Il guadagno informativo negli alberi decisionali: un nuovo approccio

Sommario

Descrizione del problema

I classificatori sono algoritmi che rientrano sia nell'area di interesse del Data – Mining, sia in quella dell'intelligenza artificiale. Il loro scopo è quello di riuscire a classificare l'appartenenza di un oggetto ad una determinata classe (cliente affidabile o inaffidabile, unità amica o nemica, livello di rischio di insorgenza di una malattia ecc.) in base agli altri attributi dell'oggetto stesso. Nei classificatori è estremamente diffuso l'utilizzo di forme induttive di apprendimento: dall'osservazione di fatti e oggetti relativi all'ambiente circostante il sistema deve generalizzare i dati e le informazioni ricevute, ottenendo conoscenza che, auspicalmente, sia valida anche per casi non ancora osservati.

Una famiglia di classificatori è formata dagli alberi decisionali. Un albero decisionale è un piano strutturato ad albero che individua una successione di test sugli attributi noti (detti attributi predittori) al fine di predire un attributo di output (non noto). Creare un albero decisionale significa dunque stabilire quali test effettuare su quali attributi. Ad ogni passo della creazione dell'albero, come già detto, l'algoritmo deve scegliere su quale attributo eseguire lo split; ovviamente è opportuno che la scelta ricada sull'attributo che permette di differenziare meglio il risultato, ottenendo quindi dei nodi figli il più "puri" possibile. La letteratura propone uno svariato numero di indicatori possibili, uno dei più noti è il guadagno informativo, basato sull'entropia.

Il guadagno informativo è dato dalla differenza tra l'entropia del nodo prima della divisione e dalla somma delle etropie dei nodi risultanti dalla divisione, pesate in base al numero di oggetti che vi finirebbero. E' quindi opportuno usare per il partizionamento del nodo l'attributo che offre il maggior guadagno informativo.

I sistemi attuali, tuttavia, fanno un'implicita assunzione: che tutti i test sui vari attributi abbiano lo stesso costo, cosa non sempre vera in tutti i casi. Un esempio può aiutare a chiarire la questione.

Immaginiamo che sia stata scoperta una nuova malattia, per la quale non esistono procedure di diagnosi valide. I pazienti vengono sottoposti a vari esami; solo alcuni pazienti presentano la malattia, mentre altri sono sani. Abbiamo quindi un database contenente i risultati degli esami svolti che possiamo usare come training set da dare in pasto ad un classificatore, in modo da scoprire quale sequenza di esami effettuare per arrivare ad una diagnosi corretta.

idPaziente	Pressione Arteriosa	Presenza proteina XK59 nel sangue	Gastroscopia	Elettrocardiogramma alterato	Malato?
1	150	Pos	Pos	Neg	SI
2	130	Neg	Neg	Pos	No
3	142	Pos	Neg	Pos	No
4	115	Neg	Neg	Neg	No
5	128	Pos	Pos	Neg	Si
…	…	…	…	…	…

Supponiamo che al primo passo dell'algoritmo i due attributi che offrono il maggiore guadagno informativo siano Gastroscopia e Presenza proteina XK59 nel sangue e che il guadagno informativo di questi due attributi sia quasi identico, ma leggermente superiore per la gastroscopia. Un albero decisionale classico suggerirebbe quindi che il primo passo da fare verso una corretta diagnosi della malattia sia eseguire una gastroscopia a tutta la popolazione, mentre il buon senso impone di preferire in prima battuta le analisi del sangue, molto meno costose ed invasive. La causa di questo errore è legata al fatto che gli algoritmi esistenti non sono in grado di pesare i guadagni informativi sulla base di altre indicazioni dovute al costo degli attributi in esame.

Il guadagno informativo di Nanni

Ipotizzando che sia possibile esprimere in maniera numerica i costi di reperimento degli attributi risulta logico preferire la migliore combinazione tra il costo necessario per ottenere un attributo e il guadagno informativo da esso promesso. Il guadagno informativo di Nanni formalizza proprio questo aspetto: esso è dato dal prodotto del guadagno informativo calcolato in maniera classica con un peso (compreso tra zero e uno) definito dall'utente. Il guadagno informativo legato ad un attributo estremamente costoso sarà, quindi, penalizzato da un valore del peso minore rispetto a quello legato ad un attributo più economico. Ciò permette di superare il problema dei classificatori attuali evidenziato con l'esempio precedente.

Per verificare l'efficacia di questa nuova visione è stato prodotto un classificatore che verrà integrato in Weka.

