che possiamo citare è rappresentato da situazioni in cui i dati siano stati ottenuti da processi di estrazione delle informazioni, per esempio la creazione di un profilo utente basato sulle sue identità pubbliche o la collezione di opinioni riguardanti un certo marchio. Infatti, se da un lato la persistenza dei dati generati dagli utenti sul Web permette una loro successiva ricerca e analisi rendendoli una fonte di informazioni molto utile, dall'altro li rende spesso di scarsa qualità, a causa di valori mancanti, incoerenti o errati. Come esempio possiamo considerare un dataset contente l'età e il reddito delle persone, basato sulle informazioni reperibili su Internet: pagine differenti potrebbero fornire valori differenti e alcuni valori potrebbero addirittura mancare ed essere quindi ipotizzati facendo uso di informazioni aggiuntive, come lo stile di vita e le amicizie. Altre volte l'incertezza può essere insita nella natura stessa di ciò che ci interessa: per esempio, come riuscire a stabilire esattamente quanto piaccia o meno un certo personaggio politico? Un altro esempio di applicazione reale è dato dal data mining relativo alla tutela

della privacy, dove vengono aggiunti volutamente disturbi ai dati per fare in modo di mascherare l'identità dei record; altri esempi sono dati dai problemi di integrazione dei dati tra fonti differenti, in cui una stessa informazione è registrata in maniere diverse e occorre trasformare il risultato in un formato comune. Talvolta i dati vengono istanziati usando metodi statistici come la previsione; in questi casi i dati sono incerti "alla nascita" e spesso anche probabilistici. In altre circostanze i database possono presentare *incertezza esistenziale*, cioè un record può essere presente o assente dalla base di dati.

I dati incerti sono utili perché permettono di fornire risposte probabilistiche a interrogazioni aggregate, molto frequenti in vari ambiti. Possiamo pensare ad esempio ai sondaggi elettorali, argomento particolarmente interessante ogni qualvolta ci si appresti a votare: come possiamo stimare la percentuale di persone sposate che voteranno per la destra? Tramite opportune elaborazioni dei dati incerti a disposizione, siamo in grado di fornire risposte sufficientemente accurate. Spesso rispondere a domande su dati incerti è più complicato, in quanto ci obbliga a trattare con probabilità, una dimensione nuova e molte volte difficoltosa. Tuttavia semplificare i dati incerti, riducendoli a dati certi (ad esempio eliminando le tuple che presentano incertezza o rimuovendo gli attributi incerti), non consente di sfruttare a pieno la potenzialità dei dati a disposizione, causando la perdita di dettagli utili, se non dell'intero record in questione. In aggiunta, come detto precedentemente, in alcuni ambiti avere risposte probabilistiche su dati incerti è interessante e utile.

Negli ultimi decenni sono state sviluppate numerose tecniche di analisi dei dati, raggruppate in due famiglie principali: metodi descrittivi, e.g. clustering, e metodi predittivi, come la classificazione. Recentemente sono state ideate una serie di applicazioni di data mining per il caso di dati incerti, che

comprendono anche clustering e classificazione [1, 3, 8]. Si può notare che la presenza di incertezza può influire in modo significativo sui risultati del data mining. Ad esempio, nel caso di classificazione, un attributo che ha bassa incertezza è più utile di un attributo che ha un livello più elevato di incertezza. Allo stesso modo, nel clustering, gli attributi che presentano livelli di incertezza più elevati hanno bisogno di essere trattati in modo diverso da quelli che hanno livelli di incertezza più bassi.

L'obiettivo di questa tesi è di estendere le attuali tecniche di gestione dell'incertezza, in particolare riguardanti la classificazione, in maniera tale da poter gestire incertezza anche sull'attributo di classe. Tale estensione è molto utile in quanto l'operazione di assegnamento di una classe piuttosto che di un'altra ad un record è spesso attuata come un'approssimazione, senza avere in realtà la certezza della decisione presa. Essere in grado di gestire incertezza anche su questo tipo di attributo consente una migliore rappresentazione dei dati di cui siamo in possesso, una migliore successiva classificazione e dunque ci permette di reagire in maniera più appropriata ai risultati che otteniamo.

Questa tesi è organizzata nel modo seguente: nel capitolo successivo verrà illustrato lo stato dell'arte sul data mining incerto, mostrando gli attuali algoritmi di clustering, frequent pattern mining, identificazione di anomalie e classificazione che consentono la gestione dell'incertezza. Saranno analizzati più approfonditamente gli alberi decisionali, che verranno usati come base per l'estensione illustrata nel capitolo 3: in tale capitolo saranno esposti anche l'analisi delle possibili situazioni da gestire e un frammento di pseudocodice che farà da base alla reale implementazione, descritta nel capitolo 4. In questo capitolo saranno inoltre mostrati i test sperimentali eseguiti, con le relative analisi dei risultati. Infine il capitolo 5 contiene le conclusioni e i possibili sviluppi futuri.

Capitolo 2

Stato dell'arte nel data mining

In questo capitolo verranno illustrati una serie di algoritmi di data mining attualmente esistenti per la gestione di dati incerti, riguardanti le operazioni di clustering, frequent pattern mining, identificazione di anomalie e classificazione. Verrà dedicata particolare attenzione agli alberi decisionali, che forniranno la base per l'estensione sviluppata in questa tesi.

2.1 Algoritmi di clustering per dati incerti

Il clustering, o analisi dei cluster, è un insieme di tecniche di analisi dei dati volte alla selezione e raggruppamento di elementi omogenei in un insieme di dati. Tutte le tecniche di clustering si basano sul concetto di distanza tra due elementi: la bontà delle analisi ottenute dagli algoritmi di clustering dipende molto dalla scelta della metrica, e quindi da come è calcolata la distanza. Gli algoritmi di clustering raggruppano gli elementi sulla base della loro distanza reciproca, e quindi l'appartenenza o meno ad un insieme dipende da quanto l'elemento preso in esame è distante dall'insieme stesso [7]. Un semplice esempio di clustering è dato dalla suddivisione del mercato

in sottoinsiemi distinti di clienti, in cui ogni insieme possa essere trattato utilizzando una specifica strategia di marketing.

La presenza di incertezza cambia la natura dei cluster, dal momento che incide sulla funzione di calcolo della distanza tra punti differenti.

2.1.1 Algoritmi basati su densità

Una tecnica proposta consiste nel trovare cluster basati sulla densità a partire da dati incerti. L'idea chiave in questo approccio è di calcolare in maniera efficace la distanza incerta tra oggetti la cui probabilità è specificata. La distanza "fuzzy" è definita in termini della funzione di distribuzione della distanza. Questa funzione calcola la probabilità che la distanza tra due oggetti incerti si trovi all'interno di un certo range definito dall'utente. Poniamo *d(X, Y)* la variabile aleatoria rappresentante la distanza tra X e Y. La funzione di distribuzione della distanza è formalmente definita come segue:

Definizione 2.1.1. *Siano X e Y due record incerti e sia p(X, Y) la funzione di densità della distanza tra questi due oggetti. Allora la probabilità che la distanza si trovi all'interno del range (a,b) è data dalla seguente relazione:*

$$P(a \leq d(X,Y) \leq b) = \int_a^b p(X,Y)(z)\,dz$$

Basandosi su questa tecnica e sulla funzione di densità della distanza, è stata definita la *probabilità di raggiungibilità* tra due punti. Questa definisce la probabilità che un punto sia direttamente raggiungibile da un altro tramite un percorso tale che tutti i punti che si trovano su questo cammino abbiano densità maggiore di una certa soglia. Si può notare come questa sia un'estensione del concetto di raggiungibilità deterministica definita nell'algoritmo DBSCAN. Nella versione deterministica dell'algoritmo, i punti sono raggruppati in cluster se sono raggiungibili l'un l'altro tramite un cammino

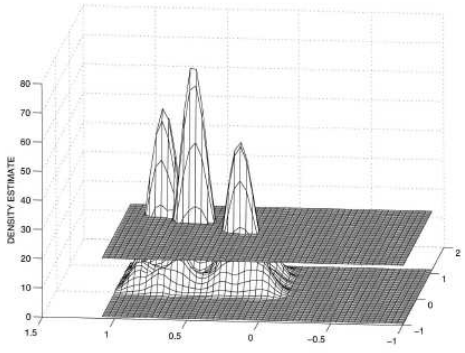

Figura 2.1: Profilo basato sulla densità, con bassa soglia di densità

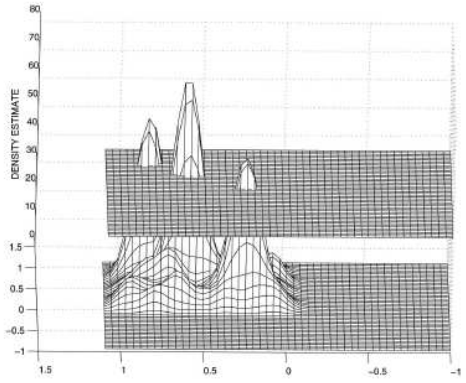

Figura 2.2: Profilo basato sulla densità, con alta soglia di densità

in cui ogni punto che vi appartiene possiede un valore di densità maggiore di una certa soglia prefissata. A tal fine l'algoritmo usa la condizione che all'interno di un certo raggio specifico (*Eps*) devono essere presenti almeno *MinPts* punti. L'algoritmo parte da un dato punto e controlla se nel raggio *Eps* sono contenuti *MinPts* punti. Se la condizione è verificata l'algoritmo ripete il controllo per ogni punto nel cluster e continua ad aggiungere punti finché possibile. Si può visualizzare il profilo di densità di un insieme di dati riportando il numero di punti interni al raggio delle varie regioni e tracciando una curva. Intuitivamente questo approccio corrisponde a quanto si può notare dalle figure 2.1 e 2.2, la prima con bassa soglia di densità e la seconda con alta soglia di densità. La soglia di densità dipende dal valore di *MinPts*. Come si può ben notare, usando una soglia più alta si ottengono 3 cluster, mentre nella versione con soglia più bassa abbiamo solo 2 diversi cluster.

La versione fuzzy dell'algoritmo DBSCAN (nota come FDBSCAN) funziona in maniera simile a DBSCAN, tranne per il fatto che la densità in un dato punto è incerta a causa dell'incertezza di base dei dati a disposizione. Questo si traduce nel fatto che il numero di punti che rientrano nel raggio

Eps di un certo punto si può solo stimare probabilisticamente ed è essenzialmente una variabile incerta; di conseguenza la raggiungibilità da un punto ad un altro non è più deterministica, proprio perché ogni punto si trova in un raggio *Eps* con una certa probabilità, che potrebbe essere minore di 1. Viene dunque aggiunto il vincolo che la probabilità di raggiungibilità calcolata sia più grande di 0.5. In questo modo si ottiene una versione particolareggiata dell'algoritmo deterministico, dove la probabilità di raggiungibilità è sempre settata a 1.

Un'altra tecnica utilizzata è quella del clustering basato su densità gerarchica: un algoritmo deterministico basato su questa tecnica è OPTICS. Si può osservare che l'idea chiave di questo algoritmo è simile a DBSCAN e si basa sul concetto di distanza raggiungibile tra punti. Mentre in DBSCAN si definisce un parametro di densità globale usato come soglia per definire la raggiungibilità, in OPTICS si tiene conto del fatto che differenti aree di dati potrebbero avere differenti densità e diventerebbe quindi impossibile definire il cluster in maniera appropriata utilizzando un solo valore di soglia. Si cerca così, tramite diversi valori di densità, di definire varie intuizioni (gerarchiche) di cluster. L'obiettivo è di ottenere un output che ordini i punti, in modo che applicando DBSCAN in quest'ordine si possa ottenere un clustering gerarchico ad ogni livello per differenti valori di densità. Il punto chiave è di assicurare che i cluster a diversi livelli della gerarchia siano coerenti tra loro. I punti vengono ordinati in base al valore di *Eps* necessario a far sì che entro il raggio compaiano almeno *MinPts*, in modo che, processando prima i punti con valori di *Eps* più bassi, avremo che le regioni ad alta densità saranno sempre definite prima delle regioni a bassa densità. Questo assicura che, applicando DBSCAN in quest'ordine, si otterrà sempre un risultato coerente. Diventa quindi facile notare che l'output di OPTICS non è una

divisione in cluster, ma un ordine in cui i punti vanno processati. Basandosi sul fatto che OPTICS presenta vari aspetti simili a DBSCAN, è triviale immaginare una versione dell'algoritmo che operi su dati incerti, utilizzando lo stesso approccio visto in precedenza con DBSCAN: questo algoritmo è noto come FOPTICS. L'unico appunto che si deve fare è che uno dei concetti chiave dell'ordinamento è il valore di *Eps* necessario a ottenere almeno *MinPts* dentro al raggio e nel caso di dati incerti questo valore sarà definito probabilisticamente [6].

2.1.2 Algoritmi UK-means e CK-means

Un approccio comune al clustering è l'algoritmo K-means. In questo algoritmo si costruiscono cluster intorno ad un predefinito numero di centroidi; esistono una varietà di funzioni per il calcolo della distanza che possono essere usate per assegnare i punti ai differenti cluster.

È stata studiata un'estensione a tale algoritmo, funzionante su dati incerti, nota come UK-means. In questo algoritmo un oggetto è assegnato al cluster il cui rappresentante è alla *distanza minima stimata*. Chiaramente un aspetto chiave è stimare la distanze tra i centroidi ed il punto in considerazione: a tal fine si ricorre spesso al metodo Monte-carlo, in cui vengono usati punti campione per stimare la distanza incerta. Questo algoritmo è però molto costoso, a causa del grande numero di campioni che possono essere richiesti per una stima della distanza accurata, perciò si usano regole di riduzione al fine di abbassare il costo computazionale. L'idea è di usare tecniche branch-and-bound per minimizzare il numero di stime di distanze; una volta stimato l'upper bound della distanza minima tra un punto e il rappresentante di un cluster, è necessario effettuare il calcolo tra questo punto e un altro rappresentate di cluster solo se può essere provato che tale distanza

è maggiore del bound. Un esempio di approccio consiste nel costruire un albero dei rappresentanti di cluster in modo da ottimizzare la strategia di riduzione e minimizzare il numero di rappresentati a cui bisogna accedere.

È stato tuttavia dimostrato che le riduzioni applicate non sempre garantiscono un effettivo abbassamento del costo computazionale. È stato ideato quindi un altro algoritmo, noto come CK-means, che propone una semplice formula per la stima delle distanze minime, in modo da ridurre notevolmente il costo. Il risultato finale è che è possibile effettuare clustering su dati incerti usando il tradizionale algoritmo k-means e ottenendo gli stessi risultati che si otterrebbero con UK-means.

2.1.3 UMicro: un algoritmo per stream di dati incerti

Recentemente il problema di clustering di dati incerti è stato esteso anche a stream di dati: l'algoritmo *UMicro* (Uncertain MICROclustering) è stato sviluppato appositamente per questo scopo. Tale procedura funziona usando un approccio iterativo, che mantiene un numero di centroidi di micro-cluster attorno a cui vengono costruiti i cluster. Uno degli input dell'algoritmo è n_{micro}, che stabilisce il numero di micro-cluster da costruire. Quando la procedura inizia non vi sono cluster presenti; al primo punto in input viene creato un nuovo cluster singleton. Per ogni successivo punto in ingresso viene determinato il centroide più vicino, tramite la stima della distanza dal punto incerto al micro-cluster incerto (per i dettagli su questo processo di stima della distanza si rimanda a [5]). Inoltre si calcola se giace entro una certa distanza incerta dal micro-cluster. Se la risposta è affermativa allora tale punto viene assegnato al micro-cluster, altrimenti è necessario creare un nuovo micro-cluster contenente solamente tale punto. Per creare un nuovo micro-cluster bisogna o aggiungerlo alla lista degli attuali oppure sostituirne

uno dei vecchi. Se siamo nelle fasi iniziali, dove il numero di micro-cluster sarà sicuramente minore di n_{micro}, basterà creare un nuovo micro-cluster e aggiungerlo alla lista degli attuali, come un cluster separato con un solo punto dentro. Altrimenti verrà rimpiazzato uno dei vecchi micro-cluster, che da specifiche è stato scelto sia sempre il meno aggiornato. Questa informazione è disponibile tramite time-stamp.

2.2 Frequent Pattern Mining su dati incerti

Per frequent pattern mining si intende l'estrazione, a partire da un insieme di transizioni, di regole che descrivano relazioni tra le variabili. Per esempio la regola {patatine, bibita} $\rightarrow$ {ketchup}, ricavata dagli scontrini di un supermercato, indica che se un cliente compra patatine e una bibita insieme, allora molto probabilmente comprerà anche il ketchup. Questo genere di informazioni è alla base di decisioni riguardanti strategie di marketing e di campagne promozionali.

Il problema dell'estrazione di pattern frequenti è stato esplorato anche nel campo dei dati incerti. In questo modello si assume che ciascun oggetto abbia un'*incertezza esistenziale* relativa all'appartenenza ad una transazione. Tramite questo approccio viene quindi gestita la probabilità che un oggetto appartenga ad una particolare transazione. Un insieme di oggetti viene definito frequente se il suo *supporto previsto* è al minimo uguale ad un valore di soglia definito dall'utente [2].

2.2.1 Algoritmo U-Apriori e tecniche di potatura

Per risolvere questa versione di estrazione di pattern frequenti può essere utilizzato l'algoritmo *U-Apriori*, un'estensione dell'algoritmo *Apriori*.

Quest'ultimo usa un approccio "genera-e-testa candidati", che ripete delle operazioni di join su itemset frequenti al fine di generare candidati con un oggetto in più rispetto alla partenza. Una proprietà chiave per la correttezza è la chiusura verso il basso, che è garantita anche nella versione estesa su dati incerti. Ciò implica che, se un pattern è frequente, cioè il suo supporto previsto è al minimo uguale al valore di soglia, allora anche tutti i sottoinsiemi del pattern sono frequenti. Ciò che differenzia U-Apriori dalla versione deterministica è il fatto che nella versione incerta si calcola il *supporto previsto* per ognuno dei diversi insiemi di oggetti. Il supporto previsto per un insieme di oggetti in una transazione si ottiene semplicemente moltiplicando le probabilità dei differenti oggetti nella transazione. Tale approccio può essere reso ulteriormente scalabile usando il concetto di trimming dei dati: gli oggetti con probabilità esistenziali molto basse vengono eliminati, lasciando il posto ai soli dati semplificati. È stato dimostrato che questo approccio può estrarre i pattern frequenti in maniera accurata mantenendo l'efficienza del metodo originale.

Sono stati anche studiati altri metodi di potatura dei dati per aumentare ulteriormente l'efficienza, tra cui uno che usa una *distribuzione bimodale* (dividendo gli oggetti in due categorie, una ad alta e una a bassa probabilità esistenziale) e uno che analizza le caratteristiche statistiche della probabilità esistenziale degli itemset (tecnica di *potatura decrementale*). È stato tuttavia dimostrato che queste tecniche sono veramente utili solo nel caso in cui abbiamo un notevole numero di oggetti con una *probabilità esistenziale bassa*. Quando non ci si trova in questo caso è infatti maggiore lo svantaggio dovuto all'overhead della computazione per la potatura che il vantaggio derivante dall'eliminazione dei dati. È stato inoltre dimostrato che, in casi come questo, è più efficiente la versione direttamente estesa di Apriori, U-Apriori,

senza variazioni per la potatura di dati.

2.2.2 Metodi con set di enumerazione

Possiamo usare tecniche simili a quelle usate per Apriori per estendere i metodi basati su set di enumerazione costruendo i candidati tramite la generazione dell'albero del set di enumerazione. Sono stati recentemente sviluppati diversi algoritmi che utilizzano questa tecnica, tra cui *MaxMiner*, *DephtProject* e *TreeProjection*. Questi metodi usano tipicamente estensioni top-down dell'albero in congiunzione con validazioni e tecniche di riduzione, sfruttando la proprietà di chiusura verso il basso. Chiaramente, differenti algoritmi usano differenti tecniche per costruire l'albero, in modo da ottenere il risultato migliore in base alle strategie di riduzione adottate. Dal momento che vale la proprietà di chiusura verso il basso, si possono facilmente estendere gli algoritmi al caso di dati incerti. Le modifiche chiave da apportare agli algoritmi sono simili a quelle effettuate nel caso dell'algoritmo Apriori: è necessario infatti calcolare il supporto previsto per ognuno dei diversi insiemi di oggetti.

2.2.3 Algoritmi di mining basati su modelli di crescita

I due principali algoritmi che sfruttano questa tecnica sono H-mine e FP-growth: adottano l'approccio *divide et impera* per scovare gli itemset frequenti. La principale differenza rispetto ai metodi visti finora risiede nella struttura di rappresentazione dei dati: FP-growth usa un albero di prefissi, mentre H-mine usa un hyper-linked array. Nel caso incerto la differenza di queste due strutture avrà un impatto molto maggiore rispetto al caso deterministico.

L'algoritmo H-mine scansiona inizialmente il database al fine di trovare gli oggetti frequenti, rimuovendo al contempo quelli non frequenti. Gli oggetti rimasti vengono poi ordinati in base ad uno schema globale. Il database trasformato viene quindi salvato in un array, dove ogni riga corrisponde ad una transazione. H-mine costruisce inoltre una tabella degli header, per accedere più velocemente all'array. Sfruttando il fatto che tale struttura non è in forma compressa, l'estensione per dati incerti è relativamente facile. Basterà salvare, oltre ai due campi base *item_id* e *hyper-link*, la probabilità *p(i,T)* associata all'oggetto, che indicherà la probabilità che l'oggetto i appartenga alla transazione T.

L'algoritmo FP-growth adotta una struttura ad albero con prefissi. Siccome è una struttura compressa, nascono una serie di sfide nell'adattarla al caso di dati incerti:

- nella versione originale, ogni nodo ha un valore "count" che registra il numero di transazioni che contengono il prefisso composto dal percorso dalla radice al nodo attuale. Per i dati incerti, se ci limitiamo a salvare in un nodo la somma delle probabilità degli oggetti che appartengono al cammino prefisso, non siamo più in grado di determinare la probabilità della presenza di un oggetto in ogni transazione. Vi è quindi una irreversibile perdita di informazioni dovuta all'uso della struttura compatta, che ci porta a dover considerare altri modi di conservare la probabilità degli oggetti senza perdita di troppe informazioni;

- nella versione originale, la computazione del supporto del cammino prefisso è banale, in quanto si prende il supporto minimo tra i nodi del cammino. Tuttavia, nel caso di dati incerti, la questione diventa complicata, in quanto bisogna tener conto del supporto previsto e, salvando

i dati nella maniera prima esplicata, non sappiamo determinare l'esatta probabilità della presenza di un oggetto in una transazione;

- dal momento che non si può determinare il supporto previsto per ogni itemset frequente dell'albero, potremmo dover prima estrarre tutti gli itemset candidati e poi eliminare quelli non frequenti, processo che può essere notevolmente oneroso in termini di tempo nel caso di dati incerti.

