Alberi di Decisione per la Classificazione

Federico Màlato

April 5, 2018

1 Introduzione

Nell'ambito dell'apprendimento supervisionato, un assai utile strumento è rappresentato dagli alberi di decisione. Tali strutture dati sono formate sulla base di esempi (forniti dall'utente) composti da un insieme di valori, detti attributi. Per ciascun attributo viene specificato inoltre un dominio, ossia un set di valori che esso può assumere. Questi esempi vengono sfruttati da una funzione per generare un albero che, a sua volta, verrà usato per predire la classificazione di nuove tuple di dati, sulla base di ciò che è stato appreso durante la fase di addestramento.

2 Apprendimento

Dato un insieme di esempi detto *data set*, si effettua una suddivisione in due parti (spesso asimmetriche, solitamente secondo un criterio 75/25) chiamate *training set* (la parte comunemente più grande del data set) e *test set*.
Nella fase di apprendimento (anche nota come *fase di training* o *di addestramento*), un'apposita funzione si occupa di generare l'albero a partire dagli esempi presenti nel training set. La costruzione dell'albero si fonda sulla scelta di una suddivisione dei dati in base ad uno specifico attributo. Dopo averlo selezionato (seguendo una particolare euristica), si procede ad espandere l'albero di tanti rami (o *branches*) quanti sono i valori che tale attributo assume negli esempi. Ricorsivamente, viene poi scelto un altro attributo su cui effettuare un nuovo *split* dei dati.

2.1 Scelta dell'attributo

Per la scelta dell'attributo, come detto nel paragrafo precedente, c'è bisogno di un criterio. Sebbene ce ne siano molti, uno dei più utili risulta essere l'**Information Gain**, che misura il guadagno ottenuto nella creazione dell'albero in base ad una certa misura di impurità. A loro volta, le misure di impurità sono molteplici, sebbene alcune siano le principali e le più utilizzate:

- Entropia: l'entropia misura la quantità di disordine presente nei dati. Usare

l'information gain con questo tipo di impurità significa cercare l'attributo per cui l'entropia dei dati viene maggiormente ridotta;
- Indice di Gini: è un indice che misura la probabilità di una classificazione errata basandosi su un particolare attributo. In questo modo, l'attributo che ha indice di Gini minore (ossia quello che ha più probabilità di portare ad una classificazione corretta) viene scelto per effettuare lo split;
- Errore di classificazione: altra misura d'impurità che si focalizza sull'errore di classificazione che genera uno specifico attributo.

3 Codice usato per l'implementazione

```
def decisionTreeLearning(trainingSet, attributes, targetAttr, default, criterion='entropy'):
    # At first, the function makes some deep copies in order to avoid mess during the various recursions.
    # This copies may slow down the process a bit, but it's worth the effort.
    localTraining = copy.deepcopy(trainingSet)
    localAttr = copy.deepcopy(attributes)
    values = [record[targetAttr] for record in trainingSet]
    # Some default cases: if there's nothing to split, it just doesn't.
    if not localTraining or ((len(localAttr) - 1) <= 0):
        return default
    elif values.count(values[0]) == len(values):
        return values[0]
    else:
        maxGainAttr = chooseBestAttribute(localTraining, attributes, targetAttr, criterion)
        decTree = {maxGainAttr: {}}
        # Create a new decision tree/sub-node for each of the values in the
        # best attribute field.
        for value in getValues(trainingSet, maxGainAttr):
            # Create a subtree for the current value under the "best" field.
            subtree = decisionTreeLearning(getExamples(localTraining, maxGainAttr, value),
                                           [attr for attr in attributes if attr is not maxGainAttr],
                                           targetAttr, default, criterion)

            # Add the new subtree to the empty dictionary object in our new
            # tree/node we just created.
            decTree[maxGainAttr][value] = subtree
    return decTree
```

4 Classificazione

Per la classificazione, è stato usato un metodo di cross validation, ossia il metodo classico secondo cui il data set viene suddiviso in training set e test set. In particolare, è stata usata una tecnica di *k-fold cross validation*.

