**Tree Induction - Decision Tree**

Tree induction è l'algoritmo per costruire un albero di decisione per problemi di classificazione o regressione. È una procedura ricorsiva, divide et conquer che utilizza tre procedure:

* Homogeneous(D), verifica se tutte le istanze appartengono alla stessa classe, allora il nodo è puro.
* Label(D), return l'etichetta più appropriata per i dati del nodo.
* Best Split(D,F), return l'insieme di letterali da mettere come root dell'albero.

Il goal è misurare la purezza dei nodi, creando un albero che massimizzi la separazione tra classi e unendo gli output delle suddivisioni.

Per la scelta del letterale per lo split possono essere usati:

* Entropia: misura l'incertezza associata a una variabile discreta. Sono il #bit necessari per comunicare la classe di un esempio estratto a caso. H=−∑P(Cj )⋅log2 P(Cj ). Con P(Cj) la probabilità della classe Cj. E' incluso tra (0,1). H=0, non vi è impurità, tutti gli esempi appartengono alla stessa classe, H=1, dataset bilanciato, stessa probabilità per la classe + che per quella -.
* Information gain: Differenza dell'entropia del dataset prima e dopo uno split. IG=H(D)−Hx (D). Se venissero usati attributi con molti valori diversi, si avrebbe |Di|=1, entropia di ogni sottoinsieme nulla, e quindi guadagno massimo. Verrebbe quindi scelto sempre quel attributo. Quello che si fa, è o penalizzare gli attributi non utili, utilizzando il Gain Ratio, o eliminare dal dataset tutti quegli attributi.
* Gini index, che misura l'impurità per k classi. Anche questo, 0 dataset puro, 1 dataset impuro.

Se un attributo è continuo, si sceglie un test del tipo x>c. Si ordinano gli esempi secondo quell attributo, si individuano i punti in cui cambia la classificazione, si generano le soglie candidate per ogni punto, e si sceglie quella che produce il maggior information gain.

Se un attributo è non noto, C4.5 assegna dei pesi agli esempi distribuendoli lungo i rami a seconda della probabilità che appartenga a ognuno.

L’overfitting può verificarsi quando il dataset contiene errori, o è piccolo. Per ridurlo, si possono usare tecniche di pruning.

* Pre pruning: si interrompe la crescita prima che classifichi correttamente,
* Post pruning, prima si fa crescere l'albero poi si tagliano rami/nodi. Questo può essere di due tipi:
  + Reduce Error Tagliare un nodo = sostituirlo con una foglia con la classe di maggioranza degli esempi che scendono lungo quello e si eliminano quelli che portano a un aumento della acc.
  + Rule post L'albero si converte in regole che sono percorsi root-leaf e si semplificano.

**Metriche di Performance**

Per valutare le performance di un modello possono essere usate:

* Accuratezza: Proporzione di esempi classificati bene. (Tp+Tn)/Tot
* TPR: Frazione di esempi predetti positivi correttamente su tutti gli esempi che sono effettivamente positivi. Tp/(Tp+Fn) (1 colonna)
* FPR: (2 colonna)
* Precision: Frazione di esempi predetti positivi correttamente su tutti i predetti positivi. Tp/(Tp+Fp) (1 riga)
* Recall: TPR.
* F1S: 2PR/(P+R).

Si possono poi usare due curve:

* Roc Space: Traccia il TPR in funzione del FPR.
  + Zona ottimale: (0,1) AUC=1
  + Zona buona: Sopra la diagonale. AUC=(½, 1)
  + Zona random: Diagonale. AUC=½
  + Zona cattiva: Sotto la diagonale.
  + Zona pessima: Inverte la classif. AUC=0

Per valutare le prestazioni, non con una curva, ma con un valore, può essere usato AUC Roc. È l’area sotto la curva.

* Curve PR: Traccia P in funzione di R. Può essere usata per rilevare le differenze che non sono visibili con la curva ROC.
  + Zona ottimale: (1,1), Alto-destra. AUC=1
  + Zona random AUC=½

Per la regressione, si possono usare:

* MSE: Errore quadratico medio. Misura quanto i dati sono concentrati intorno alla linea di migliore andamento. Errori maggiori sono considerati di più.
* RMSE: Deviazione standard.
* MAE:Considera la differenza in valore assoluto tra originale e previsto.