Esistono due soluzioni estreme per adattare l'albero FP al caso di data mining su dati incerti (chiameremo quest'albero incerto UFP): la prima è di salvare in ogni nodo la somma delle probabilità degli oggetti che compongono il cammino dalla radice al nodo corrente. L'albero UFP costruito in questo modo è compatto come l'albero FP originale. Tuttavia non può essere usato per calcolare l'upper bound o il lower bound del supporto previsto, in quanto perde informazioni rispetto alla probabilità degli oggetti nelle diverse transazioni. L'altra soluzione è di divedere un nodo in m nodi, se l'oggetto in questo nodo ha m diversi valori di probabilità. In questo caso siamo capaci di calcolare il supporto previsto, ma d'altro canto l'albero UFP costruito in questa maniera occupa molta memoria.

È stato tuttavia studiato un compromesso, che memorizza in ogni nodo un sottoinsieme delle probabilità di un oggetto: queste probabilità sono selezionate tramite clustering. Tale metodo non consuma troppa memoria e consente di calcolare l'upper bound del supporto previsto per ogni insieme di oggetti. Calcoleremo successivamente un insieme di possibili itemset frequenti, basandoci su questo upper bound. Questo insieme di candidati ci fornisce un superset dell'insieme completo degli itemset frequenti reali. Tutti i restanti falsi positivi saranno poi rimossi scansionando in una passata finale il database in input.

2.2.4 Uno studio comparativo su casi impegnativi

Sono stati testati i vari algoritmi (U-Apriori, UH-mine, UFP-growth) nel caso di dati incerti, ed è emerso che, in caso di alte probabilità esistenziali, i risultati sono molto diversi da quelli ottenuti nel caso di basse probabilità esistenziali, dove UFP-growth risulta il migliore.

Sono stati usati 4 data set. I primi due, *Connect4* e *kosarak*, sono database reali scaricati dal repository FIMI[1]. *Connect4* è un database denso, con 67.000 transazioni ognuna contenente 43 elementi, mentre *kosarak* è un database sparso, contenente 990.980 transazioni. Gli altri due, T40I10D100K e T25I15D320k, sono stati creati usando il generatore di dati di IBM[2]: contengono rispettivamente 100.000 e 320.000 transazioni. I 4 database presentano scenari molto differenti, ed è questo il motivo per cui tutti sono stati scelti per effettuare i test, in modo da presentare risultati il più possibile completi. È inoltre da notare che sono tutti deterministici, per cui è stata introdotta incertezza, in modo casuale, al fine di poter testare uno scenario con dati incerti e probabilistici. Le misure sono state fatte in termini di uso di memoria e tempo di esecuzione. Il principale risultato che emerge dallo studio è che l'algoritmo UH-mine presenta buone prestazioni, sia in termini di tempo che di uso di memoria, in tutte le situazioni, per cui è sicuramente il migliore da usare in una vasta varietà di scenari. Dall'altro lato UFP-growth presenta scarse prestazioni, sia nell'uso di memoria che nel tempo d'esecuzione, su tutti i dataset.

Anche dal punto di vista della scalabilità la situazione rimane uguale, con UH-mine che si comporta sensibilmente meglio di tutti gli altri e UFP-growth che presenta le prestazioni peggiori.

[1] **URL:** *http://fimi.cs.helsinki.fi/data/*
[2] **URL:** *http://miles.cnuce.cnr.it/~palmieri/datam/DCI/datasets.php*

2.2.5 Generalizzazione al modello dei Mondi Possibili

Tutti i casi discussi finora considerano il caso in cui ogni oggetto ha una probabilità indipendente di appartenere o meno ad una transazione. Esiste tuttavia un modello molto più potente, quello dei *mondi possibili*, dove la presenza o assenza di una data tupla influenza la presenza o assenza di un'altra. Il lato negativo nell'utilizzo di questo modello è che, a differenza di quelli analizzati finora, possiamo determinare insiemi di oggetti frequenti formati da un solo elemento.

In questo modello, invece di transazioni, abbiamo serie di *x-tuple*. Ogni x-tupla consiste in un insieme di oggetti con probabilità mutuamente esclusive di esistenza. Abbiamo quindi la probabilità $p(t,T)$ che un oggetto t appartenga alla x-tupla T. Un mondo possibile W corrisponde ad una combinazione di possibili istanziazioni delle x-tuple; la sua probabilità è data dal prodotto delle probabilità delle x-tuple. Si noti come un oggetto può essere presente più volte nei dati, dal momento che può essere scelto da più x-tuple: per descrivere questa situazione si usa un parametro di molteplicità, che sarà utilizzato, insieme alla probabilità, per calcolare la frequenza prevista di un dato oggetto. Si può quindi definire un oggetto come "*heavy-hitter*" se la sua molteplicità è al minimo ϕ con probabilità minima τ attraverso tutti i mondi possibili. L'uso di questi due parametri permette di astrarci in maniera efficace dal contesto in cui ci troviamo, rendendo possibile esaminare scenari molto diversi.

2.3 Identificazione di anomalie su dati incerti

Con identificazione di anomalie si intende l'individuazione di deviazioni significative dal comportamento abituale. Un esempio di utilizzo di que-

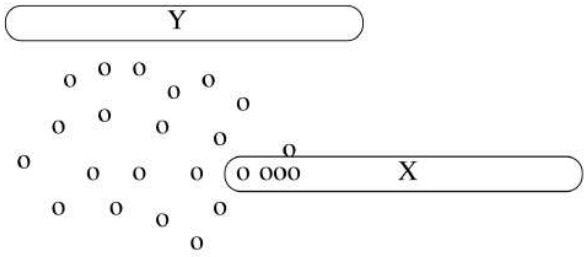

Figura 2.3: Effetto dell'incertezza nell'identificazione delle anomalie

ste tecniche è dato dall'identificazione di frodi con carte di credito: se per esempio una carta italiana, che è sempre stata usata in Italia, registra un movimento di alto importo da uno stato estero, il sistema segnalerà tale azione come un'anomalia, consentendo di attuare le predefinite misure di sicurezza.

Anche il problema di identificazione di anomalie è stato esteso al caso di dati incerti. In questa situazione, differenti livelli di incertezza attraverso diverse dimensioni possono pregiudicare la determinazione di anomalie sui dati sottostanti. Per esempio, si consideri il caso in figura 2.3, dove i limiti dell'incertezza per i due punti X e Y sono illustrati tramite ellissi che li circondano. Il punto X sembra più lontano da tutto il resto dei dati rispetto al punto Y. Tuttavia il contorno di incertezza fa sì che X abbia una maggiore probabilità di essere associato al resto dei dati. Si può quindi definire il concetto di anomalia in termini di probabilità che un certo punto sia associato ad una regione densa della distribuzione dati generale.

Al fine di quantificare la probabilità che un dato punto incerto sia associato ad una regione densa, si definisce il concetto di η-probabilità. L' η-probabilità di un punto X_i è definita come la probabilità che il punto incerto si trovi in una regione con densità minima η. La stima di questa η-probabilità può tuttavia essere impegnativa dal punto di vista computazionale, perciò viene fatta tramite l'uso di campioni. L'idea è di prelevare

diversi campioni dai dati e calcolare la frazione di campioni su cui la soglia di densità è specificata. Questo può essere usato per definire il concetto di anomalia-(δ,η).

Definizione 2.3.1. Un punto incerto X_i è definito come anomalia-(δ,η) se la η-probabilità di X_i in alcuni sottospazi è minore di δ.

Al fine di determinare le anomalie (δ,η) esiste un algoritmo che esplora tutti i sottospazi non vuoti dei dati con una ricerca in ampiezza bottom-up e determina tutti i punti per cui la condizione della definizione 2.3.1 è soddisfatta. Sono state anche discusse varie tecniche per velocizzare l'algoritmo, per esempio fare uso di micro-clustering in combinazione con stime di densità. Definendo i concetti di *precisione* e di *effettività*, rispettivamente come percentuale di anomalie rilevate che successivamente si sono dimostrate vere e come percentuale di anomalie realmente presenti trovate, è stato dimostrato che questa situazione di identificazione di anomalie su dati incerti è molto più efficace rispetto al classico algoritmo deterministico su dati certi [4].

2.4 Classificazione di dati incerti

Il processo di *classificazione*, che sarà oggetto di maggiore approfondimento in questa tesi, consiste nella costruzione (*apprendimento*) di un modello a partire da un insieme di record (*unità statistiche*) denotati da una classe, in modo che i record nuovi e non classificati possano essere assegnati alla classe che più loro corrisponde. Esempi di applicazioni reali possono essere i casi di classificazione di possibili evasori fiscali, terroristi, tumori benigni o maligni. Allo stato attuale viene gestita l'incertezza su tutti gli attributi a parte quello di classe, per cui non esistono algoritmi che prevedano la possibilità di avere incertezza anche su tale attributo. In realtà, quando si è chiamati a prendere

decisioni che possano stabilire se un individuo è un terrorista o meno oppure decisioni riguardo la natura di un tumore, l'incertezza sull'attributo di classe è sempre presente e determinante. Tramite l'estensione che presenterò sarà consentito attribuire un certo grado di confidenza all'affermazione che facciamo, per esempio permettendoci di dire che un tumore è, all'80%, benigno. Il fatto di riuscire a gestire l'incertezza ci consente di avere informazioni più dettagliate e precise e di compiere, di conseguenza, le scelte appropriate. Senza tenere conto dell'incertezza viene fatta un'approssimazione, per cui, nel caso sopra citato, il tumore viene sentenziato come benigno; in realtà non si è del tutto sicuri di tale affermazione e la possibilità di gestire l'incertezza apre a nuovi scenari e a scelte più oculate.

In questa sezione riservata ai classificatori analizzeremo l'estensione al caso di dati incerti sia per la classificazione bayesiana naif che per la classificazione tramite l'apprendimento di regole fuzzy. Per quel che riguarda la classificazione bayesiana naif l'idea chiave è di modificare la funzione di stima della probabilità del modello di Bayes per permetterle di gestire funzioni di distribuzione probabilistica. Per quanto invece concerne l'apprendimento di regole fuzzy verranno mostrati due algoritmi: l'idea base del primo è di usare relazioni di preferenza fuzzy, che consentano di gestire i concetti di *ignoranza* e *conflitto*, mentre l'idea alla base del secondo è di usare regole fuzzy anziché normali, e poi seguire i ragionamenti effettuati nel primo metodo.

2.4.1 Classificazione bayesiana naif

Un classificatore bayesiano è un classificatore probabilistico basato sull'applicazione diretta del teorema di Bayes [9], con una forte (naif) assunzione di indipendenza. Questo classificatore assume che la presenza (o assenza) di una particolare caratteristica di una classe non sia in relazione alla presenza

(o assenza) di ciascuna altra caratteristica. Per esempio, un frutto può essere considerato una mela se è rosso, rotondo e il suo diametro è di circa otto centimetri. Sebbene queste caratteristiche dipendano l'una dall'altra, e a volte anche dall'esistenza di altre caratteristiche, un classificatore bayesiano naif assume che tutte queste proprietà contribuiscano in maniera indipendente alla determinazione della probabilità che un certo frutto sia una mela.

L'estensione qui presentata può gestire dati incerti i cui valori sono rappresentati da una funzione di distribuzione probabilistica. L'idea chiave è di estendere la stima della probabilità nel modello di Bayes per far sì che possa trattare funzioni di distribuzione probabilistica.

I dati trattati avranno incertezza multi-dimensionale: in particolare, un oggetto non è un singolo punto nello spazio, ma è rappresentato da una regione incerta sulla quale è definita una funzione di distribuzione probabilistica. Formalmente, considereremo un insieme di n oggetti in uno spazio d-dimensionale. La posizione di ciascun oggetto è rappresentata da una funzione di distribuzione probabilistica p che specifica la densità probabilistica di ogni possibile locazione. Assumeremo inoltre che le funzioni di distribuzione probabilistica di ogni tupla siano indipendenti le une dalle altre.

Il problema chiave nella classificazione bayesiana naif è la stima della densità condizionale di classe. Tradizionalmente questo valore viene stimato basandosi su punti, ma nel caso incerto dovrà essere calcolato a partire da una distribuzione probabilistica. Per estendere quindi il metodo tradizionale sono stati studiati tre approcci:

- *Media (AVG)*: si calcola inizialmente il punto medio di ogni oggetto incerto. Successivamente questi punti sono passati al classificatore bayesiano naif.

- *Metodo basato su campioni (SBC)*: viene ridisegnata la funzione cen-

trale usata nel classificatore bayesiano naif per fare in modo che possa accettare come input dei campioni presi dai dati incerti. In questo metodo le distribuzioni probabilistiche possono essere arbitrarie.

- *Metodo basato sulla formula (FBC)*: è un'applicazione speciale del metodo sopra, dove viene derivata la formula della distribuzione Gaussiana per la funzione centrale.

Dai risultati ottenuti tramite gli esperimenti realizzati risulta che tutti e tre questi approcci portano a risultati più accurati del metodo originale, che non può gestire l'incertezza. AVG è il più semplice, ma non risulta accurato quanto SBC e FBC; quest'ultimo in particolare risulta essere sia più accurato che più efficiente degli altri due[10].

2.4.2 Apprendere strutture di preferenze fuzzy

L'idea di questo approccio è ridurre un problema che coinvolge m classi $\Lambda = \{\lambda_1 ... \lambda_m\}$ ad un insieme di problemi binari, ciascuno per ogni coppia ordinata di classi. Seguendo uno schema round robin, questo approccio chiamato LVPC (Learning Valued Preferences for Classification), dapprima istruisce un insieme di modelli binari, uno per ogni coppia di classi, e successivamente, data un'istanza $\mathbf{x}$ di valutazione con classe sconosciuta $\lambda(\mathbf{x})$, può derivare tre relazioni fuzzy (nella forma di mappature $\{1...m\} \times \{1...m\} \rightarrow [0, 1]$) dalle predizioni di questo insieme. Per ogni coppia di etichette (λ_i,λ_j), i valori corrispondenti in queste relazioni esprimono, rispettivamente:

- **preferenza:** grado per cui l'etichetta λ_i è strettamente preferita a λ_j come classificazione per $\mathbf{x}$ (e viceversa);

- **conflitto:** grado per cui λ_i e λ_j sono in conflitto l'un l'altra (nel caso in cui entrambe siano valide contemporaneamente come potenziale classificazione);

- **ignoranza:** grado di ignoranza che riflette come né λ_i né λ_j siano valide come classificazione.

In questo modo il problema della classificazione viene ridotto ad problema decisionale basato su relazioni di preferenze fuzzy. Una classificazione finale, o qualsiasi altro tipo di decisione (e.g., astenersi o decidere di raccogliere ulteriori informazioni), possono essere prese sulla base di queste tre relazioni. Una caratteristica chiave di questo approccio è la sua abilità di rappresentare l'ignoranza in maniera fedele. Infatti, sebbene molti metodi automatici di apprendimento siano in grado di rappresentare il conflitto in un modo o nell'altro, lo stesso non vale per l'ignoranza. Per meglio capire i concetti di conflitto e ignoranza nel contesto della classificazione possiamo considerare l'esempio in figura 2.4: date le osservazioni su due classi, `black` e `white`, tre nuove istanze (rappresentate da croci) devono essere classificate. Chiaramente l'istanza in alto a sinistra può essere classificata come `white`. Il caso dell'istanza in basso a sinistra, invece, presenta un alto livello di conflitto, in quanto entrambe le classi sembrano plausibili. La terza situazione è un esempio di ignoranza: l'istanza in alto a destra è situata in una regione dove non sono state fatte osservazioni. Di conseguenza non c'è nessuna prova a favore di una delle due classi. Tramite definizioni formali di *ignoranza* e *conflitto* si riescono a gestire anche quelle istanze che normalmente sarebbero fonte di problemi. Per i dettagli si rimanda a [18].

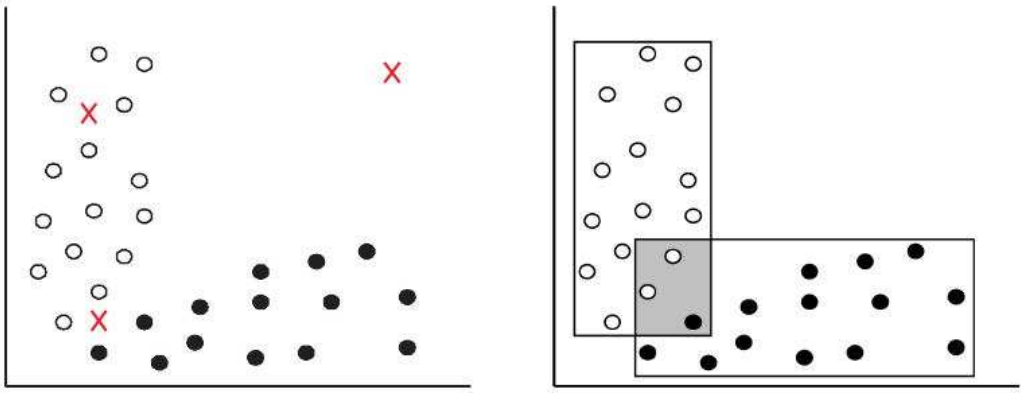

Figura 2.4: Scenari di classificazione d'esempio

2.4.3 FR3: apprendere regole fuzzy

L'idea alla base di questo metodo è di usare regole fuzzy al posto di regole convenzionali. Nello specifico è stata sviluppata un'estensione di RIPPER [20], un algoritmo di induzione su regole che produce modelli accurati in maniera efficiente; tale estensione è stata chiamata FRIPPER. Usando la versione fuzzy di RIPPER invece che quella classica come base di apprendimento nello schema di decomposizione round robin, è stato esteso il metodo R3 [21]. Sono stati eseguiti esperimenti che hanno mostrato come FR3 sia capace di rappresentare *ignoranza* e *conflitto* in maniera fedele ed inoltre sia migliore di R3 in termini di accuratezza predittiva. Per ottenere regole fuzzy l'intuizione è di rendere fuzzy le regole finali che vengono estratte dall'algoritmo FRIPPER. Nel dettaglio, l'idea è di cercare la miglior estensione fuzzy di ogni regola, dove un'estensione fuzzy è intesa come una regola dalla stessa struttura ma con gli intervalli sostituiti da intervalli fuzzy. Le regole sono rese fuzzy in maniera greedy: per ogni antecedente della regola viene eseguita un'operazione di trasformazione fuzzy, quella che risulta migliore per l'indice di purezza definito in [19]. Data la possibilità degli intervalli di esse-

re aperti, all'occorrenza viene anche gestita la limitazione delle regole fuzzy. Essendo il numero totale di operazioni limitato dal numero di attributi, n, la complessità risulta essere $O(|\mathbb{D}|n^2)$.

Supponiamo che le regole fuzzy $r_1^0 ... r_k^0$ e $r_1^1 ... r_l^1$ siano state apprese, rispettivamente, per le classi λ_0 e λ_1. Per una nuova istanza da classificare $\mathbf{x}$, il supporto di queste classi è dato da:

$$s_0 = max_{i=1...k}(\mu_{r_i^0}(x) \cdot CF(r_i^0))$$
$$s_1 = max_{j=1...l}(\mu_{r_j^1}(x) \cdot CF(r_j^1))$$

dove

$$\mathrm{CF(r)} = \frac{\sum_{x \in \mathbb{D}_T^+ : \lambda(x) = r_C} \mu(x)}{\sum_{x \in \mathbb{D}_T} \mu(x)}$$

è una misura di confidenza di validità della regola. Da questi due gradi di supporto si possono derivare le seguenti regole, che costituiscono l'output dell'algoritmo FRIPPER (nel caso si abbiano due classi):

$$P(\lambda_0, \lambda_1) = s_0 - min\{s_0, s_1\}$$
$$P(\lambda_1, \lambda_0) = s_1 - min\{s_0, s_1\}$$
$$C(\lambda_0, \lambda_1) = min\{s_0, s_1\}$$
$$I(\lambda_0, \lambda_1) = 1 - max\{s_0, s_1\}$$

$C(\lambda_0, \lambda_1)$ è il grado di *conflitto*, formalmente il grado per cui entrambe le classi sono supportate. Similarmente, $I(\lambda_0, \lambda_1)$ è il grado di *ignoranza*, cioè il grado per cui nessuna delle due classi è supportata. Infine $P(\lambda_0, \lambda_1)$ e $P(\lambda_1, \lambda_0)$ denotano rispettivamente la preferenza per λ_0 e λ_1. Si noti che almeno uno di questi due valori deve essere zero e che $P(\lambda_0, \lambda_1) + P(\lambda_1, \lambda_0) + C(\lambda_0, \lambda_1) + I(\lambda_0, \lambda_1) \equiv 1$.

Analisi sperimentali hanno dimostrato come FR3 produca regole più specifiche di R3 e come anche la lunghezza delle regole sia mediamente maggiore. Questo porta a modelli più complessi di quelli prodotti da R3, che tuttavia consentono di avere un miglioramento di performance in termini di accuratezza. Per i dettagli si rimanda a [19].

2.5 Alberi Decisionali

Per concludere questa parte introduttiva analizziamo l'estensione al caso di dati incerti per la classificazione tramite alberi decisionali. Verranno prima introdotte le nozioni base per capire al meglio il funzionamento degli alberi decisionali e successivamente mostrati due differenti algoritmi che si occupano del problema della gestione di dati incerti. L'idea alla base di entrambi è quella di modificare la funzione per la scelta del miglior attributo di split, in maniera tale che riesca a gestire dati probabilistici. La costruzione dell'albero sarà poi riadattata al fatto che ogni nodo presenterà valori di probabilità.

Come accennato in precedenza questa sezione sarà trattata in modo più esaustivo delle altre, in quanto l'algoritmo per l'estensione alla gestione dell'incertezza sull'attributo di classe sviluppato nel lavoro qui presentato si basa su questa tipologia di classificatori.

2.5.1 Nozioni di base

Analizzeremo adesso i concetti di base sugli alberi decisionali per la classificazione di dati tabulari, limitandoci alle nozioni che serviranno per questo lavoro di tesi.

Consideriamo la Figura 2.5, dove abbiamo rappresentato la tabella usata come esempio, presa da [17]. Questo dataset contiene informazioni riguardo

TID	Home Owner	Marital Status	Income	*Defaulted*
1	yes	Single	125K	no
2	no	Married	100K	no
3	no	Single	70K	no
4	yes	Married	120K	no
5	no	Divorced	95K	yes
6	no	Married	60K	no
7	yes	Divorced	220K	no
8	no	Single	85K	yes
9	no	Married	75K	no
10	no	Single	90K	yes

Figura 2.5: Tabella d'esempio T1

lo stato civile, il possesso o meno di una casa e il reddito, dati utili a stabilire se gli individui siano o meno debitori insolventi, dividendoli in due oppurtune classi. L'ultima colonna indica appunto l'attributo di *classe*, che prevede due possibili valori: `no` e `yes`. Per esempio, il record 5 ha classe `yes`. Quando non c'è incertezza questo dataset può essere usato per costruire un classificatore (albero decisionale) che potrà poi prendere altri record in input e assegnarli ad una delle due classi predefinite[3].