Il software Weka

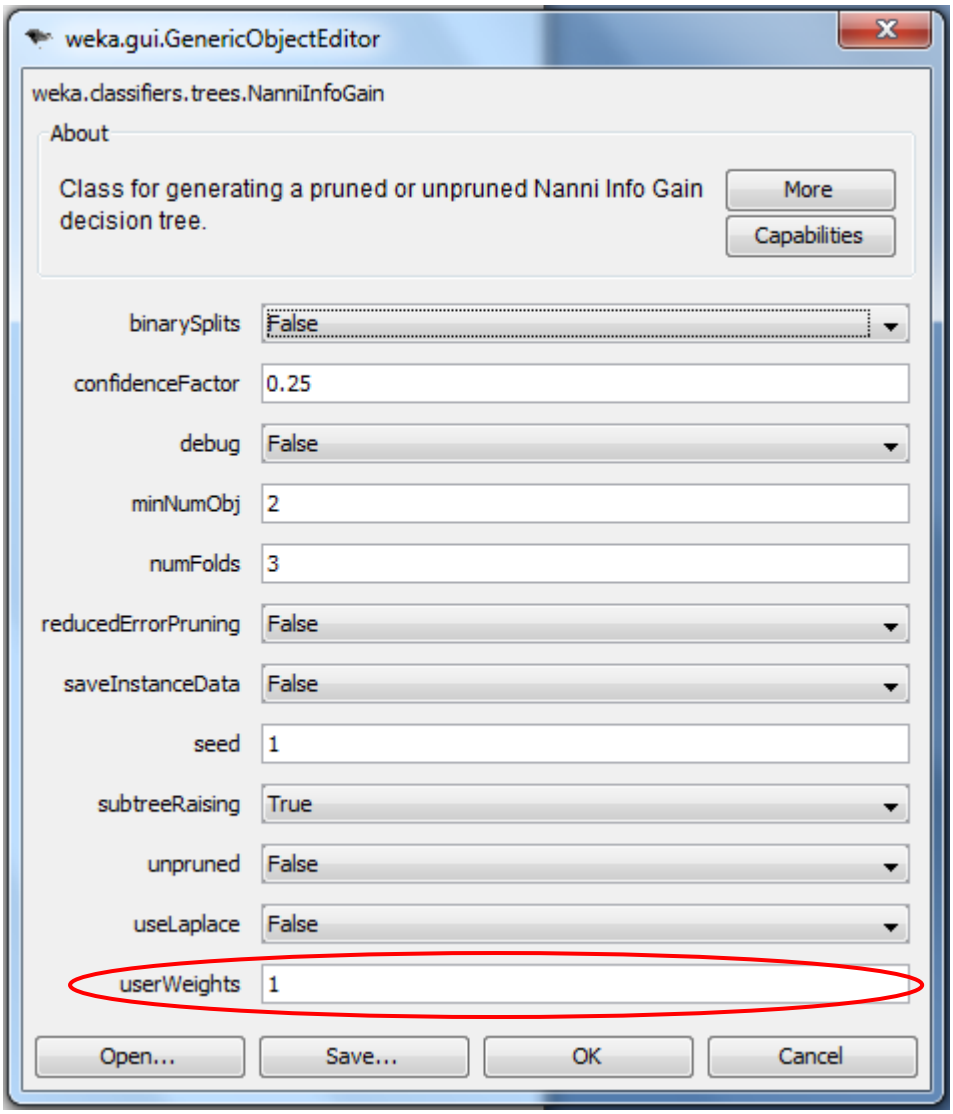
Cos'è Weka

Weka (Waikato Environment for Knowledge Analysis) è un software per l'analisi dei dati sviluppato dall'università di Waikato in Nuova Zelanda, è scritto in Java e rilasciato sotto licenza GNU. Il software permette con molta semplicità di applicare dei metodi di apprendimento automatici ad un set di dati, ed analizzarne il risultato. Grazie a questa semplicità di utilizzo ed al fatto di essere open source Weka è diventato un programma molto diffuso in tutti i corsi che affrontano tematiche di machine learning.

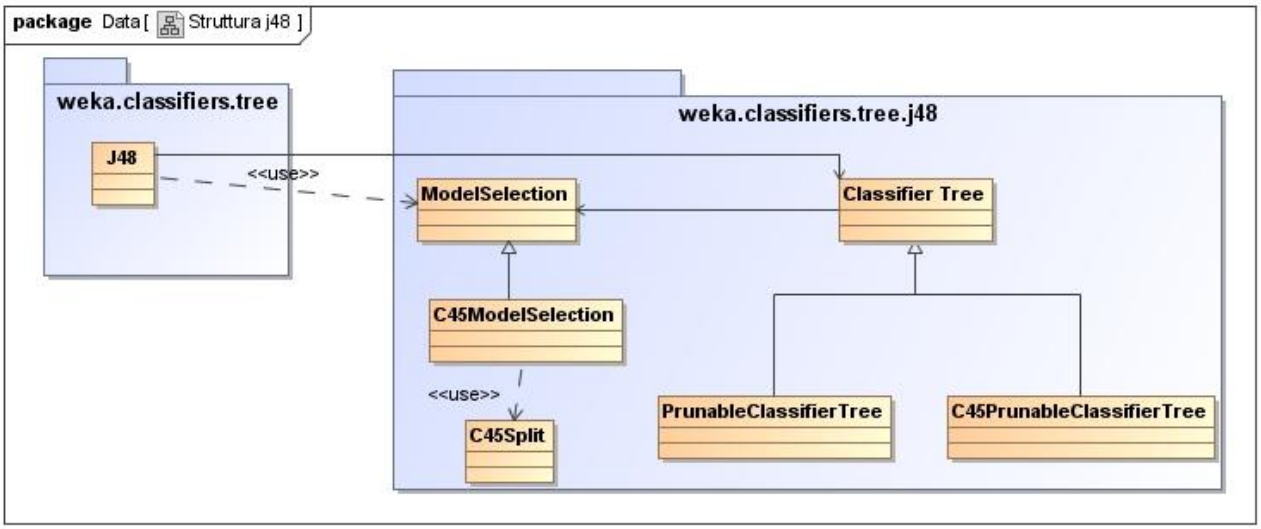
Il guadagno Informativo di Nanni in Weka

