Alberi di Decisione
DataSet privo di alcuni valori

Lorenzo De Luca

5 aprile 2018

abstract:

In questo elaborato si implementa l'algoritmo DECISION-TREE LEARNING in modo che possa gestire i casi in cui alcuni record del dataset sono privi di qualche attributo.

1 Introduzione

In questo elaborato viene risolto il problema 18.9 in R&N 2009, inerente alla costruzione di un albero di decisione che sia in grado di gestire la mancanza di attributi all'interno del dataset.
In particolare sono stati implementati due metodi:

1. Supporre che il record a cui manca l'attributo abbia tutti i possibili valori dell'attributo, pesando però ognuno di essi in base alla sua frequenza negli esempi che raggiungono quel nodo
2. Sostituire nei record a cui manca l'attributo un valore stimato in base alla frequenza dei valori sull'attributo in tutto il dataset

2 implementazione Python

2.1 Dataset

Inanzi tutto dopo aver importato un dataset, attraverso la funzione "depopulateDataset", nel file *DataSet.py*, sono stati rimossi, con una certa probabilità p, i valori degli attributi

```
def depopulateDataSet(dataSet, attributes, targetAttribute, prob):
    for i in range(0, len(dataSet)-1):
        x = random.random()
        if x <= prob:
            d = dataSet[i]
            y = random.randint(0, len(attributes)-1)
            if attributes[y] != targetAttribute:
                d[attributes[y]] = '?'

    return dataSet
```

2.2 DecisionTree

Successivamente è stata implementata, all'interno del file *DecisionTree.py* la funzione che ci ha permesso di realizzare l'albero di decisione; in particolare per calcolare l'attributo con il maggior information gain è stata usata l'entropia:

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

```
def DECISION_TREE_LEARNING(examples, attributes, targetAttribute, parent_examples, method):
    # mette in vals tutti i valori del targetAttribute
    vals = [record[targetAttribute] for record in examples]
```

```
    if not examples:
        return pluralityValue(parent_examples, targetAttribute)
    elif (len(attributes)-1) <= 0:
        return pluralityValue(examples, targetAttribute)
    elif vals.count(vals[0]) == len(vals):
        return vals[0]
    else:
        A = importance(attributes, examples, targetAttribute, method)
        tree = {A: {}}
        bestAttribute_value = get_values(examples, A)
        # costruisco il sottoalbero con chiamata ricorsiva dopo aver definito subExamples e
            subAttributes
        for value in bestAttribute_value:
            subExamples = build_subExamples(examples, A, value)
            subAttributes = [att for att in attributes if att != A]
            subTree = DECISION_TREE_LEARNING(subExamples, subAttributes, targetAttribute,
                examples, method)

            tree[A][value] = subTree
    return tree
```

Quindi durante il calcolo del *information gain*, andiamo a controllare se siamo nel caso in cui l'attributo ha un valore mancante, e in base al metodo applicato ('A' o 'B') si gestisce la situazione.
La definizione delle funzioni *Method_A* e *Method_B*, che corrispondono al problema posto dall'esercizio, possiamo trovarla all'interno del file *Missing_Attribute.py*, e si occupano di ripopolare il dataSet con il metodo adeguato.

Il codice sottoriportato è decorato con qualche breve commento affinchè sia più comprensibile ciò che sta svolgendo la funzione

```
def Method_A(data, attribute):
    val_freq = {}
    count = 0

    # calcolo la frequenza di ciascun attributo che abbia un valore
    for record in data:
        if record[attribute] in val_freq:
            val_freq[record[attribute]] += 1.0
            count += 1
        elif record[attribute] not in val_freq and record[attribute] != '?':
            val_freq[record[attribute]] = 1.0
            count += 1

    prob_val = val_freq.copy()

    # calcolo la probabilita' della frequenza all'interno del dataSet di ciascun valore
    for val in val_freq.keys():
        prob_val[val] = val_freq[val]/count

    probAtt = []
    valAtt = []
    for val in prob_val.keys():
        probAtt.append(prob_val[val])
        valAtt.append(val)
    for i in range(len(probAtt)):
```