Un albero decisionale per la classificazione ha due tipi di nodi, come illustrato in Figura 2.6. I nodi interni (a sinistra nella figura) corrispondono ad un attributo della tabella di input usato per partizionare i record associati a quel nodo in accordo col loro valore di quell'attributo. Nell'esempio, tutti i record che hanno valore `yes` nell'attributo `Home Owner` apparterranno al sotto-albero di sinistra. Sulla destra della figura è invece illustrato un nodo foglia, che indica la classe associata a tale foglia e la percentuale di record che appartengono a ciascuna delle classi. In aggiunta indicheremo il numero

[3]Generalmente una parte dei dati viene usata per costruire il classificatore e una parte per testare la sua accuratezza, ma questo non è rilevante nella nostra discussione.

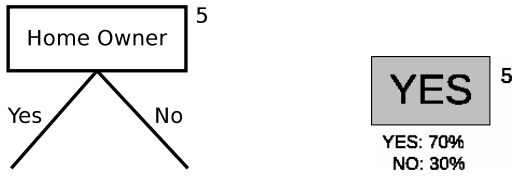

Figura 2.6: Nodi di un albero decisionale: interno e foglia

di record che appartengono ad ogni nodo: nell'esempio, 5.

L'algoritmo base usato per costruire un albero decisionale è molto semplice ed efficiente, ma potrebbe non fornire la soluzione migliore (che è invece un problema NP):

1. Parti con un singolo nodo rappresentante tutti i record nel dataset.
2. Scegli un attributo e dividi i record in base al loro valore di questo attributo.
3. Ripeti la divisione in tutti i nuovi nodi, finché un criterio di stop non viene soddisfatto.

Questa procedura è conosciuta come algoritmo di Hunt e chiaramente ci sono molti modi di implementarla. Definiremo adesso le opzioni adottate in questo lavoro, indicheremo i punti non tenuti in considerazione perché non direttamente interessati alla gestione dell'incertezza e forniremo un esempio di costruzione di un albero a cui dopo aggiungeremo incertezza. I tre aspetti che devono essere decisi per implementare l'algoritmo sono:

- Come partizionare i record successivamente alla scelta dell'attributo di split.
- Come scegliere l'attributo di split.
- Quando fermarsi.

Partizionamento

In questo esempio abbiamo due attributi categorici (`Home Owner` e `Marital Status`, rispettivamente binario e ternario), e un attributo continuo. In questo lavoro assumiamo che ogni attributo possa essere diviso in un numero di figli predefinito, perciò, per l'attributo continuo, useremo una procedura di discretizzazione per dividere i record in un numero finito di classi e nell'esempio considereremo due split con valori ≤ 80 e > 80. Non verranno fornite ulteriori indicazioni sulla scelta di questa soglia, poiché non rilevante alla discussione. Tuttavia, negli esempi seguenti, assumeremo che questa soglia sia fissata per un dato attributo.

Scelta dell'attributo di split

Una volte deciso come dividere i record, dobbiamo scegliere l'attributo di split: tale decisione è un problema chiave in un algoritmo induttivo per la costruzione di un albero decisionale. Ogni passo del processo costruttivo dell'albero necessita l'individuazione di un attributo per dividere l'insieme di dati in sottoinsiemi più piccoli: per fare ciò useremo una funzione molto comune (tra tutte le possibili) per misurare la bontà della scelta. Intuitivamente, se tutti i record appartengono ad una sola classe il nodo che risulterà avrà un alto valore di confidenza, cosa che invece non succede quanto i record sono equamente distribuiti tra tutte le classi.

Sia $p(c_i|n)$ la percentuale di record che appartengono alla classe c_i nel nodo n. L'entropia al nodo n è definita come:

$$E(n) = \sum_{\text{tutte le classi}} -p(c_i|n) \log_2 p(c_i|n) \tag{2.1}$$

La qualità complessiva di un partizionamento può essere misurata tramite

una somma pesata dell'entropia su tutti i nodi appena creati, definendo il suo livello di *impurità* [17]. Sia N il numero totale di record associati al genitore e sia N_i il numero di record associati all'i^{esimo} figlio dopo il partizionamento. L'impurità di un nodo di split S con figli $n_0, \ldots, n_k$ è definita come:

$$I(S) = \sum_{i \in [0,k]} \frac{N_i}{N} E(n_i) \tag{2.2}$$

Come esempio si può considerare la tabella T1, dove possiamo scegliere come attributo di split `Possiede Casa`, `Stato Maritale` o `Reddito`. Queste tre alternative sono illustrate in Figura 2.7 e le loro impurità sono, da sinistra a destra:

$$\begin{aligned}
\frac{3}{10}(-0\log_2 0 - 1\log_2 1) + \frac{7}{10}(-.43\log_2 .43 - .57\log_2 .57) &= .69 \\
\frac{4}{10}(-.5\log_2 .5 - .5\log_2 .5) + \frac{4}{10}(-0\log_2 0 - 1\log_2 1) &+ \\
+\frac{2}{10}(-.5\log_2 .5 - .5\log_2 .5) &= .6 \\
\frac{3}{10}(-0\log_2 0 - 1\log_2 1) + \frac{7}{10}(-.43\log_2 .43 - .57\log_2 .57) &= .69
\end{aligned}$$

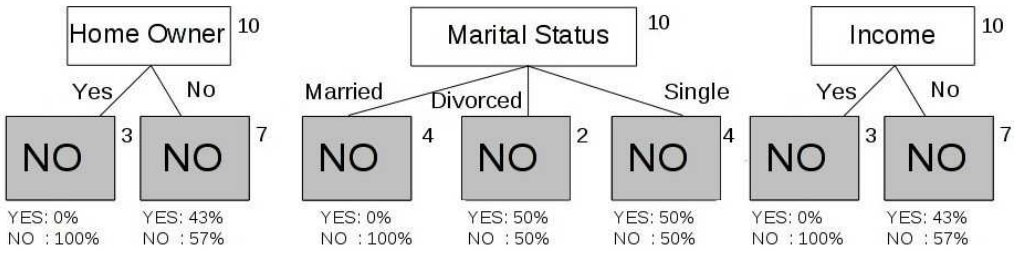

Figura 2.7: Attributi di split alternativi per il dataset T1

Da questi valori appare chiaro che lo split meno impuro sia quello effettuato sull'attributo `Stato Maritale`. A questo punto possiamo ricorsivamente

dividere i tre nuovi nodi finché non soddisferemo la condizione di arresto.

Condizione di arresto

Fermare la crescita dell'albero è molto importante per ridurne la dimensione ed incrementarne la leggibilità, argomenti che non tratteremo qui. Negli esempi successivi useremo le seguenti condizioni di arresto:
Un nodo viene processato se tutti i seguenti predicati sono verificati:

1. Esiste un nuovo attributo di split.
2. Il nodo contiene più di due record.
3. Non ci sono classi che hanno almeno il 75% dei record.

Notare che la condizione dei due record e la soglia del 75% sono arbitrarie e usate solamente come esempio.

Per concludere il nostro esempio, in Figura 2.8 abbiamo rappresentato i due successivi passi del processo di costruzione dell'albero, omettendo il calcolo dell'impurità.

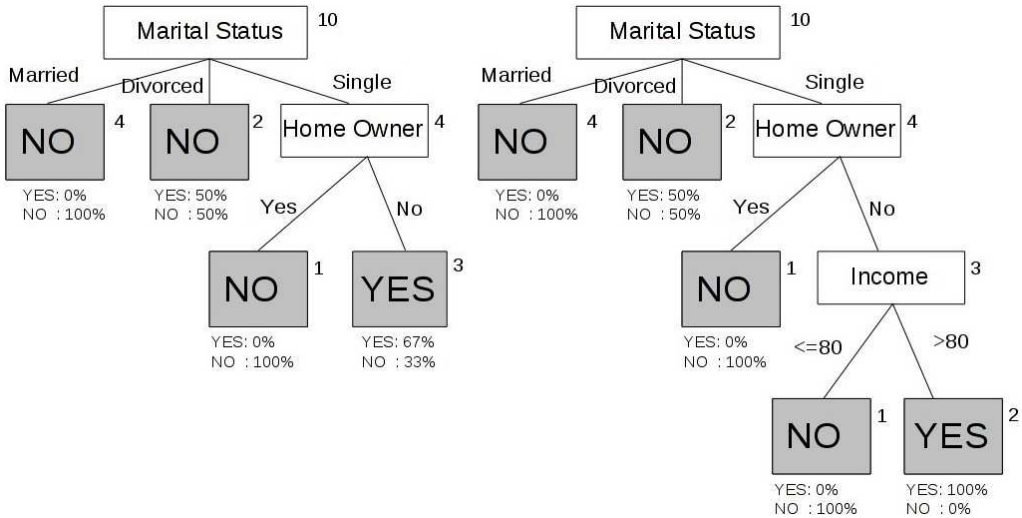

Figura 2.8: Costruzione dell'albero decisionale per il dataset T1

2.5.2 Alberi UDT

L'algoritmo alla base di questo metodo di classificazione si chiama appunto *UDT*, che sta per Uncertain Decision Tree. Gestisce i test di split solamente su attributi numerici, generando alberi binari: in questo modo ogni nodo avrà due figli, che saranno etichettati come "destro" e "sinistro". I valori degli attributi non sono valori puntuali, ma sono dati da una funzione di densità probabilistica. Generalmente verranno salvati una serie di s punti esempio con il relativo valore della funzione, in modo da approssimare la funzione ad una distribuzione discreta con s possibili valori. Per generare un buon albero di decisione sono fondamentali le scelte di A (attributo di split) e z (punto di split), che nel caso degli alberi UDT saranno effettuate da un algoritmo chiamato BestSplit, che si occuperà di garantire il minor grado di dispersione, misurato tramite entropia. Effettuata la scelta il set di tuple verrà diviso in due sottoinsiemi, D e S: per decidere come assegnare una tupla ad uno dei sottoinsiemi si verificheranno i limiti della sua funzione di densità, decidendo in base a quelli se una certa tupla sarà a destra, sinistra o se dovrà essere divisa in *tuple frazionarie* t_D e t_S. Sono stati eseguiti diversi test per misurare l'efficienza ed è emerso che tale algoritmo presenta sempre percentuali di accuratezza significativamente superiori all'algoritmo di classificazione che, data la funzione di densità probabilistica, associa alla tupla semplicemente la media della funzione. Tale algoritmo risulta tuttavia molto costoso dal punto di vista computazionale, sia per la scelta del miglior punto di split z, sia per la divisione delle tuple: sono perciò state sviluppate tecniche di potatura dei dati, che sono in grado di ridurre il numero di candidati per il punto di split e quindi rendono più veloce il calcolo dell'entropia. È stato dimostrato che si tratta di una potatura *safe*, che elimina solo candidati che peggiorerebbero l'attuale situazione, quindi alla fine del calcolo il

punto di split trovato risulta comunque il migliore possibile. Test eseguiti su basi di dati incerte hanno verificato l'effettiva efficienza di queste tecniche di potatura, che riducono notevolmente i tempi di esecuzione [13].

2.5.3 Alberi DTU

Questo secondo metodo di classificazione tramite alberi decisionali si basa sull'algoritmo DTU, che sta per Decision Tree for Uncertain Data [14]. È in grado di gestire sia dati incerti che dati certi e può trattare attributi sia numerici che categorici: l'unico vincolo imposto da questa procedura è che l'attributo di classe deve essere certo. Questo algoritmo verrà analizzato in maniera più estesa, in quanto sarà la base del mio lavoro di tesi.

I motivi che hanno portato a questa scelta sono stati soprattutto la possibilità di gestire sia dati incerti che dati certi e di trattare con attributi sia categorici che numerici. Inoltre, come sarà approfondito successivamente, questo algoritmo si basa sul diffuso metodo di apprendimento di alberi decisionali C4.5 [15], che garantisce una solida base e un'ampia garanzia in termini di risultati.

La scelta di estendere gli alberi decisionali è giustificata dalla loro semplicità ed efficacia: producono modelli facilmente interpretabili (che possono essere tradotti in regole di classificazione), non necessitano di assunzioni a priori sulla distribuzione dei dati e si costruiscono (usando euristiche) e classificano molto velocemente. Inoltre la loro accuratezza è comparabile a quella degli altri classificatori, eccezion fatta per quelli specifici. Tutto ciò non significa che gli alberi decisionali siano il miglior strumento di classificazione, ma tra tutte le differenti tecniche esistenti sono sicuramente una valida e forte opzione ed è quindi importante essere in grado di utilizzarli anche su dati incerti.

Incertezza negli attributi

Per poter gestire dati incerti vengono definite due nuove tipologie di dato, una per attributi numerici e una per categorici.

Quando il valore di un attributo numerico è incerto, l'attributo sarà chiamato *attributo numerico incerto* (UNA) e verrà indicato con $A_i^{u_n}$, mentre con $A_{ij}^{u_n}$ si indicherà la $j-$esima istanza di $A_i^{u_n}$ [4].

Un dataset può inoltre contenere attributi categorici, che possono essere anch'essi incerti; saranno chiamati *attributi categorici incerti* (UCA) e saranno indicati con $A_i^{u_c}$, mentre con $A_{ij}^{u_c}$ si indicherà la $j-$esima istanza di $A_i^{u_c}$[5].

Gli attributi incerti $A_{ij}^{u_n}$ e $A_{ij}^{u_c}$ prenderanno valore dai rispettivi domini Dom, con cardinalità $|Dom| = n$.

Mentre nel caso certo il valore dell'attributo A è il singolo valore d_k in Dom, con $Pr(A = d_k) = 1$, nel caso incerto salviamo le informazioni come una distribuzione di probabilità su Dom. Dato un dominio $Dom = \{d_1, ..., d_n\}$, gli attributi incerti A^{u_n} (UNA) e A^{u_c} (UCA) sono caratterizzati da una distribuzione probabilistica su Dom. Tale distribuzione può essere rappresentata tramite il vettore di probabilità $P = \{p_1, ..., p_n\}$ tale che $P(A_{ij}^{u_c} = v_k) = p_{jk}$ e $\sum_{k=1}^{n} p_{jk} = 1 (1 \leq j \leq n)$. Nel caso di attributi UNA avrà un ruolo determinante anche la scelta del punto di split, in quanto se tutti i valori in Dom risulteranno maggiori (o minori) di tale punto allora quella tupla dovrà essere trattata come una tupla certa. Se altrimenti alcuni valori in Dom risulteranno maggiori e altri minori allora avremo una tupla incerta da gestire come tale.

[4] la nozione di UNA è stata definita in [11].
[5] la nozione di UCA è stata definita in [12].

Identificare lo split migliore

Le misure più usate, come l'entropia (vista sopra) o l'indice Gini, non sono applicabili a dati incerti. Verranno perciò definite nuove misure, per garantire la scelta del miglior attributo di split.

TID	Home Owner	Marital Status	Income	*Defaulted*
1	(yes: 1.0)	(Single: 1.0)	(110K: 0.7; 120K: 0.3)	no
2	(no: 1.0)	(Single: 0.3; Married: 0.7)	(65K: 0.5; 80K: 0.5)	no
3	(yes: 0.9; no: 0.1)	(Divorced: 1.0)	(85K: 1.0)	yes
4	(yes: 0.5; no: 0.5)	(Married: 0.7; Divorced: 0.3)	(110K: 0.4; 145K: 0.6)	no
5	(no: 1.0)	(Single: 1.0)	(120K: 1.0)	yes
6	(yes: 0.3; no: 0.7)	(Married: 1.0)	(50K: 0.2; 80K: 0.8)	no
7	(yes: 1.0)	(Single: 0.8; Divorced: 0.2)	(170K: 0.7; 250K: 0.3)	no

Figura 2.9: Tabella d'esempio con UNA e UCA

Come descritto precedentemente, gli attributi incerti sono caratterizzati da una distribuzione di probabilità su Dom. Nella tabella riportata in figura 2.9 possiamo vedere un esempio di memorizzazione di attributi incerti UNA e UCA. Come si può notare abbiamo attributi incerti su tutte le colonne a parte quella dell'attributo di classe. Basandoci sulla probabilità di ogni istanza il cui attributo corrisponde a d_k possiamo definire la *cardinalità probabilistica* di d_k come la somma delle probabilità di ciascuna istanza il cui corrispondente attributo (UNA o UCA) sia uguale a d_k, cioè $PC(d_k) = \sum_{j=1}^{n} P(A_{ij}^{u_c} = d_k)$. La *cardinalità probabilistica* della classe C su d_k è la somma delle probabilità di ciascuna istanza in C_j il cui corrispondente attributo sia uguale a d_k, cioè $PC(d_k, C) = \sum_{j=1}^{n} P(A_{ij}^{u_c} = d_k \wedge C_j = C)$. Possiamo ora estendere tali definizioni all'intero dataset, affermando che la *cardinalità probabilistica* $PC(D)$ del dataset D è data dalla somma di tutte le probabilità delle tuple presenti, quindi è esattamente uguale al numero di record totali del dataset. La *cardinalità probabilistica* $PC(D, C_i)$ di ogni classe C_i è invece data dal numero di record appartenenti a tale classe.

Facendo riferimento al dataset riportato la cardinalità probabilistica per "Marital Status = Single" è la somma delle probabilità di ciascuna istanza il cui attributo "Marital Status" sia "Single", che è 3.1. La cardinalità probabilistica per la classe *sì* su "Marital Status = Single" è la probabilità totale delle istanze in classe *sì* il cui attributo "Marital Status" sia "Single", che è 1.0.

Con le definizioni precedenti possiamo ora definire l'entropia probabilistica per dati incerti come segue:

Definizione 2.5.1. *L'* Entropia Probabilistica *per un dataset D è:*

$$\text{ProbInfo(D)} = -\sum_{i=1}^{m} \frac{PC(D,C_i)}{PC(D)} \times \log_2(\frac{PC(D,C_i)}{PC(D)}).$$

Supponiamo che l'attributo A sia scelto come attributo di split e partizioni il dataset D in k sottoinsiemi, $\{D_1, D_2, ..., D_k\}$. Allora l'entropia probabilistica, o l'informazione attesa basandosi sul partizionamento, è data da $\text{ProbInfo}_A\text{(D)} = \sum_{j=1}^{k} \frac{PC(D_j)}{PC(D)} \times ProbInfo(D_j)$. Il termine $PC(D_j)$ agisce come peso per la $j-$esima partizione. Minore è il valore dell'entropia e maggiore sarà la purezza delle partizioni. Il guadagno in termini di divisione delle informazioni che si ottiene effettuando lo split su A è definito come ProbGain(A) = ProbInfo(D) - ProbInfo_A(D).

L'entropia probabilistica tende però a favorire gli attributi che hanno un grande numero di valori distinti. Il gain_ratio è massimo quando si ha un caso in ciascun sottoinsieme D_j. Per risolvere questo problema il criterio di split deve essere modificato in maniera che tenga conto del numero dei risultati prodotti dalla condizione di test per l'attributo. Questo criterio è definito come ProbGain_ratio(A) = $\frac{ProbGain(A)}{ProbSplitInfo_A(D)}$. In particolare $\text{ProbSplitInfo}_A\text{(D)} = -\sum_{j=1}^{k} \frac{PC(D_j)}{PC(D)} \times \log_2(\frac{PC(D_j)}{PC(D)})$ e k è il numero totale di split. Se un attributo produce un grande numero di split allora anche il suo valore di ProbSplitInfo_A(D) sarà elevato e questo riducerà il gain_ratio.

Algoritmo 1 Induzione su DTU

Input: il dataset di training D; il set di attributi candidati $att-list$.
Output: un albero di decisione per dati incerti.

```
1: crea un nodo N;
2: if (D sono tutti della stessa classe, C) then
3:     return N come un nodo foglia etichettato con la classe C;
4: else if (att-list è vuota) then
5:     return N come un nodo foglia etichettato con la classe che ha
       maggior peso in D;
6: end if;
7: seleziona l'attributo di test con il massimo gain_ratio probabilistico per
   etichettare il nodo N;
8: if (attributo di test è numerico o numerico incerto) then
9:     dividi i dati in maniera binaria dal prescelto punto di split y;
10:    for (ogni istanza R_j) do
11:        if (attributo di test è incerto) then
12:            if (attributo di test ≤ y) then
13:                inseriscilo in D_s con peso R_j.a_i.p × R_j.w;
14:            else
15:                inseriscilo in D_d con peso R_j.a_i.p × R_j.w;
16:            end if;
17:        else
18:            if (attributo di test ≤ y) then
19:                inseriscilo in D_s con peso R_j.w;
20:            else
21:                inseriscilo in D_d con peso R_j.w;
22:            end if;
23:        end if;
24:    end for;
25: else
26:    for (ogni valore a_i(i = 1, ..., n) dell'attributo do
27:        crea un ramo D_i per esso;
28:    end for;
29:    for (ogni istanza R_j) do
30:        if (attributo di test è incerto) then
31:            inseriscilo in D_i con peso R_j.a_i.p × R_j.w;
32:        else
33:            inseriscilo in D_i con peso R_j.w;
34:        end if
35:    end for;
36: end if;
37: for ogni D_i do
38:    attacca il nodo ritornato da DTU(D_i,att-list);
39: end for;
```

Algoritmo di induzione per DTU

Lo pseudocodice per l'algoritmo è mostrato nell'algoritmo 1. La strategia base è la seguente:

1. L'albero parte come un singolo nodo rappresentante il campione di training (riga 1).

2. Se i campioni sono tutti della stessa classe allora il nodo diventa una foglia e viene etichettato con quella classe (righe 2 e 3).

3. Altrimenti l'algoritmo usa la misura del massimo gain_ratio probabilistico, basata sull'entropia probabilistica, come criterio per selezionare l'attributo di split. Questo attributo diventa l'attributo di "test" per quel nodo (riga 7).

4. Se l'attributo di test è numerico o numerico incerto dividiamo i dati al punto di split prescelto y (righe 8 e 9).

5. Vengono creati due sottorami, rispettivamente per attributo di test $\leq y$ e $> y$. Se l'attributo è certo e il suo valore è minore o uguale ad y allora verrà inserito nel ramo sinistro con peso $R_j.w$; se invece il suo valore è maggiore di y verrà inserito nel ramo destro con peso $R_j.w$. Se l'attributo è invece incerto denotiamo con $R_j.a_i.p$ la probabilità del valore a_i dell'attributo: se il suo valore è minore o uguale ad y allora sarà inserito nel ramo sinistro con peso $R_j.a_i.p \times R_j.w$, se è maggiore di y sarà inserito nel ramo destro con peso $R_j.a_i.p \times R_j.w$ (righe $10-24$).

6. Se l'attributo di test è categorico o categorico incerto dividiamo i dati in più sottorami (righe $26-35$). Un sottoramo viene creato per ogni valore possibile dell'attributo di test e i dati campione sono partizionati

in accordo con i sottorami. Per ogni valore a_i dell'attributo un'istanza viene messa in D_i con peso $R_j.w$ quando l'attributo è certo. Se l'attributo è invece incerto, denotando sempre con $R_j.a_i.p$ la probabilità del valore a_i dell'attributo, un'istanza verrà inserita nel sottoramo a_i con peso $R_j.a_i.p \times R_j.w$.