4.1 k-fold cross validation

La k-fold cross validation prevede di suddividere il data set come specificato all'inizio della sezione. La particolarità risiede nel fatto che questo processo viene eseguito per un numero arbitrario di iterazioni pari a k (solitamente, si usa $k = 5$ o $k = 10$) e, per ciascuna iterazione, il data set viene suddiviso in k parti uguali. Di queste parti una viene utilizzata come test set, mentre le altre $\frac{k-1}{k}$ vengono riaggregate a formare il training set con cui istruire l'albero. Ad ogni iterazione, inoltre, la k-esima parte usata come test set viene selezionata diversa dalla precedente, così da variare a sua volta il training set e produrre pertanto un test più generico possibile.

Per ogni classificazione effettuata correttamente, viene assegnato un punteggio positivo e, alla fine delle 5 iterazioni previste dalla 5-fold cross validation, viene calcolato lo score medio. Si ha uno score diverso per ciascun tipo di impurità utilizzata.

5 Codice per la cross validation

```
def kFoldCrossValidation(k, examples, target, attributes, default):
    # The data set is divided into five parts by the trainTestSplit() function. For every iteration,
    # a different part of the data set is chosen as the validation set, while the others are used as
    # the training for the tree. Every criterion is validated with the same train/valid split. For each
    # iteration, then, a new tree is generated and tested.

    # This first part just initialize the variables.
    score = 0.0
    giniScores = []
    entropyScores = []
    missclassScores = []
    criteria = ['gini', 'entropy', 'missclass']
    rand.shuffle(examples)
    for fold in range(k):
        print "####################"
        print "### iterazione " + str(fold+1) + " ###"
        print "####################"
        print
        trainSet, validSet = trainTestSplit(examples, (len(examples)/k)*fold, (len(examples)/k)*(fold+1))
        training = []
        for line in trainSet:
            training.append(dict(zip(attributes, [datum.strip() for datum in line])))
        for crit in criteria:
            print "Criterio: " + crit
            tree = dt.decisionTreeLearning(training, attributes, target, default, crit)
            classification = classify(tree, validSet, attributes, default)
            x = 0
            for el in validSet:
                if classification[x] == el[attributes.index(target)]:
                    score += 1.0
                x += 1
            print "Round score: " + str(score/(len(validSet)))
            if crit == 'gini':
                giniScores.append(score/len(validSet))
            elif crit == 'entropy':
                entropyScores.append(score/len(validSet))
            elif crit == 'missclass':
                missclassScores.append(score/len(validSet))
            score = 0.0
    return (sum(giniScores))/k, (sum(entropyScores))/k, (sum(missclassScores))/k
```

6 Testing

Per il testing sono stati selezionati tre data sets, diversi sia nel tipo di dato sia nella loro aggregazione, al fine di mostrare punti di forza e debolezze degli alberi di decisione:
- il **primo data set** è composto di dati molto differenti tra loro, in modo tale da rendere più difficile la classificazione. Lo score medio su questo data set si attesta intorno a 0.70-0.75, in cui nessuna delle impurità domina le altre;
- il **secondo data set** risulta più regolare, e rispecchia infatti i dati attesi dallo studio teorico: l'entropia domina l'indice di Gini e l'errore di classificazione. Quest'ultima misura d'impurità risulta essere la peggiore. In ogni caso, lo score medio si attesta intorno a 0.9-0.93 per l'entropia, 0.88 per l'indice di Gini e 0.85 per l'errore di classificazione;
- il **terzo data set** è di dimensioni più contenute rispetto agli altri e presenta pochi attributi: in questo caso la classificazione risulta più semplice, ed infatti

gli score si aggirano intorno allo 0.95-0.98 per tutte e tre le misure d'impurità, con dominanza dell'entropia.

In generale, dunque, si ha che l'information gain abbinato all'entropia come misura di impurità registra lo score migliore, seguito dall'indice di Gini. Terzo posto per il misclassification error. Per dimostrare poi l'efficacia dell'algoritmo di generazione dell'albero, viene stampato l'albero di decisione corrispondente al notissimo data set "playtennis". L'unico scopo di tale data set è appunto quello di fornire una visione dell'albero di decisione, essendo troppo piccolo per qualsivoglia tipo di test.