Alberi di decisione e relativi indici di impurità

Stiven Metaj

9 febbraio 2018

Sommario

Con questo elaborato si intende implementare e differenziare i 3 indici di impurità usati negli alberi di decisione. Per il suddetto confronto viene utilizzata una *5-fold cross validation*. I risultati ottenuti seguono quelli teorici/empirici.

1 Introduzione Teorica

Nell'ambito del *Machine Learning* uno dei metodi di apprendimento conosciuti maggiormente è quello degli *alberi di decisione*. Esso consiste nel costruire un albero di decisione a partire da un insieme di dati iniziali (dataset) usato poi come strumento di predizione per dati futuri.

Questo tipo di struttura però presenta un problema, infatti essa aumenta esponenzialmente con l'aumento degli attributi e dei relativi possibili valori; si cerca allora un modo per minimizzare il numero di nodi creati durante la costruzione dell'albero, in modo da avere più velocità e minor spazio occupato. Si introducono allora i concetti di *costo* e di *information gain*:

- il **costo** è una misura dell'impurità del dataset (rispetto all'attributo target, il quale rappresenta una certa classe che vogliamo studiare e predire)
- il **gain** invece è una misura che indica l'attributo migliore da cui continuare (o iniziare) la costruzione dell'albero. Esso, dato un dataset D ed un certo attributo A è definito come segue:

$$Gain(D, A) = Cost(D) \; - \sum_{v \in Values(A)} \frac{|D_v|}{|D|} Cost(D_v)$$

Dalla definizione quindi, il valore del gain cambia a seconda della funzione che calcola l'indice di impurità. E' proprio su questo aspetto che si incentra l'elaborato, infatti si vanno a definire i principali indici di impurità da confrontare: **misclassification**, **gini** ed **entropia**:

$$Misclassification(D) = 1 - \max\{p_k\}$$

$$Gini(D) = \sum_k \; p_k(1 - p_k)$$

$$Entropy(D) = -\sum_k \; p_k(\log_2 p_k)$$

2 Implementazione in python

2.1 Indici di Impurità ed Information Gain

Nel file *impurityIndexes.py* troviamo le funzioni che calcolano i costi e la funzione per calcolare il gain. Tutte fanno uso di una importante sotto funzione che ritorna un dizionario dove ad ogni possibile valore di un certo attributo all'interno del dataset è associato il numero di volte che esso compare. Questa informazione infatti è fondamentale per il calcolo degli indici.

```
def gain(data, attribute, targetAttribute, functionName):
    frequencyList = getValuesFrequency(data, attribute)
    secondTerm = 0.0
    for value in frequencyList.keys():
        tmp = functionName([i for i in data if i[attribute] == value],
                           targetAttribute)
        secondTerm = secondTerm + ((frequencyList[value] /
                                    sum(frequencyList.values())) * tmp)
    return functionName(data, targetAttribute) - secondTerm

def entropy(data, targetAttribute):
    frequencyList = getValuesFrequency(data, targetAttribute)
    entropy = 0.0
    for i in frequencyList.values():
        entropy = entropy + (-i / len(data)) * math.log(i / len(data), 2)
    return entropy

def misclassification(data, targetAttribute):
    frequencyList = getValuesFrequency(data, targetAttribute)
    maxValue = max(frequencyList.values())
    return 1.0 - maxValue/len(data)

def gini(data, targetAttribute):
    frequencyList = getValuesFrequency(data, targetAttribute)
    gini = 0.0
    for i in frequencyList.values():
        gini = gini + (i / len(data)) * (1 - (i / len(data)))
    return gini
```

2.2 Albero di Decisione

Nel file *decisionTree.py* troviamo le funzioni che permettono la costruzione dell'albero dati un dataset, la lista degli attributi e l'attributo target. E' la funzione *createDecisionTree()* che in realtà costruisce l'albero, mentre le altre funzioni presenti servono per calcolare qual'è l'attributo con maggior *information gain*, per trovare il valore del target che compare più volte e per costruire i dataset nelle chiamate ricorsive.

```
def createDecisionTree(data, attributes, targetAttribute, impurity, parentData):
    vals = [i[targetAttribute] for i in data]
    if not data:
        return pluralityValue(parentData, targetAttribute)
    elif (len(attributes) - 1) <= 0:
        return pluralityValue(data, targetAttribute)
```

```
    elif vals.count(vals[0]) == len(vals):
        return vals[0]
    else:
        bestAttribute = importance(data, attributes, targetAttribute, impurity)
        tree, tmp = {bestAttribute:{}}, []
        for i in data:
            if tmp.count(i[bestAttribute]) != 1: tmp.append(i[bestAttribute])
        for i in tmp:
            subAttributes = [attr for attr in attributes if attr != bestAttribute]
            exs = getSubDataset(data, bestAttribute, i)
            subtree = createDecisionTree(exs, subAttributes, targetAttribute,
                                         impurity, data)
            tree[bestAttribute][i] = subtree
    return tree
```

2.3 K-Fold Cross Validation

Per confrontare i diversi indici di impurità viene richiesta la tecnica della **5-fold cross validation**. Essa consiste in 5 test dove il dataset viene diviso ogni volta in un *trainset* ($\frac{4}{5}$ del dataset), da cui si costruisce l'albero, e in un *testset* ($\frac{1}{5}$ del dataset), usato per il test vero e proprio. Per ogni test effettuato i due set devono essere DIVERSI (non devono contenere righe presenti nei test precedenti).