7. L'algoritmo si riapplica ricorsivamente per generare un albero di decisione tramite i dati forniti.

8. Il processo ricorsivo di partizionamento si ferma quando una delle due condizioni diventa vera:

   - Tutti i campioni di un dato nodo appartengono alla stessa classe (righe 2 e 3);
   - Non ci sono più attributi su cui partizionare (riga 4). In questo caso il nodo viene prima trasformato in foglia e successivamente etichettato con la classe di peso maggiore (riga 5).

Esempio

Viene riportata la tabella d'esempio 2.1, per meglio capire la scelta dell'attributo di split e la costruzione dell'albero di decisione.

Iniziamo dunque a calcolare i valori dei parametri globali e di quelli relativi ai diversi attributi per la tabella 2.1:

$\textbf{ProbInfo(D)} = -(\frac{4}{11}\log_2\frac{4}{11} + \frac{7}{11}\log_2\frac{7}{11}) = .95$

XID	TID	Conf	H. Owner	M. Status	Income	*Defaulted*
11	1	1.0	yes	Single	125K	no
12	2	1.0	no	Married	100K	no
13	3	1.0	no	Single	70K	no
14	4	1.0	yes	Married	120K	yes
15	5	1.0	no	Divorced	95K	yes
16	6	1.0	no	Married	60K	no
17	7	1.0	yes	Divorced	220K	no
18	8	1.0	no	Single	85K	yes
19	9	0.7	**no**	Married	75K	no
19	10	0.3	**yes**	Married	75K	no
110	11	1.0	no	Single	90K	yes
111	12	0.8	yes	**Single**	100K	no
111	13	0.2	yes	**Divorced**	100K	no

Tabella 2.1: Tabella DTU T1

HOME OWNER:

$PC(YES) = 4.30$

$PC(NO) = 6.70$

$PC(YES, YES) = 1.00$

$PC(YES, NO) = 3.30$

$PC(NO, YES) = 3.00$

$PC(NO, NO) = 3.70$

$ProbInfo(D_{YES}) = -(\frac{1}{4.3}\log_2\frac{1}{4.3} + \frac{3.3}{4.3}\log_2\frac{3.3}{4.3}) = .78$

$ProbInfo(D_{NO}) = -(\frac{3}{6.7}\log_2\frac{3}{6.7} + \frac{3.7}{6.7}\log_2\frac{3.7}{6.7}) = .99$

$ProbInfo_A(D) = \frac{4.3}{11} \times 0.78 + \frac{6.7}{11} \times 0.99 = .91$

$ProbGain(A) = 0.95 - 0.91 = .04$

$ProbSplitInfo_A(D) = -(\frac{4.3}{11}\log_2\frac{4.3}{11} + \frac{6.7}{11}\log_2\frac{6.7}{11}) = .97$

$ProbGain\_ratio(A) = \frac{0.04}{0.97} = .04$

MARITAL STATUS:

$PC(SINGLE) = 4.80$

$PC(MARRIED) = 4.00$

$PC(DIVORCED) = 2.20$

$PC(SINGLE, YES) = 2.00$

$PC(SINGLE, NO) = 2.80$

$PC(MARRIED, YES) = 1.00$

$PC(MARRIED, NO) = 3.00$

$PC(DIVORCED, YES) = 1.00$

$PC(DIVORCED, NO) = 1.20$

$ProbInfo(D_{SINGLE}) = -(\frac{2}{4.8}\log_2\frac{2}{4.8} + \frac{2.8}{4.8}\log_2\frac{2.8}{4.8}) = .98$

$ProbInfo(D_{MARRIED}) = -(\frac{1}{4}\log_2\frac{1}{4} + \frac{3}{4}\log_2\frac{3}{4}) = .81$

$ProbInfo(D_{DIVORCED}) = -(\frac{1}{2.2}\log_2\frac{1}{2.2} + \frac{1.2}{2.2}\log_2\frac{1.2}{2.2}) = .99$

$ProbInfo_A(D) = \frac{4.8}{11} \times 0.98 + \frac{4}{11} \times 0.81 + \frac{2.2}{11} \times 0.99 = .92$

$ProbGain(A) = 0.95 - 0.92 = .03$

$ProbSplitInfo_A(D) = -(\frac{4.8}{11}\log_2\frac{4.8}{11} + \frac{4}{11}\log_2\frac{4}{11} + \frac{2.2}{11}\log_2\frac{2.2}{11}) = 1.52$

$ProbGain\_ratio(A) = \frac{0.03}{1.52} = .02$

INCOME:

$PC(\leq 80) = 3.00$

$PC(> 80) = 8.00$

$PC(\leq 80, YES) = 0.00$

$PC(\leq 80, NO) = 3.00$

$PC(> 80, YES) = 4.00$

$PC(> 80, NO) = 4.00$

$ProbInfo(D_{\leq 80}) = -(\frac{0}{3}\log_2\frac{0}{3} + \frac{3}{3}\log_2\frac{3}{3}) = .00$

$ProbInfo(D_{>80}) = -(\frac{4}{8}\log_2\frac{4}{8} + \frac{4}{8}\log_2\frac{4}{8}) = 1.00$

$ProbInfo_A(D) = \frac{3}{11} \times 0.00 + \frac{8}{11} \times 1.00 = .73$

$ProbGain(A) = 0.95 - 0.73 = .22$

$ProbSplitInfo_A(D) = -(\frac{3}{11}\log_2\frac{3}{11} + \frac{8}{11}\log_2\frac{8}{11}) = .85$

$ProbGain\_ratio(A) = \frac{0.22}{0.85} = .26$

Home Owner	YES	NO
$PC(d_j)$	4.30	6.70
$PC(d_j, YES)$	1.00	3.00
$PC(d_j, NO)$	3.30	3.70
$ProbInfo(D_j)$	0.78	0.99
$ProbInfo_A(D)$	0.91	
$ProbGain(A)$	0.04	
$ProbSplitInfo_A(D)$	0.97	
$ProbGain\_ratio(A)$	0.04	

Figura 2.10: Tabella riepilogativa per l'attributo Home Owner

Income	≤ 80	> 80
$PC(d_j)$	3.00	8.00
$PC(d_j, YES)$	0.00	4.00
$PC(d_j, NO)$	3.00	4.00
$ProbInfo(D_j)$	0.00	1.00
$ProbInfo_A(D)$	0.73	
$ProbGain(A)$	0.22	
$ProbSplitInfo_A(D)$	0.85	
$ProbGain\_ratio(A)$	0.26	

Figura 2.11: Tabella riepilogativa per l'attributo Income

Marital Status	Single	Married	Divorced
$PC(d_j)$	4.80	4.00	2.20
$PC(d_j, YES)$	2.00	1.00	1.00
$PC(d_j, NO)$	2.80	3.00	1.20
$ProbInfo(D_j)$	0.98	0.81	0.99
$ProbInfo_A(D)$	0.92		
$ProbGain(A)$	0.03		
$ProbSplitInfo_A(D)$	1.52		
$ProbGain\_ratio(A)$	0.02		

Figura 2.12: Tabella riepilogativa per l'attributo Marital Status

Notiamo come l'attributo evidenziato (`Income`) sia quello che presenti maggior gain_ratio, quindi quello scelto come primo attributo di split. L'albero che verrà costruito è riportato in figura 2.13:

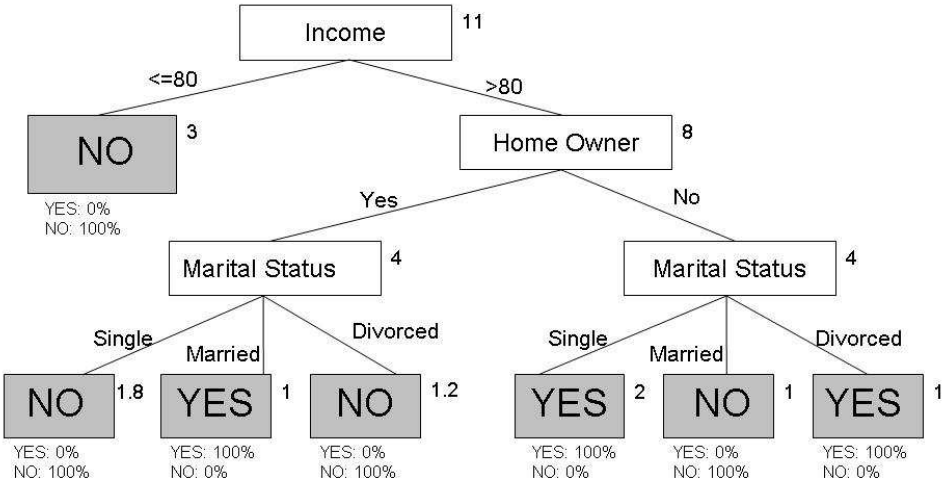

Figura 2.13: Albero per T1

Predizione con DTU

Una volta che si è costruito un albero DTU lo si può usare per predire la classe delle tuple. Il processo di predizione parte dal nodo radice, la condizione di test viene applicata a tutti i nodi del DTU e il sottoramo appropriato viene seguito in base ai risultati del test. Quando l'istanza di test R è certa il processo è abbastanza semplice, in quanto il risultato del test porterà ad un unico ramo senza ambiguità. Quando il test invece è su un attributo incerto l'algoritmo di predizione procede nel modo seguente:

1. se la condizione di test è su un attributo UNA (A) e il punto di split è a:

   - se tutti i valori in Dom sono $>$ di a allora R seguirà il ramo destro, con peso $R.w$;
   - se tutti i valori in Dom sono $\leq$ di a allora R seguirà il ramo sinistro, con peso $R.w$;

- se i valori in Dom si dividono tra maggiori e minori di a allora R potrà seguire sia il ramo destro che quello sinistro, con peso rispettivamente di $R.w * P(R.A > a)$ e $R.w * P(R.A \leq a)$.

2. se la condizione di test è su un attributo UCA (A) e $a_1, a_2, ..., a_k$ sono i possibili valori per l'attributo categorico A, allora supponiamo che $R.A$ sia un UCA, cioè $R.A = \{p_1, p_2, ..., p_k\}$, con $p_i (i = 1, .., k)$ come probabilità di $R.A = a_i$. Allora R sarà nell'$i-$esimo ramo con probabilità p_i.

Per quanto riguarda i nodi foglia del DTU, ogni classe C_i avrà probabilità $PL(C_i)$, cioè la probabilità che un'istanza appartenga alla classe C_i se finisce nel nodo foglia. $PL(C_i)$ viene calcolata come una frazione della cardinalità probabilistica delle istanze in classe C_i in un nodo foglia sulla cardinalità probabilistica totale delle istanze in quel nodo. Assumendo che il percorso L dalla radice al nodo foglia contenga t test e che i dati siano alla fine classificati in una classe c_i, supponiamo che $P(T_i)$ sia la probabilità che l'istanza segua il percorso all'$i-$esimo test: allora la probabilità di un'istanza di essere in classe c_i prendendo quel particolare percorso L è $P^L_{c_i} = PL(c_i) * \prod_{i=1}^{t} P(T_i)$.

Quando si predice la classe per un'istanza T con attributi incerti è possibile che il processo prenda percorsi multipli. Supponendo che vengano presi in totale m percorsi, allora la probabilità che T appartenta alla classe c_i è $P_{c_i} = \sum_{i=1}^{m} P^i_{c_i}$.

Concludendo, la predizione affermerà che l'istanza appartiene alla classe c_i che ha la maggior P_{c_i} tra tutte le $P_{c_i}, i = 1, ..., n$.

Capitolo 3

Alberi DTCU

In questo capitolo verranno analizzate le possibili situazioni da gestire avendo incertezza sull'attributo di classe e verrà mostrato lo pseudocodice della procedura che si occuperà di tale operazione.

Come precedentemente accennato l'algoritmo DTU ha come unico vincolo l'obbligo, per l'attributo di classe, di essere certo. Tuttavia non è sempre questa la situazione in cui ci si può imbattere: consideriamo per esempio il dataset visto sopra, in tabella 2.1. Tale base di dati è utile alla classificazione dei debitori insolventi, riconosciuti tramite i tre attributi in tabella. Effettuare una distinzione, assegnando quindi il valore *yes* o *no* all'attributo di classe è, come noto, il passo decisivo in un algoritmo di classificazione. Proprio questo passo potrebbe tuttavia essere l'operazione su cui abbiamo maggiore incertezza, non essendo sempre in grado di riconoscere in maniera totalmente sicura la classe di appartenenza di un record. Sono infatti molto più frequenti i casi in cui l'affermazione che siamo in grado di fare è "*Molto probabilmente questa persona è un debitore insolvente*", oppure *"Questa persona non dovrebbe essere un debitore insolvente"*, che non casi in cui siamo completamente certi che una persona sia o meno un debitore insolvente.

Altro caso d'esempio che possiamo prendere in analisi è quello delle diagnosi fatte da un medico sui tumori: in base a diversi parametri del paziente sarà chiamato a dire se il tumore è benigno o maligno, ma nella maggior parte dei casi sarà solamente in grado di propendere verso un'ipotesi piuttosto che verso un'altra, non avendo la certezza assoluta della propria diagnosi. Farebbe dunque comodo poter gestire casi in cui anche l'attributo di classe sia incerto o probabilistico, essendo questa una decisione spesso incerta. L'estensione parte da questo presupposto e, analizzando diversi esempi, verranno illustrati l'idea alla base dell'estensione e lo pseudocodice da integrare nell'algoritmo 1 del paragrafo precedente per riuscire a rimuovere quest'ultimo vincolo e rendere così l'algoritmo DTU completo.

3.1 Analisi delle situazioni possibili

Analizziamo quindi tramite la tabella 3.1 le possibili situazioni di incertezza di fronte a cui possiamo trovarci.

Si può osservare che la tupla con $XID = 14$ presenta incertezza solo sull'attributo di classe, la tupla con $XID = 16$ denota incertezza sia sull'attributo `Home Owner` che sull'attributo di classe, la tupla con $XID = 18$ presenta incertezza solamente sull'attributo `Home Owner`, come l'esempio nel capitolo precedente, e infine la tupla con $XID = 110$ possiede incertezza sia sull'attributo `Marital Status` che sull'attributo di classe, in questo caso combinando le diverse possibilità.

Per il calcolo dei diversi valori, spiegati nel paragrafo precedente, che occorrono per scegliere l'attributo di split bisognerà questa volta prestare attenzione al fatto che la classe non è più un attributo certo, ma andrà considerata in relazione al peso dell'istanza cui si riferisce.

XID	TID	Conf	H. Owner	M. Status	Income	*Defaulted*
11	1	1.0	yes	Single	125K	no
12	2	1.0	no	Married	100K	no
13	3	1.0	no	Single	70K	no
14	4	0.8	yes	Married	120K	**yes**
14	5	0.2	yes	Married	120K	**no**
15	6	1.0	no	Married	60K	no
16	7	0.6	**no**	Divorced	220K	**yes**
16	8	0.4	**yes**	Divorced	220K	**no**
17	9	1.0	no	Single	85K	yes
18	10	0.7	**no**	Married	75K	no
18	11	0.3	**yes**	Married	75K	no
19	12	1.0	no	Single	90K	yes
110	13	0.2	yes	**Single**	100K	**no**
110	14	0.4	yes	**Single**	100K	**yes**
110	15	0.3	yes	**Divorced**	100K	**no**
110	16	0.1	yes	**Divorced**	100K	**yes**

Tabella 3.1: Tabella DTCU T2 con incertezza su classe

Procediamo dunque con il calcolo dei parametri globali e di quelli relativi ai diversi attributi per la tabella 3.1:

ProbInfo(D) $= -(\frac{3.9}{10}\log_2\frac{3.9}{10} + \frac{6.1}{10}\log_2\frac{6.1}{10}) = .96$

HOME OWNER:

$PC(YES) = 3.70$

$PC(NO) = 6.30$

$PC(YES, YES) = 1.30$

$PC(YES, NO) = 2.40$

$PC(NO, YES) = 2.60$

$PC(NO, NO) = 3.70$

$ProbInfo(D_{YES}) = -(\frac{1.3}{3.7}\log_2\frac{1.3}{3.7} + \frac{2.4}{3.7}\log_2\frac{2.4}{3.7}) = .94$

$ProbInfo(D_{NO}) = -(\frac{2.6}{6.3}\log_2\frac{2.6}{6.3} + \frac{3.7}{6.3}\log_2\frac{3.7}{6.3}) = .98$

$ProbInfo_A(D) = \frac{3.7}{10} \times 0.94 + \frac{6.3}{10} \times 0.98 = .97$

$ProbGain(A) = 0.96 - 0.97 = -.01 \rightarrow .00$

$ProbSplitInfo_A(D) = -(\frac{3.7}{10}\log_2\frac{3.7}{10} + \frac{6.3}{10}\log_2\frac{6.3}{10}) = .95$

$ProbGain\_ratio(A) = \frac{.00}{0.95} = .00$

MARITAL STATUS:

$PC(SINGLE) = 4.60$

$PC(MARRIED) = 4.00$

$PC(DIVORCED) = 1.40$

$PC(SINGLE, YES) = 2.40$

$PC(SINGLE, NO) = 2.20$

$PC(MARRIED, YES) = 0.80$

$PC(MARRIED, NO) = 3.20$

$PC(DIVORCED, YES) = 0.70$

$PC(DIVORCED, NO) = 0.70$

$ProbInfo(D_{SINGLE}) = -(\frac{2.4}{4.6}\log_2\frac{2.4}{4.6} + \frac{2.2}{4.6}\log_2\frac{2.2}{4.6}) = 1.00$

$ProbInfo(D_{MARRIED}) = -(\frac{0.8}{4}\log_2\frac{0.8}{4} + \frac{3.2}{4}\log_2\frac{3.2}{4}) = .72$

$ProbInfo(D_{DIVORCED}) = -(\frac{0.7}{1.4}\log_2\frac{0.7}{1.4} + \frac{0.7}{1.4}\log_2\frac{0.7}{1.4}) = 1.00$

$ProbInfo_A(D) = \frac{4.6}{10} \times 1.00 + \frac{4}{10} \times 0.72 + \frac{1.4}{10} \times 1.00 = .89$

$ProbGain(A) = 0.96 - 0.89 = .07$

$ProbSplitInfo_A(D) = -(\frac{4.6}{10}\log_2\frac{4.6}{10} + \frac{4}{10}\log_2\frac{4}{10} + \frac{1.4}{10}\log_2\frac{1.4}{10}) = 1.44$

$ProbGain\_ratio(A) = \frac{0.07}{1.44} = .05$

INCOME:

$PC(\leq 80) = 3.00$

$PC(> 80) = 7.00$

$PC(\leq 80, YES) = 0.00$

$PC(\leq 80, NO) = 3.00$

$PC(> 80, YES) = 3.90$

$PC(> 80, NO) = 3.10$

$ProbInfo(D_{\leq 80}) = -(\frac{0}{3}\log_2\frac{0}{3} + \frac{3}{3}\log_2\frac{3}{3}) = .00$

$ProbInfo(D_{>80}) = -(\frac{3.9}{7}\log_2\frac{3.9}{7} + \frac{3.1}{7}\log_2\frac{3.1}{7}) = 0.99$

$ProbInfo_A(D) = \frac{3}{10} \times 0.00 + \frac{7}{10} \times 0.99 = .69$

$ProbGain(A) = 0.96 - 0.69 = .27$

$ProbSplitInfo_A(D) = -(\frac{3}{10}\log_2\frac{3}{10} + \frac{7}{10}\log_2\frac{7}{10}) = .88$

$ProbGain\_ratio(A) = \frac{0.27}{0.88} = .31$

Home Owner	YES	NO
$PC(d_j)$	3.70	6.30
$PC(d_j, YES)$	1.30	2.60
$PC(d_j, NO)$	2.40	3.70
$ProbInfo(D_j)$	0.94	0.98
$ProbInfo_A(D)$	0.97	
$ProbGain(A)$	0.00	
$ProbSplitInfo_A(D)$	0.95	
$ProbGain\_ratio(A)$	0.00	

Figura 3.1: Tabella riepilogativa per l'attributo Home Owner

Income	≤ 80	> 80
$PC(d_j)$	3.00	7.00
$PC(d_j, YES)$	0.00	3.90
$PC(d_j, NO)$	3.00	3.10
$ProbInfo(D_j)$	0.00	0.99
$ProbInfo_A(D)$	0.69	
$ProbGain(A)$	0.27	
$ProbSplitInfo_A(D)$	0.88	
$ProbGain\_ratio(A)$	0.31	

Figura 3.2: Tabella riepilogativa per l'attributo Income

Risulta evidente come l'attributo evidenziato (`Income`) sia quello che presenti maggior gain_ratio, quindi quello scelto come primo attributo di split. L'albero che verrà costruito è riportato in figura 3.3:

Marital Status	Single	Married	Divorced
$PC(d_j)$	4.60	4.00	1.40
$PC(d_j, YES)$	2.40	0.80	0.70
$PC(d_j, NO)$	2.20	3.20	0.70
$ProbInfo(D_j)$	1.00	0.72	1.00
$ProbInfo_A(D)$	0.89		
$ProbGain(A)$	0.07		
$ProbSplitInfo_A(D)$	1.44		
$ProbGain\_ratio(A)$	0.05		

Figura 3.2: Tabella riepilogativa per l'attributo Marital Status

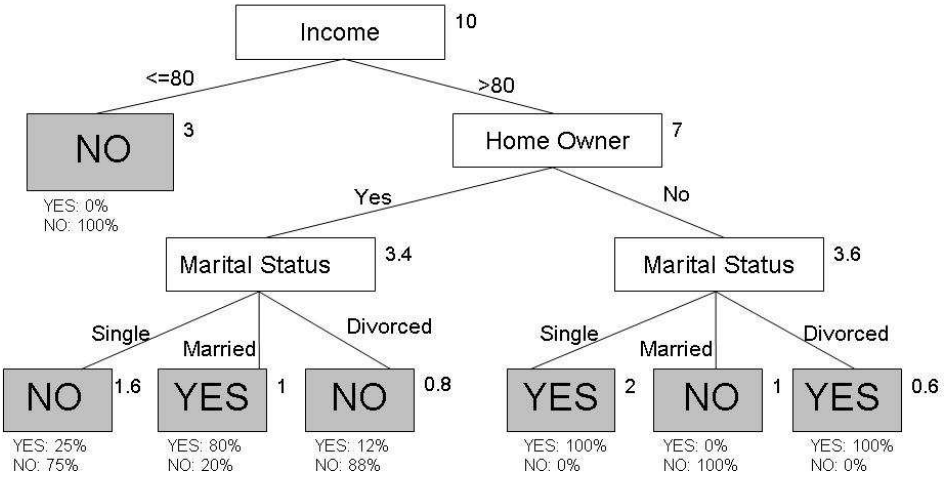

Figura 3.3: Albero per T2

3.2 Algoritmo per l'estensione

Vediamo infine lo pseudocodice della procedura. L'idea di base è che ogni dato verrà inserito nel ramo d'appartenenza (D_s o D_d nel caso di attributi numerici certi e incerti, D_i per attributi categorici certi e incerti[1]) seguendo l'algoritmo definito nel capitolo precedente, con la differenza che, ad ogni inserzione, si terrà conto della possibile incertezza sull'attributo di classe, assegnando un peso alla rispettiva classe: tale valore corrisponderà alla probabilità di tale attributo. Sarà quindi necessario controllare, per ogni situazione che si può verificare, l'incertezza sull'attributo di classe e agire in maniera tale da gestire opportunamente i valori che si hanno.