Si è sviluppato L'algoritmo come estensione di J48 la versione Java di C4.5, uno degli algoritmi basati sul guadagno informativo più famoso. Di conseguenza è in grado di predire attributi categorici basandosi su istanze che presentano sia attributi categorici che numerici tollerando la presenza di valori mancanti.

L'algoritmo presenta, inoltre, tutte le opzioni fornite da J48, con l'aggiunta di una casella dove potere indicare i pesi relativi agli attributi. I pesi devono essere compresi tra 0 ed 1. Se viene indicato un numero di pesi inferiore a quello degli attributi, viene attribuito un peso pari ad 1 al guadagno informativo degli attributi con peso non specificato. Se, al contrario, viene specificato un numero di pesi superiore a quello degli attributi, i pesi eccedenti non vengono cosiderati

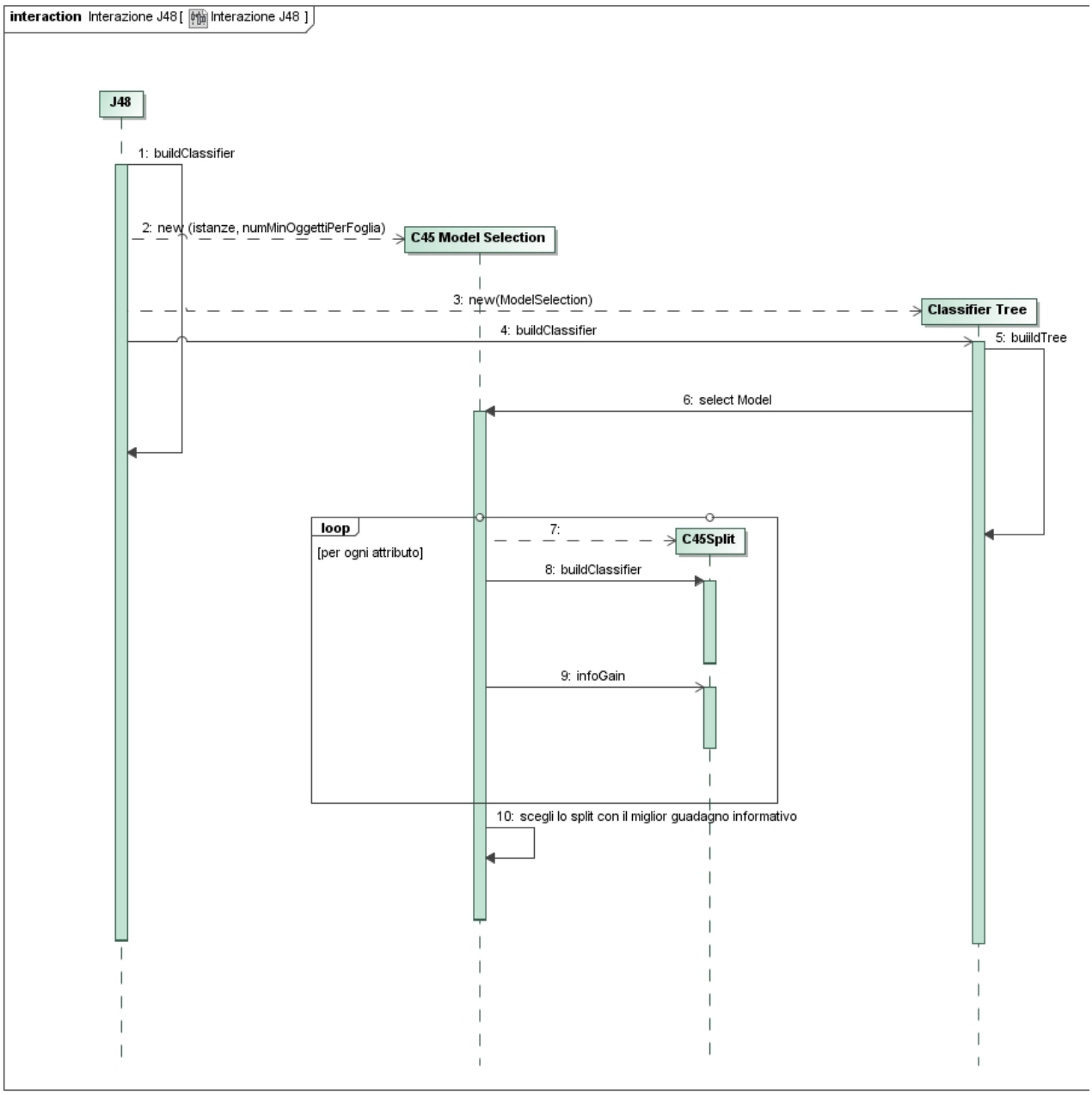


Le classi di Weka coinvolte nella determinare il migliore split da eseguire secondo la logica di C4.5 sono mostrate nella seguente figura



La classe principale è J48 che rappresenta l'algoritmo di classificazione. I componenti da essa usati sono contenuti nel package weka.classifiers.tree.j48. In particolare J48 ha un riferimento ad un Classifier Tree, il quale è la struttura dati ad albero usata dal classificatore. A seconda delle opzioni impostate può essere o un PrunableClassifierTree o un C45PrunableClassifierTree. Ogni classifier Tree ha un riferimento ad un Model Selection, inizializzato da J48 alla sottoclasse C45ModelSelection. La classe C45ModelSelection usa una serie di C45Split (uno per ogni attributo) per determinare il miglior split possibile. E' la classe C45Split che effettivamente provvede a calcolare il guadagno informativo relativo ad un attributo.