Per misurare l'accuratezza rispetto ad ogni test (o *cross validation*) si ha uno score calcolato come $\frac{\text{numero di predizioni corrette}}{\text{numero di predizioni totali}}$.

Segue il codice della funzione *kFoldCrossValidation()*. All'interno si nota l'uso di una sotto funzione *getTargetValue()* che controlla, data una riga del testset, il valore dell'attributo target che l'albero ritorna (utile per il calcolo dello score).

```
def kFoldCrossValidation(data, attributes, targetAttribute, impurity, k):
    numberExamples = len(data)
    totalScoreAverage = 0.0
    scores = []
    for i in range(0, k):
        tmpScore = 0.0
        trainSet = data[:]
        testSet = []
        tmp = []
        for j in range(0, i*numberExamples/k):
            tmp.append(trainSet.pop(0))
        trainSet.extend(tmp)
        for j in range(0, numberExamples/k if i != k-1 else
                                (numberExamples/k)+numberExamples % k):
            testSet.append(trainSet.pop(0))
        tree = createDecisionTree(trainSet, attributes, targetAttribute,
                                  impurity, None)
        for j in testSet:
            if getTargetValue(tree, j) is not None:
                tmpScore = tmpScore + 1.0
        tmpScore = tmpScore / len(testSet)
        totalScoreAverage = totalScoreAverage + tmpScore
        scores.append(float(Decimal(tmpScore).quantize(Decimal('0.00001'))))
    return scores, Decimal(totalScoreAverage/k).quantize(Decimal('0.00001'))
```

3 Risultati Sperimentali

Per il test da effettuare viene richiesta la scelta di 3 dataset diversi scelti tra quelli presenti nel sito MLData. I dataset scelti sono indicati in un file all'interno del progetto (per ogni dataset sono indicati url e lista di attributi).

I risultati mostrati in output dall'applicazione rispecchiano (almeno in parte) quelli teorici, infatti l'indice di *misclassification* ha uno score più basso rispetto a quello di *gini* e di *entropia*.

Si nota però che tra indice di *gini* e indice di *entropia* non c'è grande differenza in fatto di accuratezza, a differenza di ciò che la teoria afferma (infatti per la sua definizione matematica, l'indice di *entropia* ha una curva di costo più alta); in realtà i risultati empirici, riportati per esempio in questo articolo, dimostrano che solo nel 2% dei casi l'indice di *entropia* è migliore di quello di *gini*; anche riguardo quest'ultimo punto i risultati ottenuti sono corretti.

```
DATABASE NUMBER 1 : nurseryClassifier.csv
5-Fold Cross Validation (with impurity type = 0) is starting ...
Test number 5 of 5 finished!
5-Fold Cross Validation (with impurity type = 1) is starting ...
Test number 5 of 5 finished!
5-Fold Cross Validation (with impurity type = 2) is starting ...
Test number 5 of 5 finished!

Scores with MISCLASSIFICATION : [0.97184, 0.97647, 0.97608, 0.98148, 0.97261]  |  Average : 0.97569
Scores with GINI : [0.98688, 0.98495, 0.98302, 0.98688, 0.98457]  |  Average : 0.98526
Scores with ENTROPY : [0.98302, 0.98495, 0.98225, 0.98341, 0.98302]  |  Average : 0.98333


DATABASE NUMBER 2 : letterClassifier.csv
5-Fold Cross Validation (with impurity type = 0) is starting ...
Test number 5 of 5 finished!
5-Fold Cross Validation (with impurity type = 1) is starting ...
Test number 5 of 5 finished!
5-Fold Cross Validation (with impurity type = 2) is starting ...
Test number 5 of 5 finished!

Scores with MISCLASSIFICATION : [0.84025, 0.8405, 0.83675, 0.83725, 0.83075]  |  Average : 0.83710
Scores with GINI : [0.85425, 0.8425, 0.8585, 0.85025, 0.86]  |  Average : 0.85310
Scores with ENTROPY : [0.84475, 0.8555, 0.85825, 0.851, 0.858]  |  Average : 0.85350


DATABASE NUMBER 3 : carClassifier.csv
5-Fold Cross Validation (with impurity type = 0) is starting ...
Test number 5 of 5 finished!
5-Fold Cross Validation (with impurity type = 1) is starting ...
Test number 5 of 5 finished!
5-Fold Cross Validation (with impurity type = 2) is starting ...
Test number 5 of 5 finished!

Scores with MISCLASSIFICATION : [0.85797, 0.82899, 0.89565, 0.88406, 0.87356]  |  Average : 0.86805
Scores with GINI : [0.87826, 0.87536, 0.92174, 0.92754, 0.89368]  |  Average : 0.89932
Scores with ENTROPY : [0.87826, 0.87536, 0.92464, 0.92754, 0.89368]  |  Average : 0.89990
```

4 Uso del codice e Riferimenti

Per usare l'applicazione si deve inserire nella lista *dataSets* i file CSV dei dataset e nella lista *targetPositions* le posizioni degli attributi target.

Per ulteriori dettagli e per i riferimenti si rimanda al file ReadMe del progetto caricato su GitHub.