[1]Con D_s si intende il sottoramo sinistro, con D_d il sottoramo destro e con D_i l'i-esimo sottoramo.

Algoritmo 2 Estensione DTCU

```
1: if (attributo di test è numerico o numerico incerto) then
2:     if (attributo di test è incerto) then
3:         if (attributo di classe è incerto) then
4:             for (ogni classe C_i a cui può appartenere R_j do
5:                 somma al valore della classe C_i il valore R_j.a_i.p × R_j.w
6:             end for;
7:         else
8:             somma al valore della classe C_i a cui appartene R_j il valore
               R_j.a_i.p × R_j.w
9:         end if;
10:    else
11:        if (attributo di classe è incerto) then
12:            for (ogni classe C_i a cui può appartenere R_j do
13:                somma al valore della classe C_i il valore R_j.a_i.p × R_j.w
14:            end for;
15:        else
16:            somma al valore della classe C_i a cui appartene R_j il valore
               R_j.w
17:        end if;
18:    end if;
19: else
20:    if (attributo di test è incerto) then
21:        if (attributo di classe è incerto) then
22:            for (ogni classe C_i a cui può appartenere R_j do
23:                somma al valore della classe C_i il valore R_j.a_i.p × R_j.w
24:            end for;
25:        else
26:            somma al valore della classe C_i a cui appartene R_j il valore
               R_j.a_i.p × R_j.w
27:        end if;
28:    else
29:        if (attributo di classe è incerto) then
30:            for (ogni classe C_i a cui può appartenere R_j do
31:                somma al valore della classe C_i il valore R_j.a_i.p × R_j.w
32:            end for;
33:        else
34:            somma al valore della classe C_i a cui appartene R_j il valore
               R_j.w
35:        end if;
36:    end if;
37: end if;
```

Capitolo 4

Implementazione e analisi dei risultati

Una volta definito lo pseudocodice per la procedura si è quindi passati all'implementazione effettiva: la scelta è stata di estendere l'algoritmo C4.5 presente in Weka [16]. Sono successivamente stati eseguiti diversi test, prima per dimostrare la validità dell'estensione sviluppata e in un secondo momento per ottenere dei primi risultati sperimentali. Per tali analisi abbiamo scelto un dataset specifico, che consentisse di cogliere immediatamente i vantaggi ottenuti dell'estensione per la gestione dell'incertezza sull'attributo di classe.

4.1 Overview su Weka

Weka è un software open source scritto in Java, sviluppato dall'università neozelandese di Waikato e rilasciato sotto licenza GNU GPL [23]. Contiene una collezione di algoritmi utili a operazioni di data mining: classificazione, clustering, estrazione di regole, regressione e anche una libreria grafica che consente di visualizzare i risultati della propria elaborazione.

La sua architettura consente di ampliarlo con nuovi algoritmi che possano sfruttare le basi delle procedure già implementate. Si è deciso quindi di estendere l'algoritmo C4.5 presente in Weka, piuttosto che scrivere un nuovo tool da zero, in quanto ciò permette di avere sia una solida e affidabile base che una libreria grafica che consenta di osservare fin da subito e in maniera chiara i risultati ottenuti.

4.2 Scelta del dataset

Per effettuare i test e le analisi abbiamo scelto un dataset tra quelli presenti nel repository UCI[1]. Nello specifico la scelta è ricaduta sul database "Breast Cancer", contenente informazioni riguardanti le analisi effettuate su 683 pazienti e la relativa diagnosi sul tumore riscontrato: si è deciso l'utilizzo di questo dataset sia per le caratteristiche adatte a prestarsi come base di dati per la classificazione, sia per l'efficacia con cui si potranno confrontare i risultati ottenuti. Contiene undici attributi, riportati nella tabella 4.1.

La distribuzione di classe di questo dataset è:

- Benigno: 444 (65%)
- Maligno: 239 (35%)

L'utilizzo di questo dataset aiuta a far capire chiaramente quale sia l'importanza della gestione dell'incertezza: partendo dalla versione certa fornita dal repository UCI, possiamo stabilire, dopo aver istruito il classificatore, se in base al valore dei suoi parametri un paziente ha un tumore benigno o maligno.

In realtà l'affermazione che facciamo non è sicura, poiché nella maggior parte dei casi non si può stabilire con certezza la natura del tumore: è quindi

[1] **URL:** *http://archive.ics.uci.edu/ml/datasets.html*

#	**Attributo**	**Dominio**
1	Sample code number	ID numerico
2	Clump Thickness	$1 \sim 10$
3	Uniformity of Cell Size	$1 \sim 10$
4	Uniformity of Cell Shape	$1 \sim 10$
5	Marginal Adhesion	$1 \sim 10$
6	Single Epithelial Cell Size	$1 \sim 10$
7	Bare Nuclei	$1 \sim 10$
8	Bland Chromatin	$1 \sim 10$
9	Normal Nucleoli	$1 \sim 10$
10	Mitoses	$1 \sim 10$
11	Class	1 se benigno, 2 se maligno

Tabella 4.1: Attributi dataset Breast Cancer

ragionevole dire che i dati originali su cui si istruisce il classificatore sono in realtà incompleti. Se invece avessimo un dataset di partenza contenente dati incerti, nel nostro caso quindi una classificazione incerta, potremmo essere in grado di assegnare un valore di *confidenza* all'affermazione che facciamo, senza essere obbligati a distinguere nettamente tra benigno e maligno.

Questo nuovo scenario consente una migliore gestione delle informazioni di cui siamo in possesso e quindi una più adeguata risposta al risultato. Se nel caso di dati incerti siamo capaci di stabilire, per esempio, che un tumore ha il 60% di probabilità di essere benigno, allora può essere una buona scelta indagare meglio sulla natura dello stesso, effettuando altre analisi ed accertamenti, vista la poca confidenza nell'affermazione che facciamo. Nel caso di dati certi invece lo stesso tumore verrebbe semplicemente classificato come benigno, con conseguenze potenzialmente molto gravi.

Un discorso analogo lo si può fare per altre situazioni ed altri dataset: l'individuazione di terroristi e il riconoscimento di evasori fiscali sono validi esempi di come questi concetti possano essere applicati in molti ambiti.

4.3 Implementazione dell'estensione

L'implementazione in **Weka** ha richiesto l'estensione di svariate porzioni di codice, in quanto l'incertezza sull'attributo di classe influisce sul calcolo del gain_ratio, della classe da associare ad un nodo e sulla predizione della classe da attribuire ad un'istanza da classificare.

Inizialmente sono stati ampliati i "tipi" gestiti nel *core* di **Weka**, per consentire l'utilizzo di un attributo incerto come attributo di classe. È poi stata aggiunta una classe al package dei classificatori, che identificasse il nostro classificatore incerto, ereditando i metodi principali dalla super classe. In seguito si è reso necessario modificare tutte le parti di codice in cui veniva usato l'attributo di classe, poiché non si ha più un singolo intero a rappresentarla, ma un array bidimensionale composto da tutte le classi con le relative probabilità associate.

Una volta terminata la parte di scrittura di codice sono stati eseguiti i primi test, per verificare la correttezza dell'estensione. In particolare, sfruttando la funzione di visualizzazione della struttura dell'albero decisionale messa a disposizione dall'interfaccia grafica di Weka, sono state confrontate le diverse strutture degli alberi decisionali ricavati.

Per l'esecuzione dei test era necessario modificare il dataset inziale, aggiungendo incertezza all'attributo di classe. Essendo presenti un grande numero di tuple è stato sviluppato uno script in *bash* [24], che prendendo come parametri il file di origine, la percentuale di tuple da rendere incerta e il livello di *confidenza* da associare alla classe del record, restituisce un nuovo file dove i record da modificare presentano l'attributo di classe incerto a seconda della soglia passata come parametro.

4.3.1 Verifica della correttezza

La prima analisi eseguita è stata quella atta a verificare che l'algoritmo implementato fosse una reale estensione dell'algoritmo di partenza. Per fare ciò è stato lanciato l'algoritmo di classificazione C4.5 nativo di Weka sul dataset iniziale, senza valori incerti: in questo modo è stato possibile ricavare la struttura base per tutti i confronti successivi, riportata in figura 4.1.

Per eseguire il primo test è stato dunque creato un dataset incerto tramite lo script bash, a cui sono stati passati come parametri `100%` per quanto riguarda il numero di tuple da rendere incerte e `1` per quanto riguarda la confidenza da associargli, in modo da ottenere gli stessi valori del dataset certo, ma con la notazione incerta. A questo punto abbiamo lanciato l'algoritmo esteso *DTCU* su questo nuovo dataset, ottenendo la struttura in figura 4.2.

È facile notare come questa struttura sia del tutto identica alla struttura originale in figura 4.1: questo ci conferma che l'algoritmo DTCU è una reale estensione dell'algoritmo C4.5 implementato da Weka, in quanto, partendo dagli stessi valori nel dataset, produce una struttura completamente uguale a quella prodotta dall'algoritmo originale.

4.4 Risultati sperimentali

Una volta verificata l'affidabilità dell'algoritmo, tramite l'analisi effettuata al paragrafo precedente, è stato possibile iniziare ad eseguire i test. Lo scopo è di confrontare le strutture ottenute, in modo da comprendere quanto l'incertezza possa influire sulla costruzione dell'albero decisionale.

Abbiamo analizzato l'influenza sia della soglia di incertezza, che del numero di record ai quali questa è applicata, dividendo i test in quattro macro categorie:

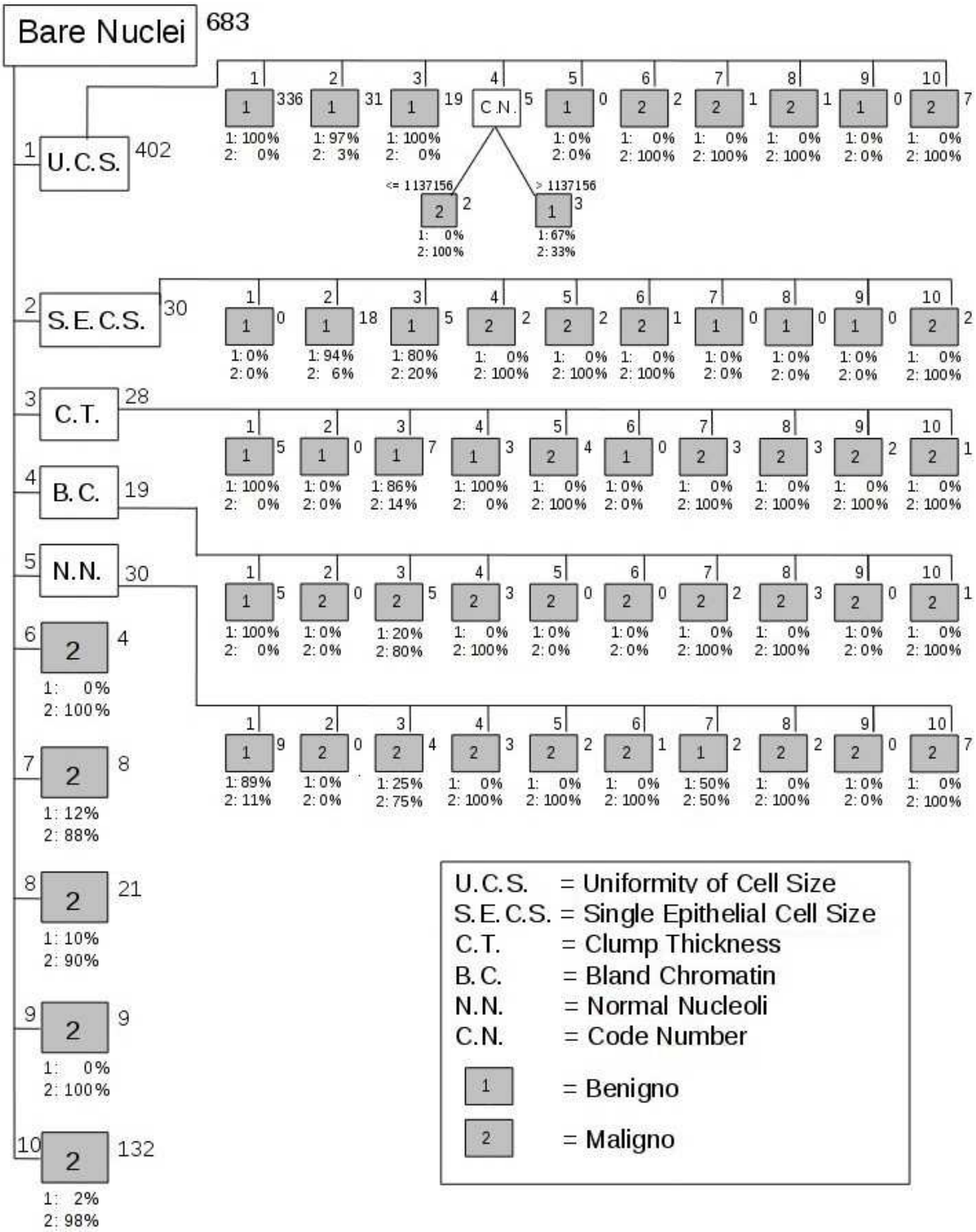

Figura 4.1: Struttura dell'albero decisionale ottenuta dal dataset certo

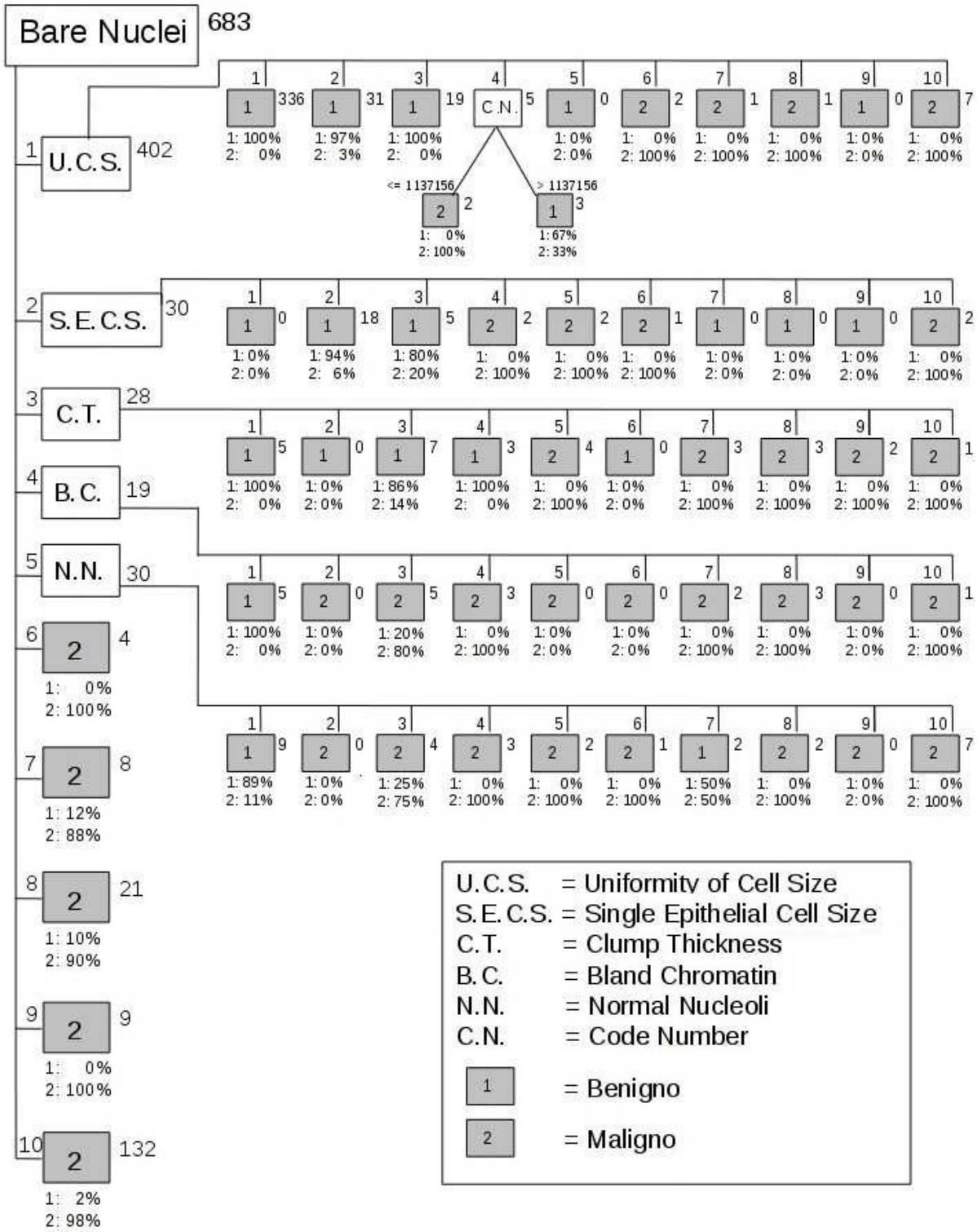

Figura 4.2: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza a 1 (su tutti i record)

- tutti i record incerti;
- metà dei record incerti;
- un quarto dei record incerti;
- un decimo dei record incerti.

Per ognuna di queste categorie è stata variata la *confidenza* associata alle tuple, partendo da `1` fino ad arrivare a `0.50`.

4.4.1 Tutti i record incerti

Rendendo incerti tutti i record si ricade nella situazione in cui l'incertezza ha sicuramente maggiore influenza. Infatti, variando la confidenza nel range `1`∼`0.50`, si ottengono ben otto strutture differenti.

Fino al valore di confidenza `0.97` abbiamo una struttura identica a quella base, riportata in figura 4.1: questo ci indica che, mantenendo un alto livello di confidenza, l'albero decisionale che costruiamo non cambia. Si può pensare che sia un comportamento auspicabile, in quanto, se abbiamo alti livelli di confidenza, siamo quasi completamente certi della nostra affermazione e ci si può aspettare di ottenere gli stessi risultati ottenuti nel caso certo[2].

La prima soglia a cui si verifica qualche differenza è al valore `0.96`. La struttura è riportata in figura 4.3 e si può notare come la differenza in questo caso sia davvero minima, andando ad influenzare solo la generazione dei figli del nodo avente `Bare Nuclei = 1` e `Uniformity of Cell Size = 4`

[2]È da specificare che, per diverse soglie di incertezza, cambia la percentuale di record appartenenti ad una classe o all'altra in ogni nodo: si è scelto di non rappresentare ogni singolo caso per una migliore leggibilità dell'elaborato, riportando solamente i casi per cui si verifica un cambiamento della struttura. Tale discorso è applicabile a tutte le strutture associate ad un intervallo di valori che verranno di seguito riportate.

Vi è poi una maggiore differenza con la soglia `0.95`. La struttura è riportata in figura 4.4 e si può osservare che in tale situazione non vengono generati i figli del nodo `Bare Nuclei = 4`, ma tale nodo diventa una foglia.

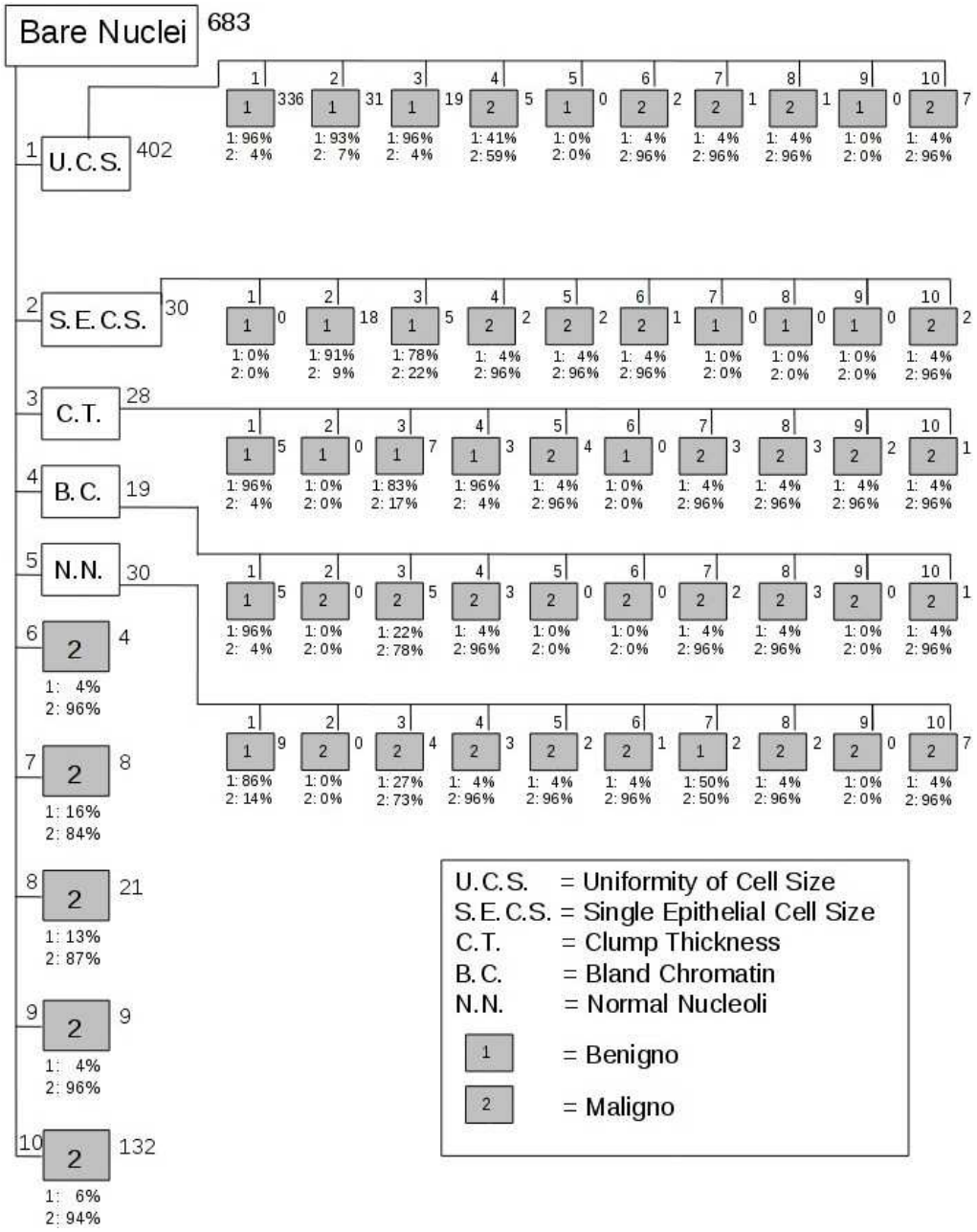

Figura 4.3: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza a 0.96 (su tutti i record)