Tutti sempre non negativi, più sono piccoli, meglio è. Preferito è RMSE, perché riporta le unità di misura corrette, non al quadrato come MSE.

**KNN**

KNN è un algoritmo di apprendimento supervisionato, basato sulle istanze. L’apprendimento consiste nella memorizzazione degli esempi, mentre la classificazione consiste nel reperimento degli esempi simili e classificazione sulla base di quelli. KNN classifica gli esempi basandosi sui k più vicini. Si basa sull’idea che esempi simili siano vicini nel piano RN. Per reperire gli esempi simili si usa la distanza. Questa può essere: Euclidea, di Manhattan, di Chebyshev o di Minkowski. La scelta di K può portare a overfitting se troppo piccolo, considerando pochi vicini, oppure a underfitting se troppo grande, perché si considererebbe un’area molto grande, e si generalizza la previsione.

Alcune estensioni di KNN sono:

* Ponderazione: Assegnare un peso agli esempi e quindi i vicini più vicini sono più influenti. Infatti w=1/d+e, Pesi inversamente proporzionali alla distanza.
* A distanza ponderata: Per evitare che vicini lontani influenzino quanto uno vicino.
* Feature scaling: Si Normalizzano le feature per evitare che attributi con valori grandi dominino il calcolo.

KNN può andare incontro a Curse of Dimensionality. Perché vengono considerate tutte le feature. Es: 20 attr. con solo 2 rilevanti. Se due esempi valori simili per quei due, possono risultare lontani. Per evitare il problema, si possono pesare diversamente gli attributi, variando la lunghezza degli assi a seconda della rilevanza dell’attr. o si annullano alcuni pesi, eliminando alcune feature, o ridurre la dimensionalità comprimendo lo spazio delle feature.

**Random-Forest**

Le RF sono un algoritmo di Model Ensemble. Combina più alberi decisionali che sono costruiti a partire da diversi sottocampioni del dataset originale. Ogni albero produce una predizione, queste vengono combinate o tramite voting (classificazione) o averaging (regressione). Vengono creati i bootstrap sample tramite il bagging. Vengono selezionate f feature casuali dall’insieme di tutte le feature F (F1/2 classif. ⅓F regres.). Una volta creati gli alberi, si combinano i risultati. Sono robusti al rumore e riducono il rischio di overfitting rispetto a un singolo albero, ma hanno una interpretabilità ridotta.

**Bagging**

È una tecnica di model ensemble che costruisce modelli predittivi multipli con sottocampioni casuali, combinando i risultati. I sottocampioni possono essere creati con il Sample with replacement, dove ogni item può essere ripescato in altri sample, ogni sample ha la stessa DIM del dataset e ognuno lascia fuori circa ⅓ dei data point e questi usati sui modelli che non li hanno visti, possono essere un test set. Ogni modello viene addestrato da un campione diverso e poi si combinano i risultati o per voting o per averaging. Bagging, riduce la varianza, ed è robusto agli outlier e dataset rumorosi.

**Boosting**

È una tecnica di model ensemble che costruisce modelli predittivi deboli in sequenza, facendo in modo che ognuno si concentri sugli errori commessi dal precedente. È iterativo. Il modello successivo deve correggere gli errori del precedente e gli esempi più difficili verranno pesati di più. Al termine di una iterazione, può succedere che esempi difficili diventino più difficili, come quelli facili, più facili. Tutte le combinazioni sono possibili. AdaBoost, aggiorna i pesi dopo ogni iter, e combina i risultati con una somma ponderata. Boosting riduce il bias.

**Clustering - Distance based - K means - Fuzzy C means - Gerarchico**

Distance based clustering è una tecnica che suddivide un dataset in gruppi basandosi sulla distanza tra i punti nello spazio delle feature. Vuole minimizzare la distanza intra cluster e massimizzare quella inter cluster. Possono essere usate diverse distanze (euclidea) e ogni cluster contiene i data point che sono più vicini tra loro.