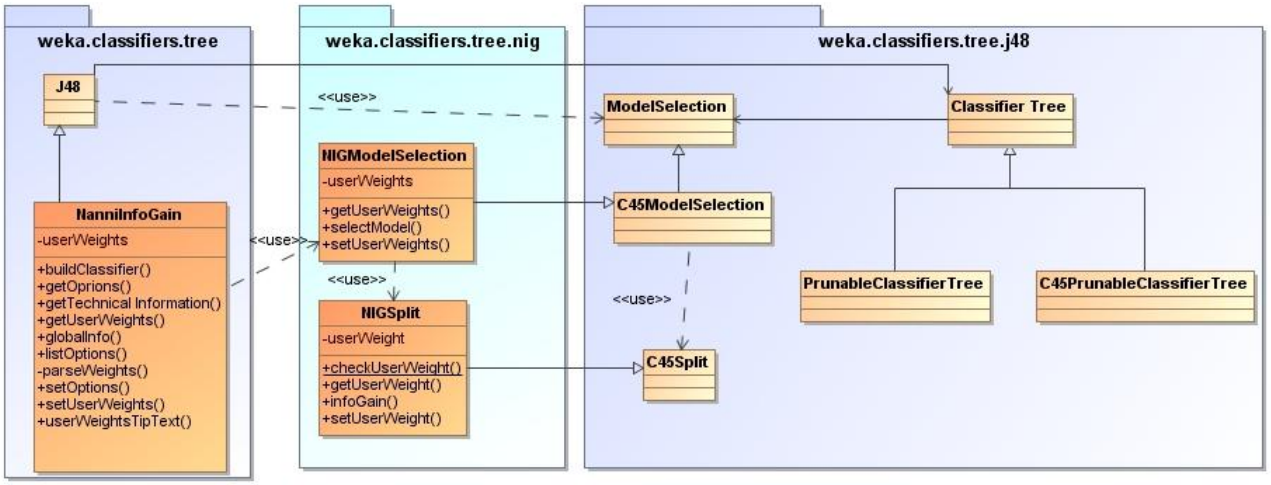
Il diagramma di sequenza sottostante riassume l'interazione tra i vari componenti. Per una corretta interpretazione del diagramma occorre precisare che questo non è in UML stretto, bensì ad un più alto livello di astrazione.



Nello specifico per determinare il miglior split il metodo buildClassifier della classe J48 crea un un c45ModelSelection altrimenti. Al modelSelection vengono passate le istanze da classificare ed il numero minimo di oggetti che devono essere presenti in una singola foglia. Dopodichè se l'opzione di Reduced Error Pruning l'albero di classificazione viene inizializzato a un C45Prunable Classifier tree, altrimenti a un PrunableClassifierTree. In entrambi i casi agli alberi viene passato come parametro il ModelSelection creato precedentemente. Dopodiché viene invocato il metodo buildClassifier sull'albero. Questo metodo invoca il metodo buildTree della superclasse ClassifierTree. Questo metodo decide prima su quale attributo fare lo split, poi lo esegue.

Per il primo compito viene usato il metodo selectModel del ModelSelection. Questo metodo è il centro della selezione dell'attributo in base a quale fare lo split; crea un vettore di C4.5Split, uno per ogni attributo. Poi, invoca per ogni attributo il metodo BuildClassifier, che crea uno split, il risultato viene valutato sia in termine di validità sia in termine di guadagno informativo. Infine si verifica se è stato trovato uno split valido e se ne decide il migliore.