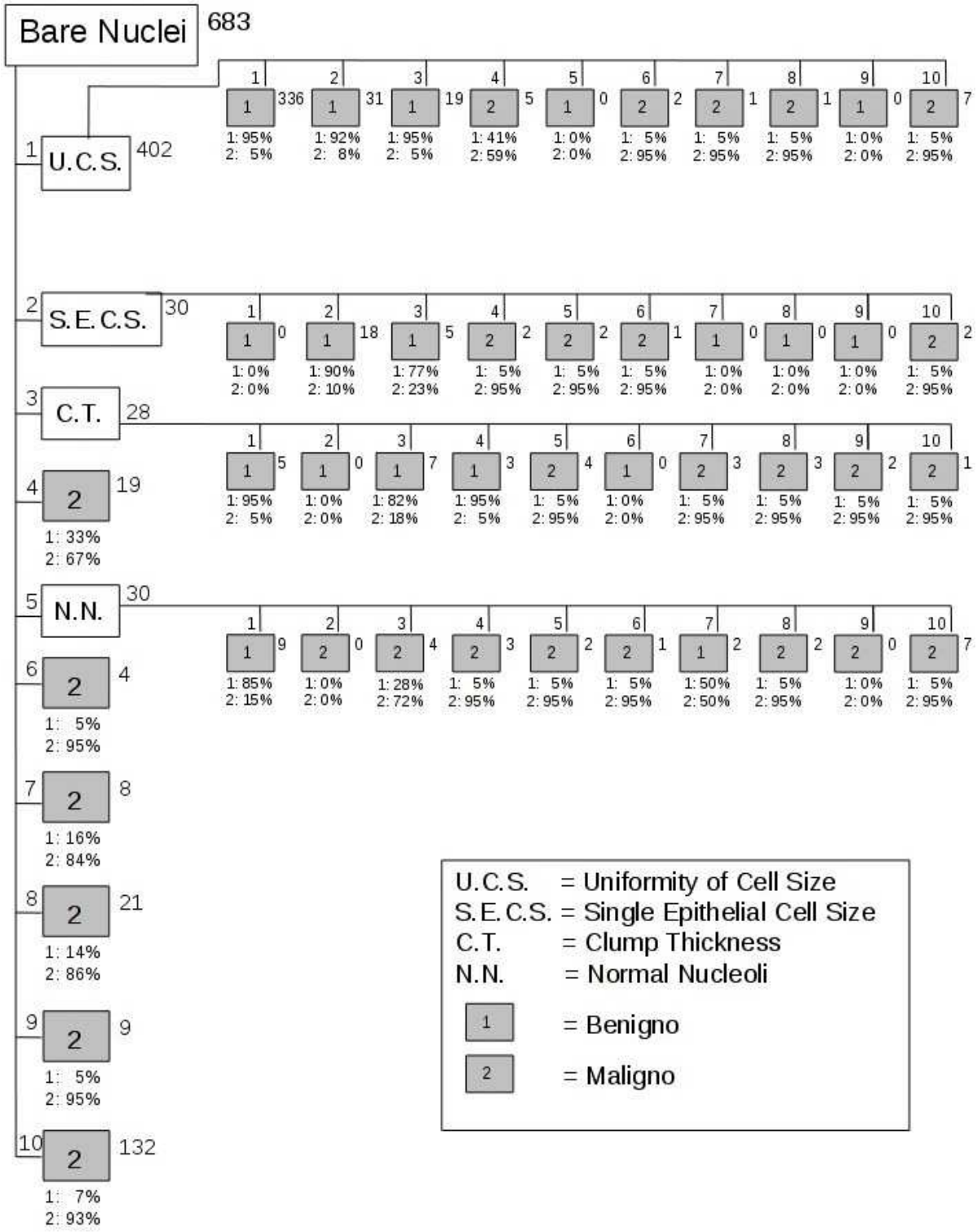

Figura 4.4: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza a 0.95 (su tutti i record)

Abbassando ancora la confidenza otteniamo una struttura visibilmente diversa per i casi da `0.94` a `0.84`. In questa situazione cambia il primo attributo di split, dando in questo modo origine ad una struttura completamente diversa rispetto al caso di partenza. Questo accade a causa del valore di gain_ratio calcolato per gli attributi, che con queste soglie di incertezza favorisce l'attributo `Uniformity of Cell Size`. La struttura risultante è riportata in figura 4.5.

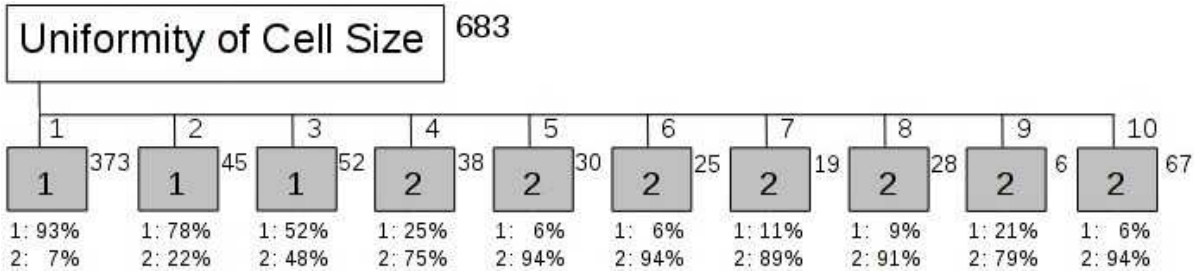

Figura 4.5: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.94 a 0.84 (su tutti i record)

Quando il livello di confidenza va da `0.83` a `0.81` torna ad essere scelto come primo attributo di split da `Bare Nuclei`. Possiamo quindi notare dalla figura 4.6 come si torni ad avere una struttura simile a quella originale, ma con alcune differenze sui figli al primo livello: in questo caso vengono infatti ri-splittati solamente il secondo e il terzo figlio, tutti gli altri sono nodi foglia.

Proseguendo troviamo una struttura ancora diversa per valori di confidenza da `0.80` a `0.75`, riportata in figura 4.7. In questo caso solamente sul terzo figlio viene ricalcolato l'attributo di split, mentre tutti i restanti figli sono foglie.

Andiamo adesso infine ad analizzare le ultime due strutture del caso in cui si rendano incerti tutti i record. La prima è relativa al caso in cui la soglia sia compresa tra `0.74` e `0.56` e, come possiamo osservare dalla figura

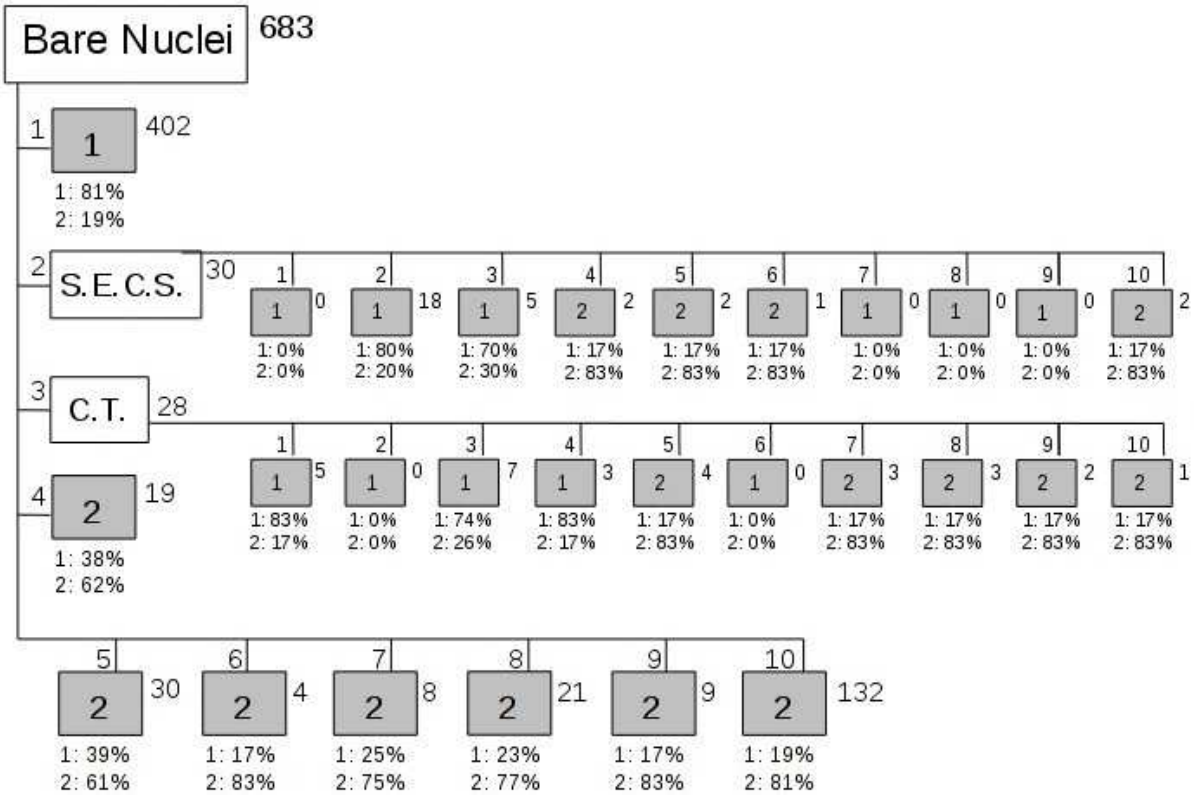

Figura 4.6: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.83 a 0.81 (su tutti i record)

4.8, presenta solamente lo split effettuato sul nodo radice, avendo già come foglie tutti i nodi di primo livello.

La seconda struttura invece, relativa a soglie da `0.55` a `0.50` e visualizzata in figura 4.9, presenta un solo nodo foglia, senza nessuna divisione dei record. Tale struttura è sicuramente la struttura per l'albero decisionale meno specifico che si possa costruire. È tuttavia normale trovarsi di fronte a questa situazione, in quanto con valori di confidenza così bassi è ragionevole non riuscire ad effettuare una scelta ben distinta relativamente alla classe da assegnare a nuove istanze da classificare.

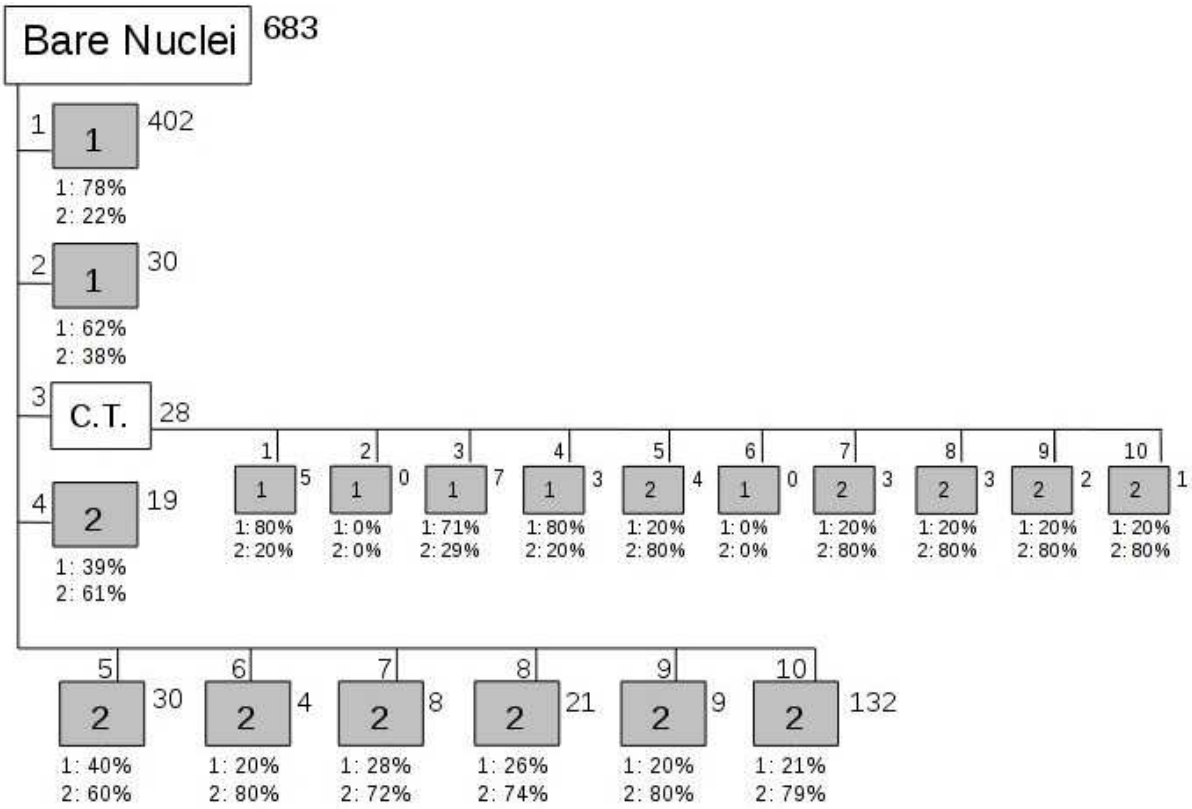

Figura 4.7: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.80 a 0.75 (su tutti i record)

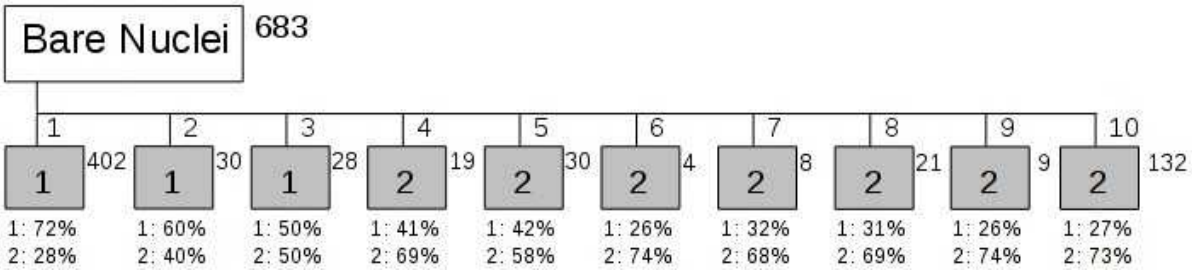

Figura 4.8: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.74 a 0.56 (su tutti i record)

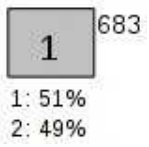

Figura 4.9: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.55 a 0.50 (su tutti i record)

4.4.2 Metà dei record incerti

Passiamo ora a verificare le variazioni alla struttura dell'albero nel caso in cui siano incerti solo la metà dei record. Tramite lo script bash descritto sopra è stato possibile rendere incerta la classe di solamente la metà delle tuple, modificando il valore delle altre in modo da ottenere notazione incerta ma valori certi (come si era fatto nel caso discusso precedentemente relativo alla verifica dela correttezza). Avendo incertezza sulla metà dei record e variando la confidenza nel range `1`∼`0.50`, si ottengono sei strutture differenti.

La prima lieve differenza nella struttura si può notare già a partire dalla soglia `0.99`, la cui struttura è riportata in figura 4.10. È immediato osservare che le foglie con valore di `Clump Thickness` uguale a 2 e 6, figlie di "`Bare Nuclei` = 3", non hanno associato nessun record: la classe loro assegnata in questo caso è `2`, anziché, come avviene nel caso originale, la `1`. Questo cambiamento avviene perché l'algoritmo, non potendo assegnare la classe al nodo a partire dai suoi record, gli assegna la classe a cui appartiene il numero maggiore dei record in quel ramo.

Un altro cambiamento nella struttura si verifica al valore `0.92`, la cui struttura è visualizzata in figura 4.11. Si può notare come le differenze siano site in due punti: il primo riguarda l'attributo di split del secondo figlio, che non è più `Single Epithelial Cell Size` ma `Uniformity of Cell Size`, il secondo l'espansione del quarto figlio, che in questo caso non avviene.

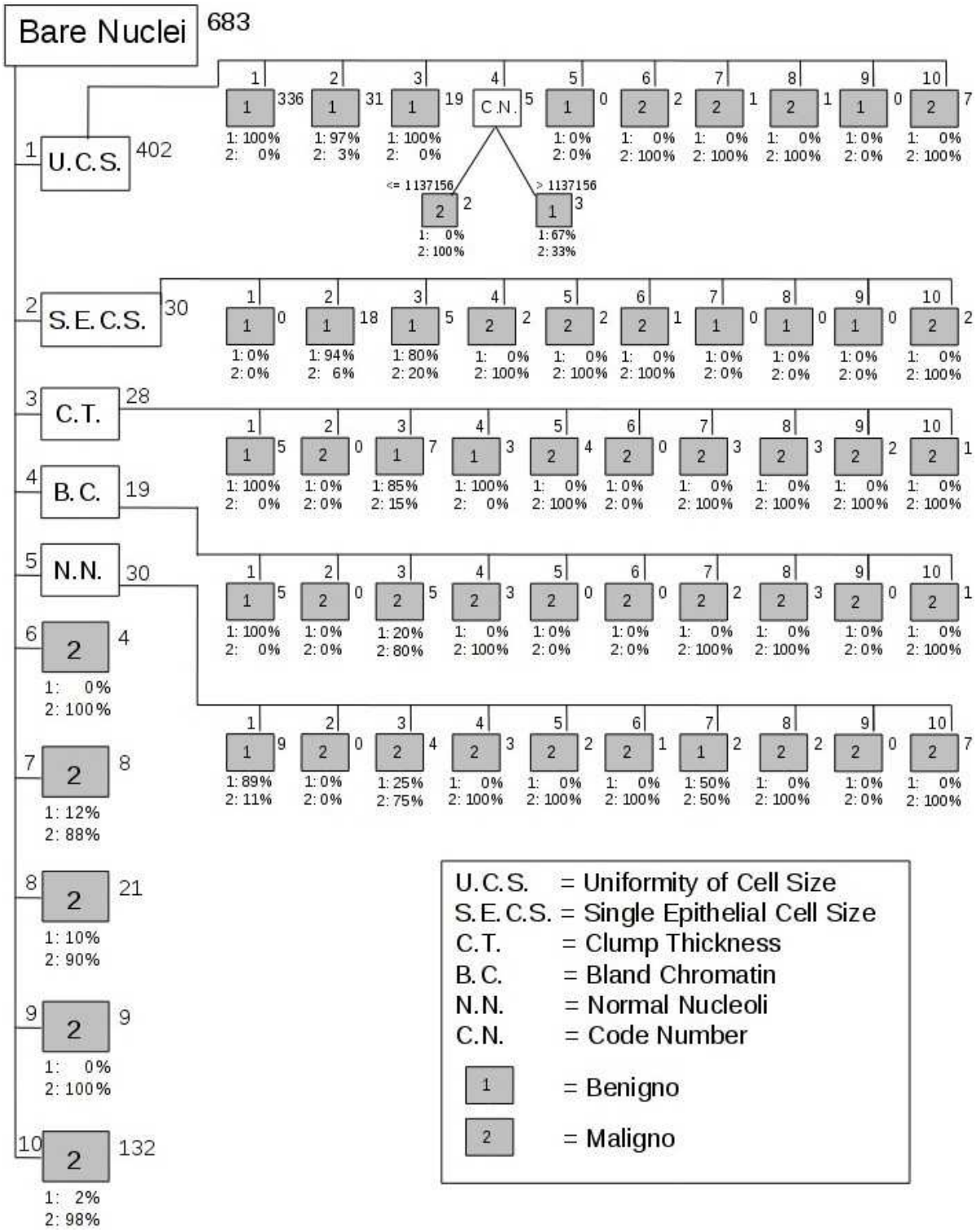

Figura 4.10: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.99 a 0.93 (su metà dei record)

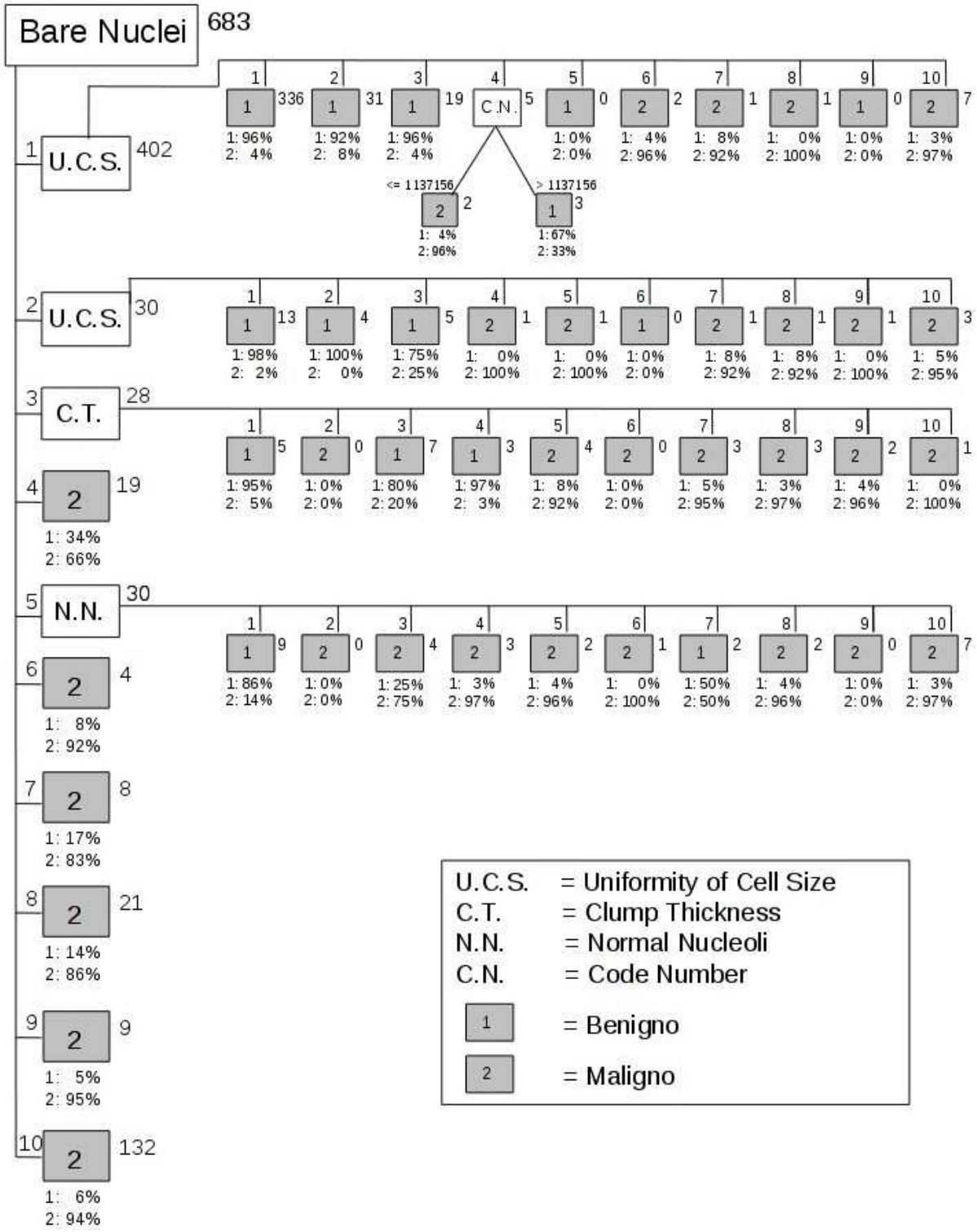

Figura 4.11: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.92 a 0.90 (su metà dei record)

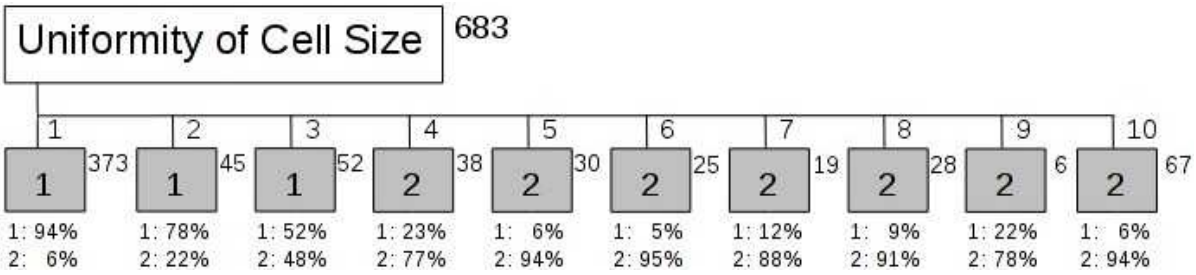

Figura 4.12: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.89 a 0.68 (su metà dei record)

Abbassando ancora la confidenza otteniamo una struttura visibilmente diversa per i casi da `0.89` a `0.68`. In questa situazione, come accadeva con soglia a `0.94` nel caso si rendessero incerti tutti i record, cambia il primo attributo di split, dando in questo modo origine ad una struttura completamente diversa rispetto al caso di partenza. Vale anche qui il discorso relativo al gain_ratio fatto prima; la struttura risultante è riportata in figura 4.12.