K means è un algoritmo di distance based clustering. La funzione obiettivo vuole minimizzare la somma delle distanze quadratiche tra ogni punto e il centro del cluster. Obj = argmin ∑∑ || x - ci ||2. La scelta iniziale dei centroidi può avvenire in diversi modi:

* Le prime k istanze,
* Etichettare gli esempi con numeri progressivi e selezionare valori precisi: m/k, 2m/k,
* Scelta random,
* Generare k punti scegliendo random i valori di ogni coor nell'intervallo di coordinate,
* Generare una partizione del dataset in k sottoinsiemi e considerare i loro centri.

Fuzzy C means è un clustering di tipo overlapping, in cui un dato può appartenere a più cluster, ma con grado di appartenenza diverso. Si utilizza la matrice di appartenenza U. Ogni elemento uij indica il grado di appartenenza di xi al cluster j. L’algoritmo oltre a restituire U, anche l’elenco dei cluster. La funzione obiettivo è argmin ∑∑ uijm|| x - ci ||2. L’algoritmo parte inizializzando la matrice U. Calcola il vettore dei centri, Aggiorna la matrice, Se la differenza con il passo precedente è minore di una soglia, stop, altrimenti torna al passo 2.

Il clustering gerarchico identifica gruppi e produce una rappresentazione ad albero. Può essere di due tipi:

* Bottom Up: Si parte da un oggetto un cluster, e si arriva a tutti gli elementi in un unico cluster.
* Top Down: Si parte dalla root che è un unico cluster, e si specializza in un cluster per ogni oggetto.

Non si vuole ottenere un unico cluster, ma per ottenerne k, si taglia l’albero, i rami dell’albero, al livello interessato. Vi sono tre metriche per misurare la distanza:

* Single, considera la distanza più breve tra due membri
* Complete, considera la distanza più grande tra due membri
* Average Linkage: considera la media tra due membri

Si ottiene un dendrogramma, con altezza proporzionale alla dissimilarità dei figli.

**Data mining - KDD**

KDD, Knowledge Discovery in database, è il processo non banale di identificazione di modelli validi, nuovi, potenzialmente utili, comprensibili nei dati.

* Processo non banale, comprende delle fasi come ricerca, inferenza, non il calcolo di quantità predefinite,
* Validi, i modelli devono essere applicabili a nuovi dati,
* Nuovi, i modelli non devono essere conosciuti prima,
* Pot. Utili, portare ad azioni utili,
* Comprensibili, compresi da esseri umani.

KDD è iterativo e interattivo, Il processo prevede la comprensione del dominio applicativo, il consolidamento dei dati, selezione e pre-elaborazione. Queste sono le attività preliminari prima di applicare l'algoritmo, unire i dati provenienti da diversi supporti, selezionare campioni, selezionare solo le feature utili. Poi vi è la trasformazione dei dati, la conversione in un formato usabile. Il data mining vero e proprio, quindi l’applicazione dell’algortimi. Arrivando all’interpretazione, analizzando i pattern ottenuti per garantire che siano validi.

Una regola associativa (A→B) descrive una correlazione tra eventi. Mostra relazioni frequenti tra elementi del dataset. Ad esempio: Se vengono comprati latte e pane, è probabile che si compri anche il burro: Latte, Pane → Burro.

La scoperta delle regole associative vuole identificare quelle relazioni che soddisfano i vincoli di supporto e confidenza minimi. Dove con supporto(x) = #transazioni con x/#totale, confidenza(x→y) = supp(x→y)/supp(x). Può essere usato l’algoritmo APriori. Prevede prima l’individuazione dei pattern frequenti, identificare quindi gli itemset che appaiono frequentemente rispettando il minsup. Poi vi è la generalizzazione delle regole, in cui solo quelle con confidenza maggiore di minconf vengono considerate valide. Apriori inizia dagli itemset con length 1. Calcola i supporti, Mantiene quelli con supp>min supp, e itera. Per ogni itemset, si trovano le possibili divisione A→B e si mantengono quelle con conf>minconf.