Il codice di Weka è stato modificato introducendo le classi descritte dal seguente diagramma Uml:



La classe NIGSplit estende C45Split. Questa classe ha il compito di calcolare lo split ed il relativo guadagno informativo di Nanni per un attributo.

- Ha un campo per il peso definito dall'utente per l'attributo in esame, con relativo getter e setter
- Fornisce anche un metodo per verificare che il peso sia compreso tra zero ed uno (checkUserWeight)
- Ridefinisce il metodo infoGain in modo da ritornare il guadagno informativo fornito dalla classe madre moltiplicato per il peso specificato dall'utente.

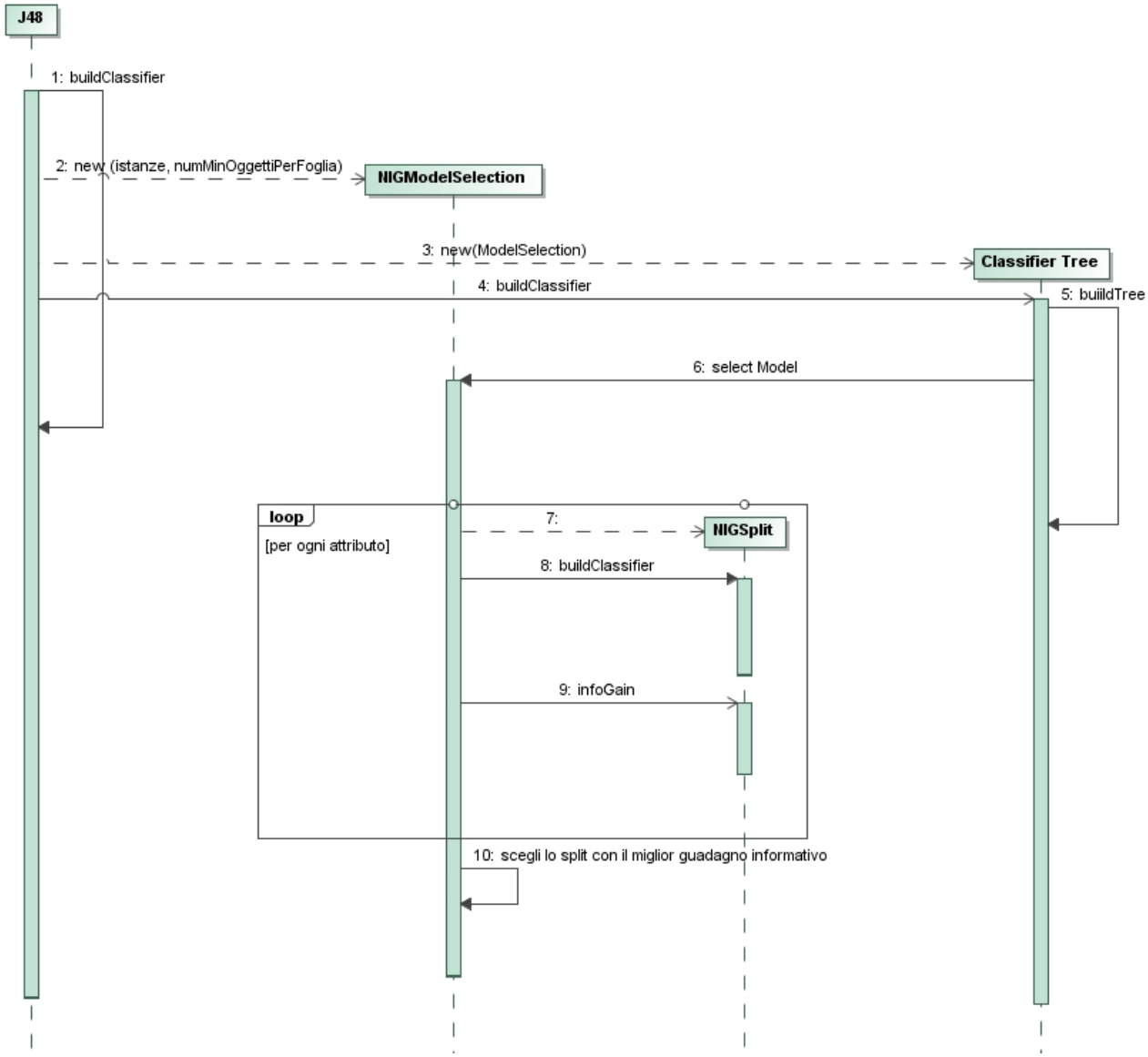
La classe NIGModelSelection estende C45ModelSelection. E' questa classe che ha il compito di determinare il miglior split tra quelli disponibili in base al guadagno informativo di Nanni.

- Ha un campo userWeights contenente i pesi dei vari attributi, con relativi getter e setter
- Ridefinisce il metodo selectModel in modo da creare tanti NIGSplit quanti sono gli attributi e scegliere il miglior split usando il guadagno informativo fornito dal metodo infoGain degli split.