Quando il livello di confidenza va da `0.67` a `0.64` torna ad essere scelto come primo attributo di split `Bare Nuclei`. Possiamo quindi notare dalla figura 4.13 come si torni ad avere una struttura ricalcante quella originale, ma con differenze sui figli al primo livello: in questo caso viene infatti ri-splittato solamente il terzo figlio, mentre tutti gli altri sono nodi foglia.

Proseguendo troviamo una struttura ancora diversa per valori di confidenza da `0.63` a `0.56`. In questo caso la struttura è molto simile al caso appena precedente, ma l'attributo di split scelto per il terzo figlio diventa `Single Epithelial Cell Size` anziché `Uniformity of Cell Size`. La struttura risultante è visualizzabile in figura 4.14.

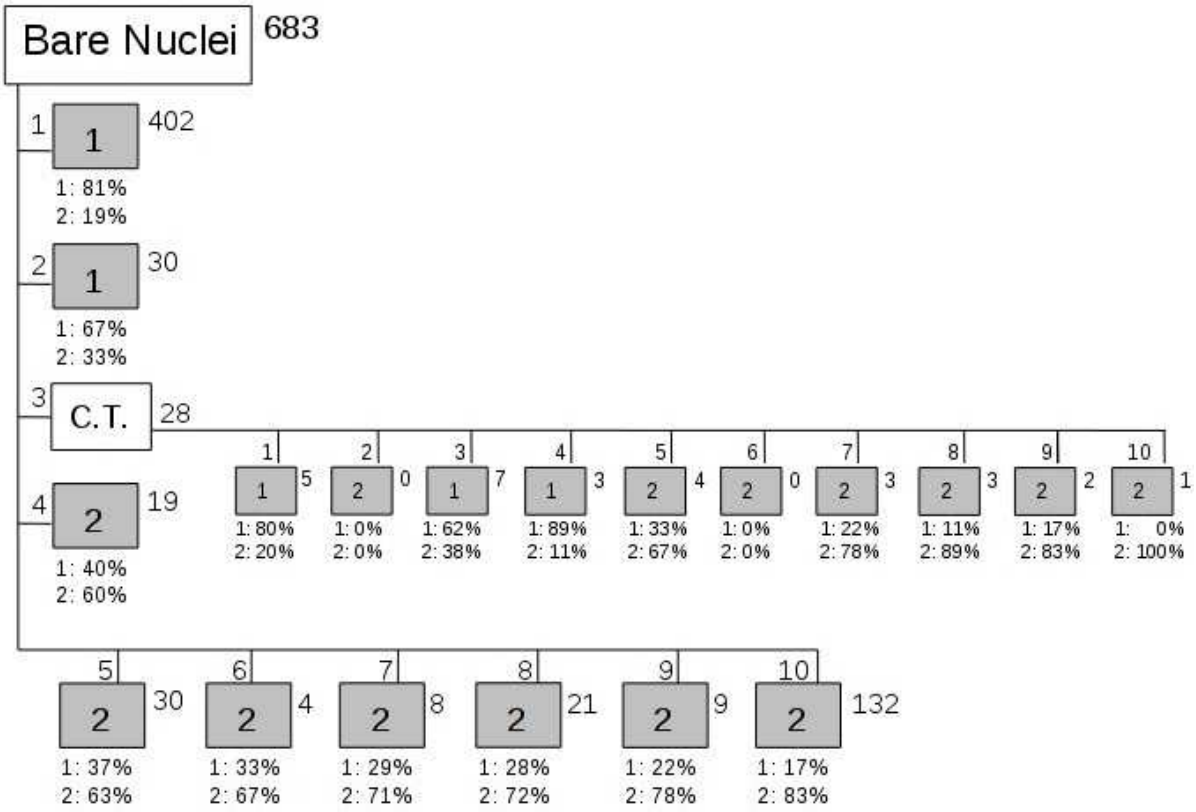

Figura 4.13: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.67 a 0.64 (su metà dei record)

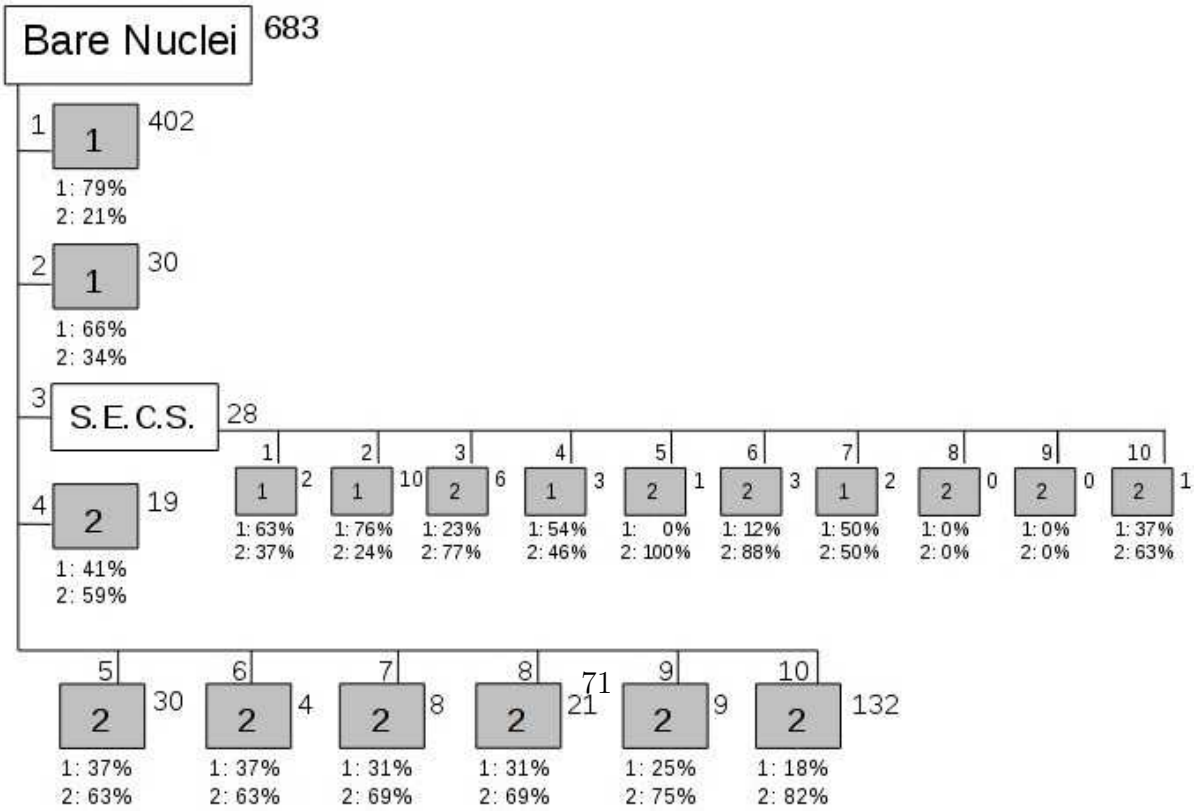

Figura 4.14: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.63 a 0.57 (su metà dei record)

Andiamo adesso infine ad analizzare l'ultima struttura del caso in cui si rendano incerti metà dei record. La soglia in questa situazione sarà compresa tra `0.56` e `0.50` e, come possiamo osservare dalla figura 4.15, presenta solamente lo split effettuato sul nodo radice, avendo già come foglie tutti i nodi di primo livello.

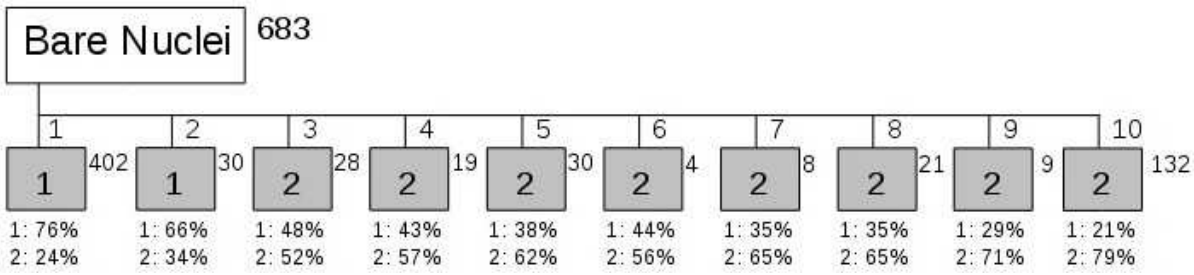

Figura 4.15: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.56 a 0.50 (su metà dei record)

4.4.3 Un quarto dei record incerti

Verifichiamo ora le modifiche alla struttura dell'albero nel caso in cui sia incerto solo un quarto dei record. Come in precedenza, tramite lo script è stato modificato il dataset in modo da rendere incerta la classe di solamente un quarto delle tuple. Trasformando così il database di origine e variando la confidenza nel range `1`∼`0.50`, si ottengono quattro strutture differenti.

Anche in questo caso già a partire dalla soglia `0.99` si può notare la minima differenza relativa alle foglie con valore di `Clump Thickness` uguale a 2 o 6, figlie di "`Bare Nuclei` = 3", che non hanno associato nessun record. Classificaranno future istanze con la classe `2`, poiché anche in questo caso l'algoritmo, non potendo assegnare la classe al nodo a partire dai suoi record, gli assegna la classe a cui appartiene il numero maggiore dei record in quel ramo, nel nostro caso quelli aventi `Bare Nuclei` uguale a 3. La struttura è osservabile in figura 4.16

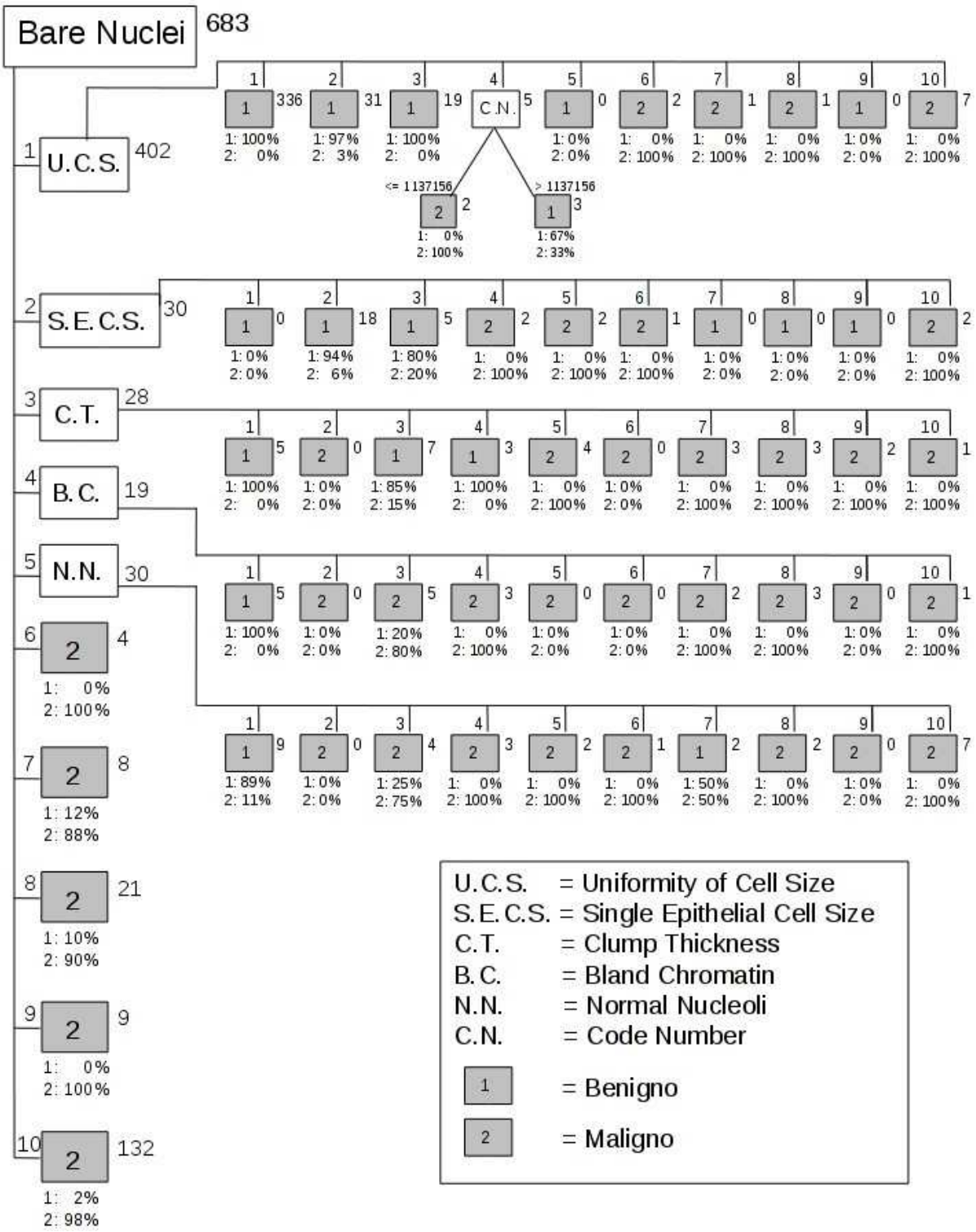

Figura 4.16: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.99 a 0.97 (su un quarto dei record)

Un altro cambiamento nella struttura si verifica al valore `0.96`. La struttura è riportata in figura 4.17 e si può notare come la differenza risieda nel quinto figlio, che non viene più splittato ma diventa un nodo foglia.

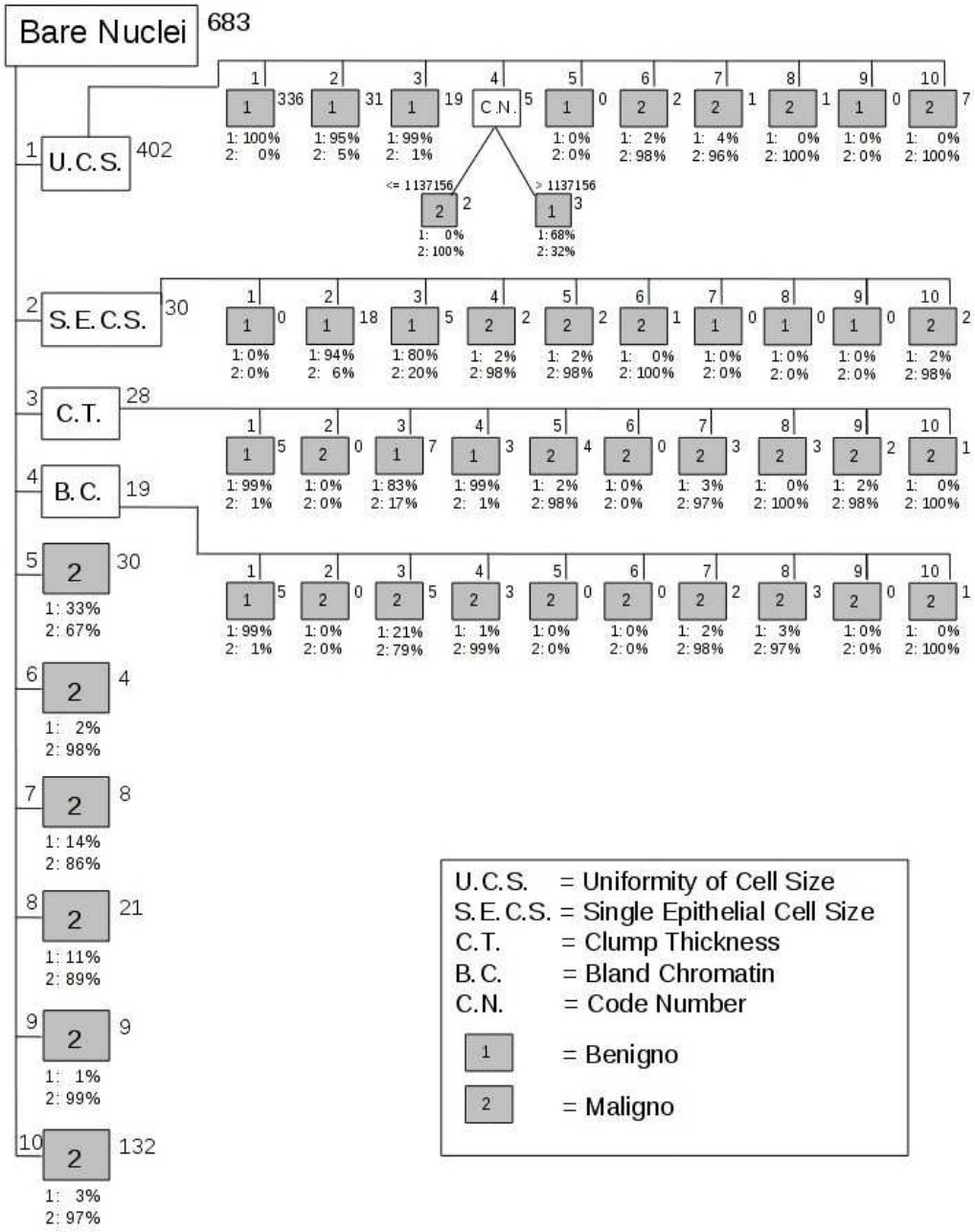

Figura 4.17: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.96 a 0.92 (su un quarto dei record)

Proseguendo si può notare un altro cambiamento nella struttura al valore `0.91`, visualizzato in figura 4.18. La differenza è sita nell'attributo di split del secondo figlio, che diventa `Uniformity of Cell Size`.

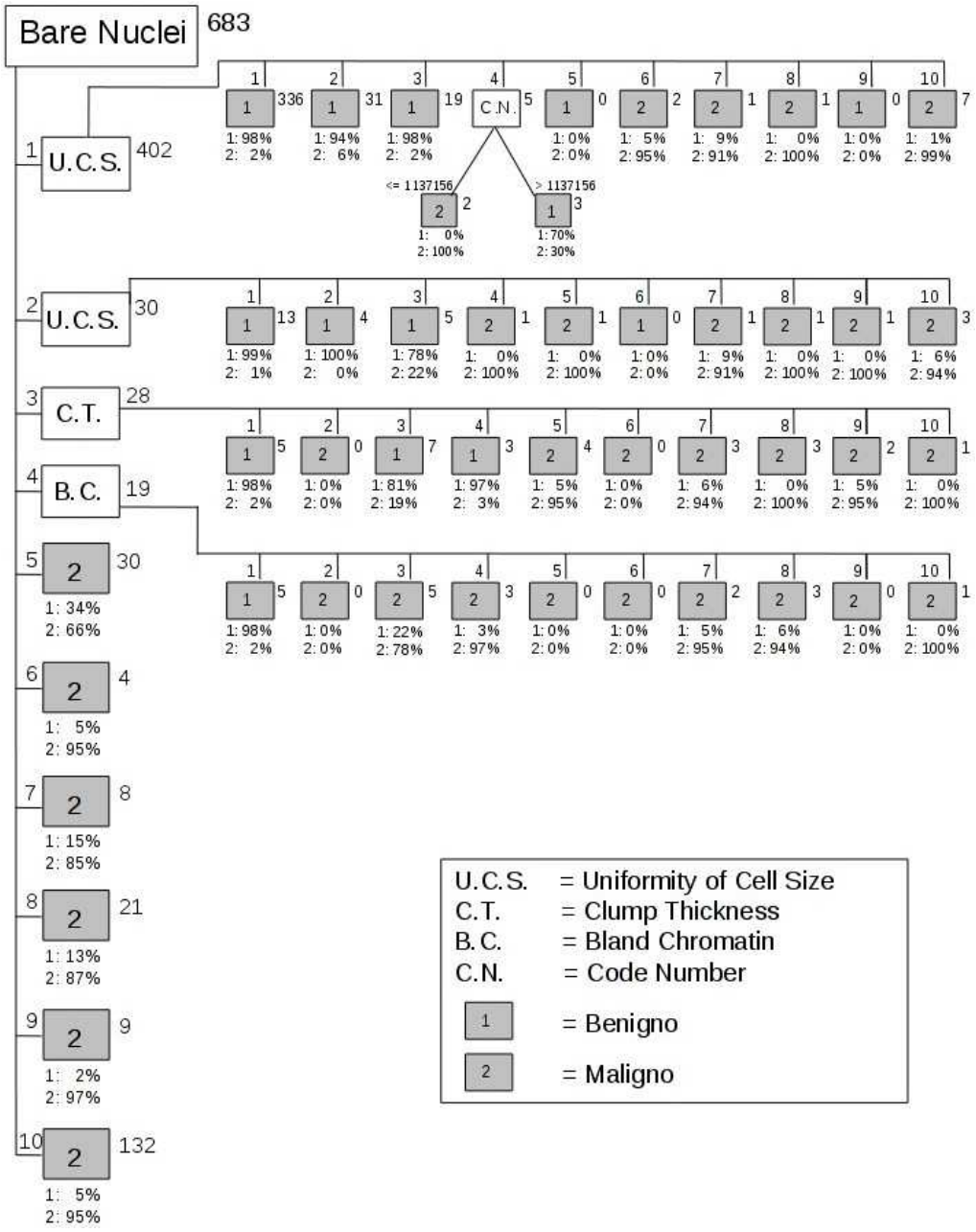

Figura 4.18: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.91 a 0.74 (su un quarto dei record)

Andiamo infine ad analizzare l'ultima struttura del caso in cui si renda incerto un quarto dei record. La soglia in questa situazione sarà compresa tra `0.73` e `0.50` e, come possiamo osservare dalla figura 4.19, presenta solamente lo split effettuato sul nodo radice, avendo già come foglie tutti i nodi di primo livello. Si fa notare come l'attributo scelto sia `Uniformity of Cell Size`, a differenza delle ultime strutture dei casi precedenti per cui era `Bare Nuclei`.