La classe NanniInfoGain rappresenta il classificatore vero e proprio

- Ha un campo userWeights dove tiene memorizzati i pesi specificati dall'utente
- Ridefinisce il metodo buildClassifier usando non più un C45ModelSelection, ma un NIGModelSelection, al quale passa i pesi definisti dall'utente
- I metodi getUserWeights, setUserWeights userWeights TipText sono usati dall'interfaccia grafica di Weka per permettere all'utente di specificare i pesi nella finestra delle opzioni
- Ridefinisce i metodi getOptions/setOptions, listOption per far sì che il classificatore sia utilizzabile anche attraverso l'interfaccia a linea di comando.
- Il metodo parseWeights consente di trasformare la stringa di pesi inserita dall'utente in un vettore numerico, inoltre controlla anche la correttezza dei valori immessi dall'utente

Il diagramma di sequenza relativo alla scelta dell'attributo in base al quale eseguire lo split è analogo a quello presentato per descrivere il funzionamento di J48

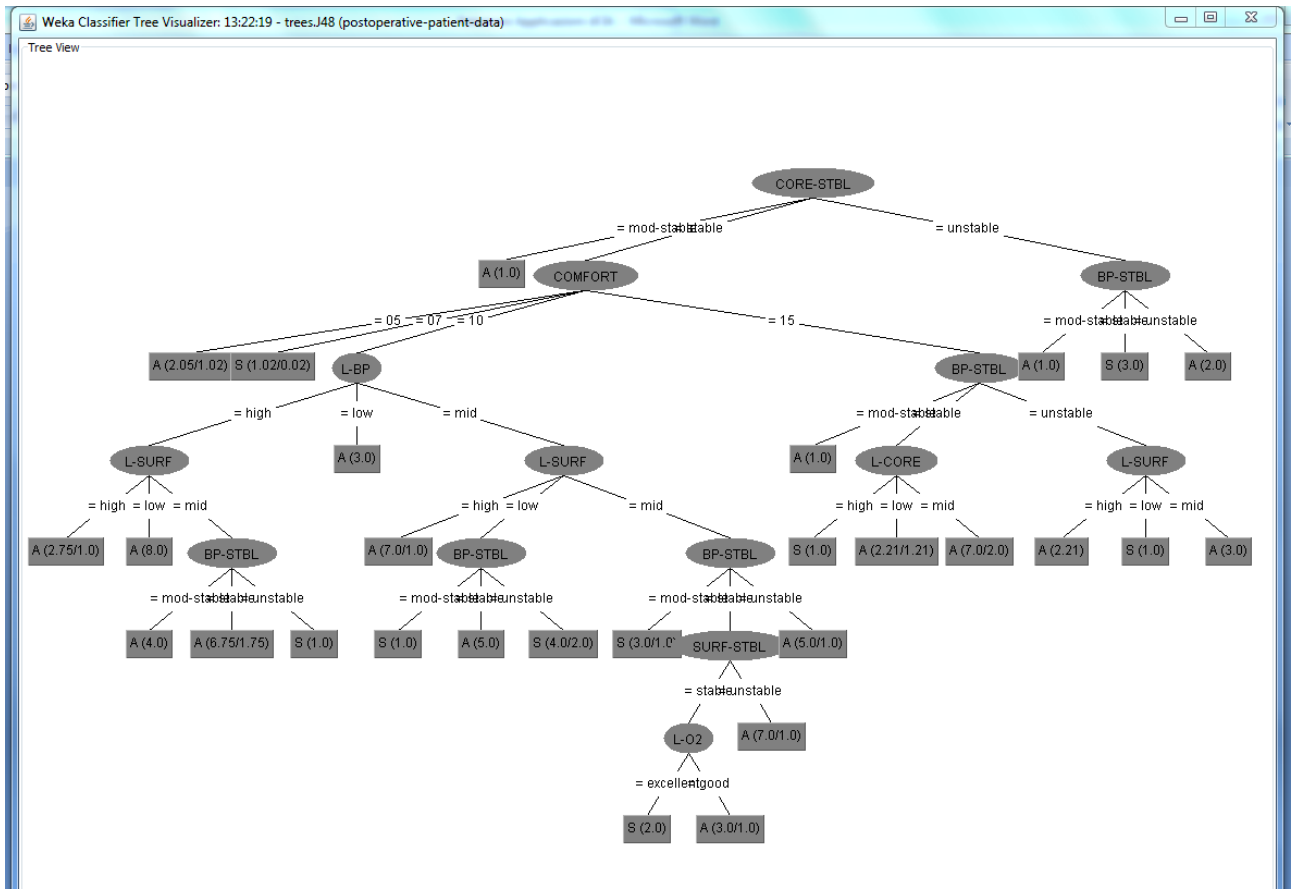


Risultati Sperimentali

Il software prodotto è stato testato su un sistema dotato di Intel Core 2 6600 a 2.40 GHz, 2 GB di ram, con Windows 7 come SO.

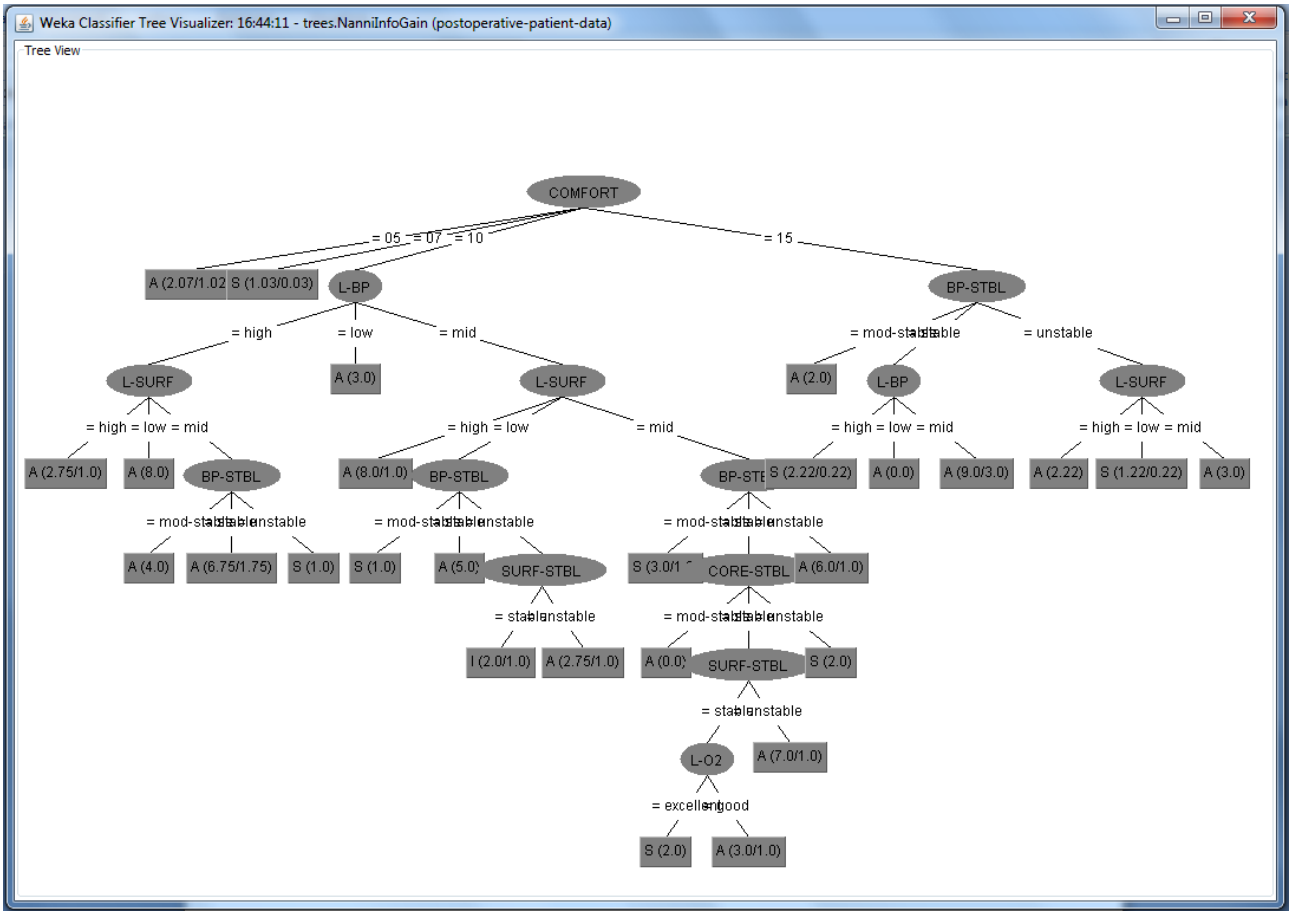
Efficacia

Per questo test si è usato il dataset Post Operative Patient (http://archive.ics.uci.edu/ml/datasets/Post-Operative+Patient), il cui scopo è determinare dove mandare un paziente(dimetterlo S, corsia generica A, oppure in rianimazione I) dopo un intervento chirurgico in base ad una serie di parametri medici. L'albero (senza pruning) generato da J4.8 è il seguente:



Come si vede il primo split è eseguito in base all'attributo CORE STBL, attributo legato alla stabilità della temperatura interna del paziente.