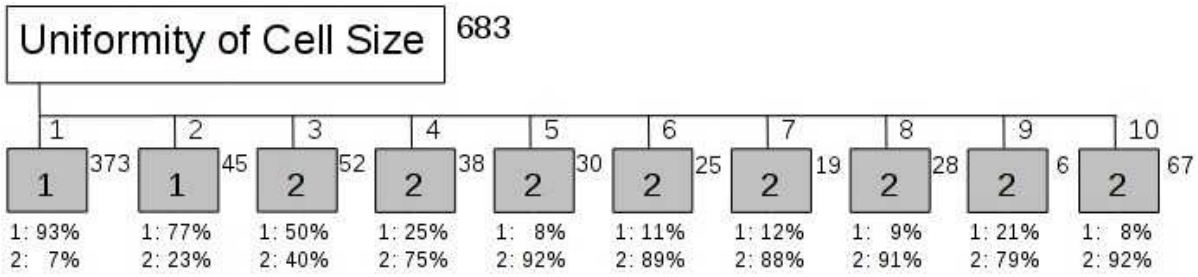

Figura 4.19: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.73 a 0.50 (su un quarto dei record)

4.4.4 Un decimo dei record incerti

Passiamo all'ultima verifica, quella relativa al caso in cui sia incerto solamente un decimo dei record. Come prima, l'operazione di rendere incerti i record del dataset è stata effettuata tramite lo script bash. Trasformando in tale modo il dataset di origine e variando la confidenza nel range `1`$\sim$`0.50` si ottengono due sole strutture differenti.

La prima si ha dalla soglia `0.99`, in cui si può notare, come nei casi precedenti, la minima differenza relativa alle foglie con valore di `Clump Thickness` uguale a 2 o 6, figlie di "`Bare Nuclei` = 3", che classificheranno future istanze con la classe `2`, anziché `1` come avviene nel caso originale. La struttura è osservabile in figura 4.20

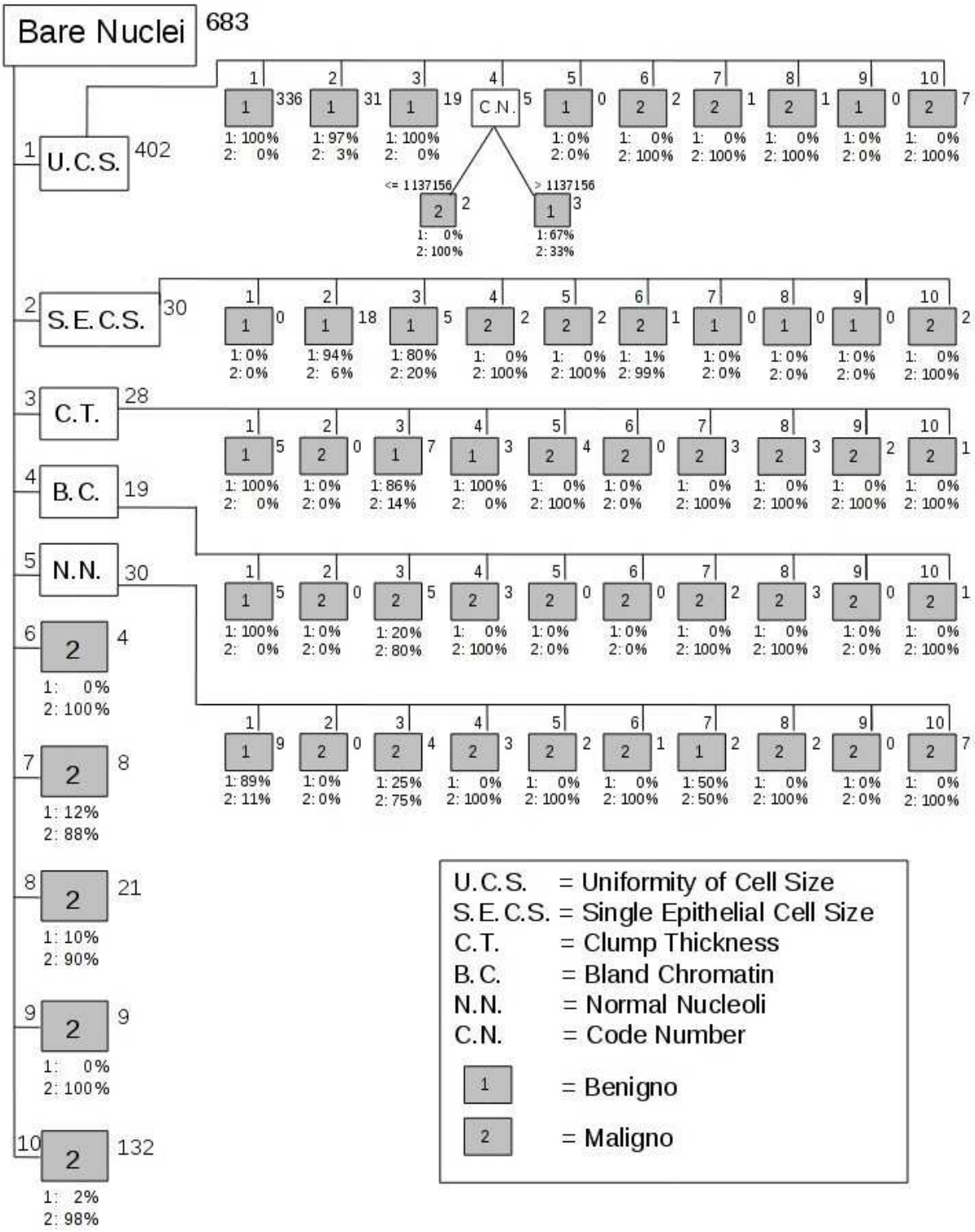

Figura 4.20: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.99 a 0.97 (su un decimo dei record)

L'altro solo cambiamento nella struttura si verifica al valore `0.96` ed è riportato in figura 4.21. Si può notare come la differenza risieda nel quinto figlio, che non viene più splittato ma diventa un nodo foglia.

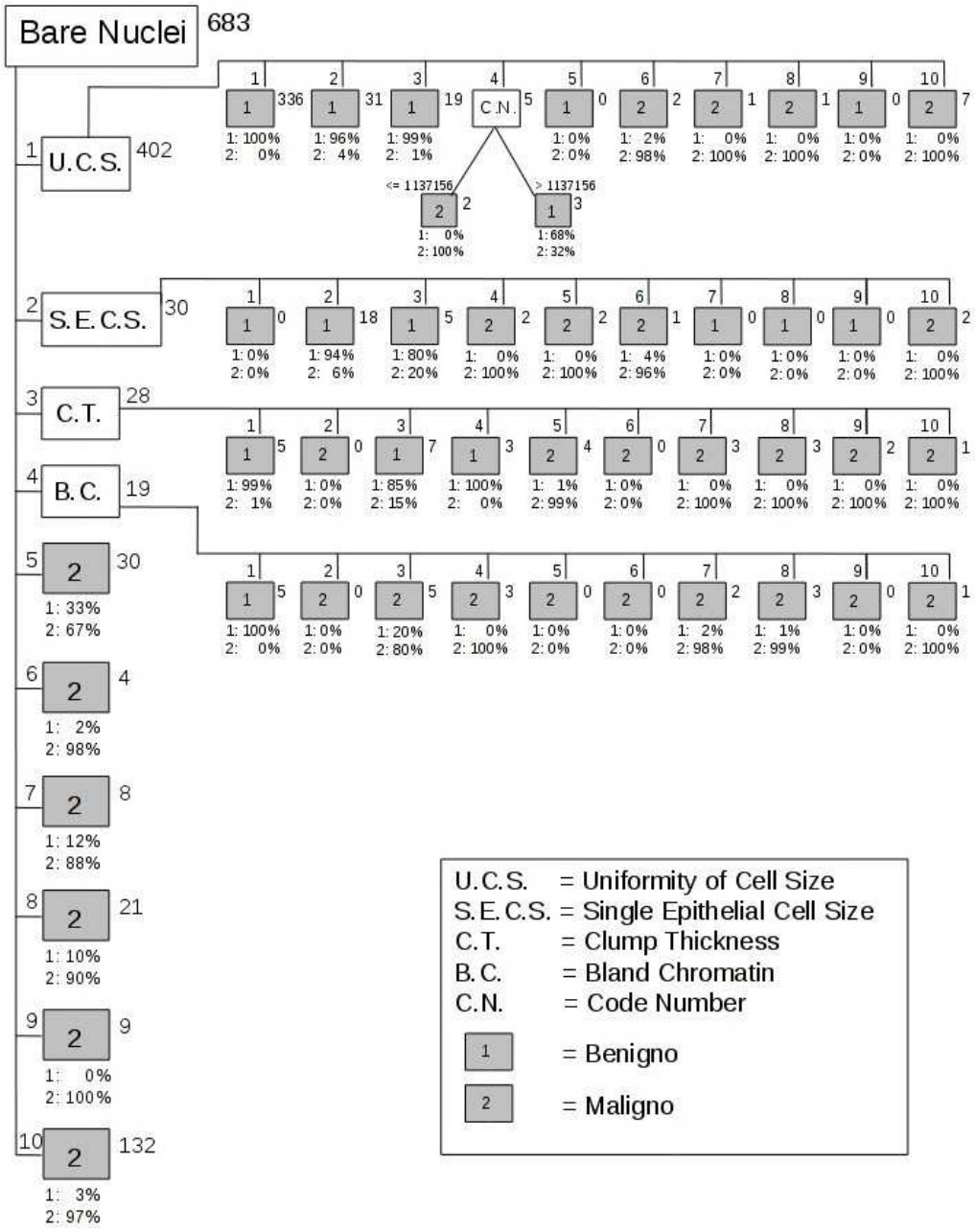

Figura 4.21: Struttura dell'albero decisionale ottenuta dal dataset incerto con confidenza da 0.96 a 0.50 (su un decimo dei record)

4.5 Analisi dei risultati

Analizzando le strutture ottenute si possono estrarre diversi concetti.

Il primo di questi, già evidenziato in precedenza, è che l'algorimo DTCU è una effettiva estensione dell'algorimo originale C4.5 di Weka. Infatti, sia dando in input il dataset originale all'algoritmo implementato in Weka che dando in input il dataset con notazione incerta all'algoritmo DTCU, si ottengono gli stessi risultati di classificazione. Tutto ciò è testimoniato dalle figure 4.1 e 4.2, dove si può osservare che non vi è alcuna differenza nella struttura dell'albero decisionale.

Il secondo risultato che possiamo notare è che, anche con soglie di confidenza molto alte, la struttura ottenuta cambia rispetto all'originale. Ciò è osservabile, per esempio, in figura 4.10, dove la soglia è del `99%` e sono incerti la metà dei record. In questo caso la variazione della struttura è quasi impercettibile, ma evidenzia come l'incertezza, anche se presente in modo quasi irrilevante, svolga un ruolo importante nel processo di classificazione. Questo fa capire che riuscire a gestire l'incertezza, anche quando è molto vicina a valori sicuri, permette di essere più precisi nella definizione di modelli di classificazione e quindi di ottenere risultati migliori.

Un altro risultato molto interessante che è emerso e che inizialmente non avevamo previsto è che, rendendo incerti tutti i record con confidenza al `50%`, la struttura dell'albero risultante è composta da una sola foglia, come riportato in figura 4.9. Tale struttura fornisce sicuramente la peggior classificazione possibile, assegnando semplicemente tutti i record ad una sola classe. È tuttavia esattamente il comportamento che si dovrebbe avere, poiché, riflettendo sul dataset preso in input, nessun record fa parte in maniera evidente di una specifica classe, ma hanno tutti uguale probabilità di appartenere ad una delle due classi. Perciò, in questa situazione, è normale non saper ricavare

un modello di classificazione specializzato.

Infine, da un'analisi globale, si può evincere una continuità nel numero di strutture ottenute rendendo incerti i record in diverse percentuali. Tale risultato è ben rappresentato in figura 4.22, dove tramite la tabella riassuntiva ed il grafo associato è possibile capire facilmente come vari il numero di strutture in base alla percentuale di record incerti.

% Record Incerti	# Strutture Differenti
10	2
25	4
50	6
100	8

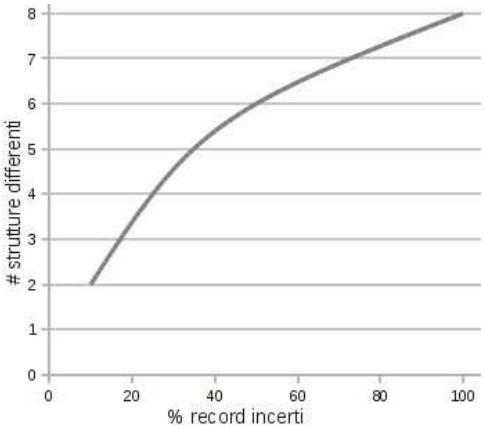

Figura 4.22: Tabella e grafico riassuntivi: % record incerti → # strutture differenti

Capitolo 5

Conclusioni e sviluppi futuri

Nei capitoli precedenti sono stati illustrati e analizzati algoritmi per il data mining su dati incerti, soffermandosi in maniera più precisa su quelli relativi alla classificazione tramite alberi decisionali. Di questi ne è stato scelto uno come base per il lavoro di tesi, il cui obiettivo era estendere la classificazione a dataset che presentino incertezza sull'attributo di classe.

Si è quindi proceduto all'identificazione specifica del problema, anche tramite esempi che permettessero di capire immediatamente le diverse situazioni che si volevano gestire. Una volta fatto questo è stato formalizzato un frammento di pseudocodice che descrivesse esaustivamente l'estensione da implementare, in modo da avere una base chiara da cui partire con la scrittura del codice. La nostra estensione DTCU è stata realizzata in linguaggio Java, partendo dall'algoritmo C4.5 fornito dal software Weka. La scelta di estendere un algoritmo già esistente ha consentito di poter poggiare il lavoro su solide basi e aver utilizzato Weka ci ha permesso di ottenere tool di visualizzazione grafica, utili all'analisi dei risultati, già disponibili nel pacchetto software.

Siamo infine passati all'esecuzione di prove sperimentali. Il primo test

è servito a verificare la correttezza dell'estensione: prendendo in input un dataset con classe annotata come incerta, ma che in realtà presenta valori certi, è stato osservato che l'algoritmo presentato in questa tesi produce lo stesso risultato prodotto dall'algoritmo base sul dataset con classe certa. Era esattamente il comportamento aspettato, che certifica sia che l'algoritmo DTCU funziona correttamente, sia che è una reale estensione del caso certo.

I successivi test sono stati invece effettuati sul dataset con valori di incertezza variabili e con percentuale delle tuple incerte che andava dal totale ad un decimo. L'analisi dei risultati ha portato a dedurre diverse nozioni: quando si ha incertezza massima, cioè tutte le tuple incerte al `50%`, l'albero ottenuto consiste in una sola foglia, che è la struttura decisionale peggiore che si possa ottenere, in quanto non consente l'ottenimento di nessun modello. Un altro aspetto che è stato evidenziato è che, anche con soglie di confidenza molto alte, la struttura dell'albero cambia, seppure di poco, rispetto all'orginale: questo indica come, fin da subito, l'incertezza influisce sull'albero decisionale. Infine è stato osservato che, al diminuire della percentuale di tuple incerte, diminuiscono anche le diverse strutture per l'albero decisionale generate dalle diverse soglie di incertezza.

Tutto questo è utile a capire che, riuscendo a gestire l'incertezza, siamo anche in grado di fornire classificazioni più accurate e che possano portare a decisioni migliori.

Un possibile sviluppo futuro per quanto riguarda l'algoritmo DTCU è sicuramente l'analisi di metriche di confronto tra le diverse strutture ottenute, che tengano conto delle percentuali assegnate alle classi nei nodi foglia e che sappiano riconoscere la differenza nella classificazione di nuove istanze.

Una prima intuizione è che, in un caso base in cui si ipotizzano di avere solamente due attributi, dividere i record prima per uno e poi per l'altro o

viceversa conduce ad ottenere quattro nodi foglia che sono esattamente gli stessi, anche se in ordine diverso. In questo caso la differenza tra le due strutture deve essere nulla, nonostante risultino differenti, poiché i nuovi record saranno classificati in maniera identica.

Una metrica di confronto che consenta di verificare la differenza tra due strutture può permettere di capire al meglio cosa implichi tenere conto dell'incertezza, anche su diversi livelli percentuali, rispetto ad approssimare tutto a dati certi.

Bibliografia

1. C. C. Aggarwal. *Managing and Mining Uncertain Data*, pages 389-459. Springer, Feb. 2009.

2. C. C. Aggarwal, Yan Li, Jianyong Wang and Jing Wang. Frequent Pattern Mining with Uncertain Data. In *ACM KDD Conference*, 2009.

3. C. C. Aggarwal and P. S. Yu. A Survey of Uncertain Data Algorithms and Applications. In *IEEE TKDE*, May 2009.

4. C. C. Aggarwal and P. S. Yu. Outlier Detection with Uncertain Data. In *SIAM Data Mining Conference*, 2008.

5. C. C. Aggarwal and P. S. Yu. A Framework for Clustering Uncertain Data Streams. In *ICDE Conference*, 2008.

6. C. C. Aggarwal. On Density Based Transforms for Uncertain Data Mining. In *ICDE Conference*, 2007.

7. http://en.wikipedia.org/wiki/Cluster_analysis

8. M. Hua, Y .Tao, J .Pei and X. Lin. Mining Uncertain and Probabilistic Data: Problems, Challenges, Methods and Applications. In *Proceedings of the 14th ACM SIGKDD International Conference on*

Knowledge Discovery and Data Mining (KDD 2008), Las Vegas, NV, USA, August 24-27 2008.

9. http://en.wikipedia.org/wiki/Bayes'_theorem

10. J. Ren, S. D. Lee, X. Chen, B. Kao, R. Cheng and D. Cheung. Naive bayes classification of uncertain data. In *ICDM*, pages 944-949, 2009.

11. R. Cheng, D. Kalashnikov, S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of the ACM SIGMOD*, pages 551-562, 2003.

12. S. Singh, C. Mayfield, S. Prabhakar, R. Shah, S. Hambrush. Indexing categorical data with uncertainty. In *Proceedings of ICDE 2007*, pages 616-625, 2007.

13. S. Tsang, B. Kao, K.Y. Yip, W.S. Ho, S.D. Lee. Decision trees for uncertain data. In *IEEE International conference on Data Engineering*, 2009

14. B. Qin, Y. Xia, F. Li. DTU: A decision tree for uncertain data. In *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, 2009.

15. JR. Quinlan. C4.5: Programs for Machine Learning, Morgan Kaufman Publishers, 1993.

16. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten. The WEKA Data Mining Software: An Update. In *SIGKDD Explorations*, Volume 11, Issue 1, 2009.

17. P.N. Tan, M. Steinbach, V. Kumar. Introduction to Data Mining. Addison Wesley, 2005.

18. E. Hullermeier, K. Brinker. Learning valued preference structures for solving classification problems. *Fuzzy Sets and Systems*, 2008.

19. E. Hullermeier, J. Huhn. FR3: A Fuzzy Rule Learner for Inducing Reliable Classifiers. *Fuzzy Systems, IEEE Transactions*, 2009.

20. W. Cohen. Fast effective rule induction. *Proceedings of the 12th International Conference on Machine Learning*, pages 115-123, 1995.

21. J. Furnkranz. Round robin classification. *Journal of Machine Learning Research*, pages 721-747, 2002.

22. M. Magnani, D. Montesi. Uncertainty in Decision Tree Classifiers. In *Proceedings of SUM 2010*, pages 250-263, 2010.

23. http://www.gnu.org/licenses/gpl.html

24. http://en.wikipedia.org/wiki/Bash_(Unix_shell)

Ringraziamenti

E così siamo giunti alla fine di questa esperienza universitaria, la laurea magistrale: è durata circa due anni ed è costata "qualche" sacrificio, ma mi ha ripagato con molte soddisfazioni.

Il primo e più sentito ringraziamento va sempre ai miei genitori, che hanno reso tutto questo possibile, sia "*sponsorizzando*" (anche questa volta) l'iscrizione che aiutandomi in ogni mia scelta, guidandomi sempre con consigli preziosissimi e senza mai imporsi sulle mie decisioni.

Ringrazio poi i miei zii, Laura e Luigi, e miei cugini, Danilo e Vanessa, per essere sempre stati presenti e per avermi sempre supportato e incoraggiato durante la durata di questo mio cammino.

Ringrazio anche Serena, la mia morosa: mi ha supportato (e soprattutto sopportato) durante questo percorso, in particolare nell'ultimo periodo di tesi, dove devo riconoscere che non sono sempre stato il massimo della simpatia :) . Grazie infinitamente per il sostegno, l'aiuto e la pazienza nello starmi vicino, per esserci sempre stata.

Non posso dimenticare il fantastico gruppo che abbiamo formato in università: Cruz, Bedo, Uovo, Mandro, Rappo e Diego, con cui ho condiviso ansie da appelli, risate, pranzi in giro per pizzerie e mense, momenti di svago e di studio e mangiate serali a base di tigelle e crescentine davvero niente male! Un particolare ringraziamento va a Cata e Squera, con i quali ho pre-

parato diversi progetti e quasi tutti gli esami, condividendo ansie da consegna e soddisfazioni nel vederci riconosciuto positivamente il lavoro svolto.

Ringrazio anche i miei compagni delle superiori, Diom, Fabio, Gabe, Rich e Simon, coi quali ancora esco e mi diverto, per tutte le volte che mi hanno fatto ridere e mi hanno distratto dagli impegni universitari, facendomi un po' rilassare.

Un grandissimo grazie anche a tutti i compagni dei calcetti vari, in particolare a quelli dell'Orange Bud, per le soddisfazioni che ci siamo tolti e le risate fatte.

Voglio ringraziare anche tutto il Piano B, con cui abbiamo organizzato meravigliose serate: Ciccio, Gigi, Lucia e Vale. Grazie per avermi fatto divertire davvero tanto e per avermi distratto un po' dalle ansie universitarie.

Un sentito ringraziamento va al Prof. Danilo Montesi e in particolare al Dott. Matteo Magnani, il quale mi ha guidato con pazienza, gentilezza e professionalità alla realizzazione di questo elaborato, organizzando ricevimenti in tutti gli orari possibili e rispondendo alle mail sempre molto celermente. Se non fosse stato per lui, la strada percorsa per questo elaborato sarebbe stata molto più in salita.

Un immenso ringraziamento va a Carlos, Romano e Tosci, tre insostituibili amici senza i quali non saprei davvero come fare. Grazie per le serate in giro per locali a divertirci e per i mille caffè pomeridiani (o forse nel mio caso ACE, visto che di caffè ne prendo pochi!), per le corse al talon e le chiacchierate su tutto, per il sostegno e l'aiuto nel momento del bisogno. Semplicemente, grazie per esserci sempre stati!

Un ultimo grazie di cuore a tutte le persone citate e anche a quelle non citate, che hanno comunque contribuito a farmi crescere e maturare e mi han sostenuto e fatto divertire in questi anni. Ciao ciao!