Per testare l'efficacia del classificatore creato si è abbassato il peso dell'attributo in questione fino a 0.5, impostando il peso degli altri attributi a 1. L'albero risultante è:



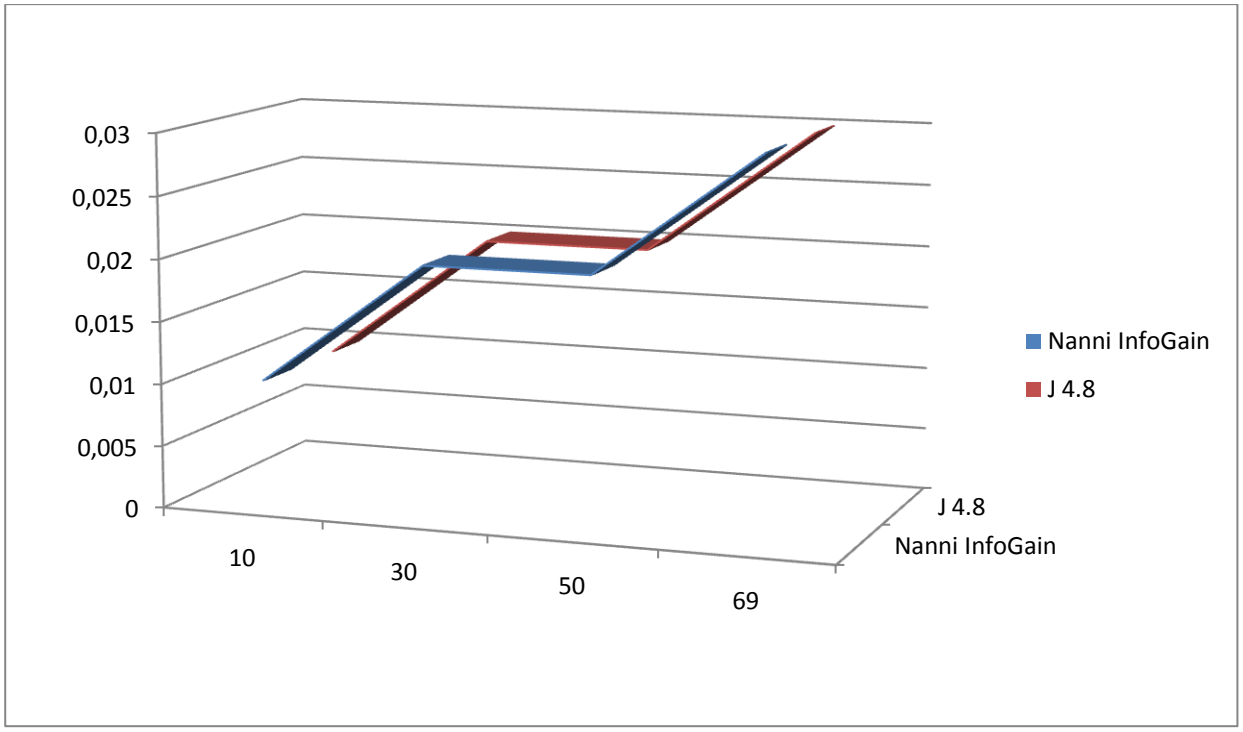
Come si può notare nel primo split l'attributo CORE-STB risulta penalizzato da un peso inferiore, tanto che gli è preferito l'attributo COMFORT , il quale indica quanto si sente bene il paziente.

Accuratezza

L'accuratezza del classificatore dipende molto da quanto vengono penalizzati gli attributi che più sono utili a determinare la classe risultato. Più questi attributi sono penalizzati da un peso basso, peggiore sarà l'accuratezza del classificatore. Questa misura, come facile intuire, ha come upper bound quella di C45, corrispondente all'avere pesi uguali per tutti gli attributi. Nell'esempio precedente, usando come test-set il 20% del dataset, l'algoritmo classifica correttamente 11 istanze su 18, contro le 12 di C4.5.

Efficienza

Per mostrare le performance dell'algoritmo in base al variare del numero di attributi lo si confronta con J4.8 nella classificazione dell'attributo class del dataset Auditology (http://archive.ics.uci.edu/ml/datasets/Audiology+%28Standardized%29) . In particolare per enfatizzare eventuali differenze e rendere la misura meno sensibile a rumore si è impostata per ogni prova la cross-validation con 20 folds, in modo che ogni esecuzione corrisponda, in realtà a 20 esecuzioni su dati diversi.



Come prevedibile le differenze nei tempi di esecuzione non sono apprezzabili, in quanto l guadagno informativo di Nanni introduce solo una moltiplicazione per attributo.

Sviluppi futuri

La prima direzione di sviluppo può riguardare la formulazione dei pesi: attualmente sono dei valori compresi tra zero ed uno e restano invariati per tutta la costruzione dell'albero. In futuro si può prevedere di accettare anche funzioni il cui valore può cambiare, ad esempio, in base alla profondità raggiunta o al numero di istanze presenti nel nodo.

Si può prevedere ad un sistema che accetti non solo preferenze numeriche (quantitative), ma anche preferenze qualitative e condizionali.

Si può anche pensare di aiutare l'utente nel risolvere il tradeoff tra perdita di accuratezza e correttezza nella definizione dei pesi creando un supporto che mostri per ogni nodo intermedio dell'albero i guadagni informativi degli attributi.