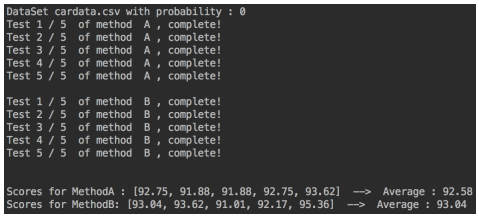
```
        if i == 0:
            probAtt[i] = probAtt[i]
        else:
            probAtt[i] = probAtt[i]+probAtt[i-1]
    # scelgo il valore per ripopolare il dataSet pesandolo in base alla sua frequenza
    x = random.random()
    for p in probAtt:
        if x < p:
            scelta = p
            break

    for i in range(len(probAtt)):
        if probAtt[i] == scelta:
            l = i

    newValue = valAtt[l]
    return newValue


def Method_B(data, attribute):
    val_freq = {}
    equalVal_freq = []
    valMax = 0

    # calcolo la frequenza di ciascun attributo che abbia un valore
    for record in data:
        if record[attribute] in val_freq:
            val_freq[record[attribute]] += 1.0
        elif record[attribute] not in val_freq and record[attribute] != '?':
            val_freq[record[attribute]] = 1.0

    # calcolo il valore con la frequenza massima
    for val in val_freq.keys():
        if val_freq[val] > valMax:
            valMax = val_freq[val]

    # gestisco il caso in cui piu' di un attributo ha la stessa frequenza
    for val in val_freq.keys():
        if val_freq[val] == valMax:
            equalVal_freq.append(val)
    #scelgo com valore per ripopolare il dataSet quello con la frequenza massima
    if len(equalVal_freq) > 1:
        x = random.randint(0, len(equalVal_freq)-1)
        newValue = equalVal_freq[x]
    else:
        newValue = equalVal_freq[0]

    return newValue
```

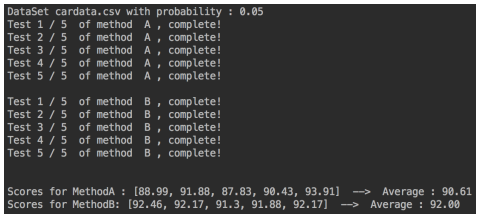
2.3 Accuratezza

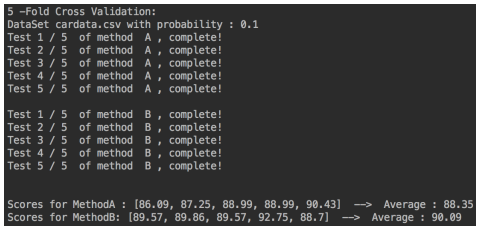
Infine ne è stata calcolata l'accuratezza, usando il metodo della Cross Validation; si rimandanda al codice nel file *CrossValidation.py*

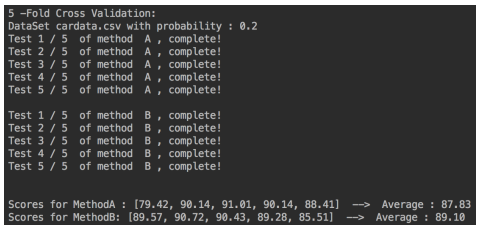
3 Risultati sperimentali

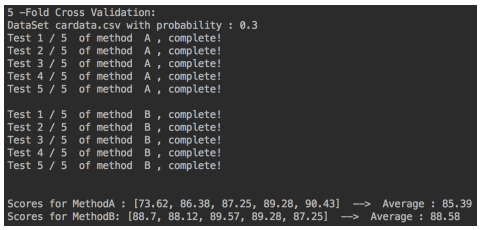
Nel test vengono analizzati 3 dataset diversi scelti dal repository UCI, ciascuno dei quali doveva essere privato di dei valori di alcuni attributi con probabilita $p = [0, 0.05, 0.1, 0.2, 0.3]$.
Di seguito sono riportati i test applicati solo a uno dei tre dataSet, ovvero *cardata.csv*, poichè i risulatati si rispecchiano in ciascuno dei dataSet esaminati:











in particolare notiamo come la percentuale di accuratezza sia inversamente proporzionale alla probabilità applicata, questione che rispecchia l'aspetto teorico in quanto la presenza di più valori mancanti comporta necessariamente una costruzione meno accurata del nostro albero di decisione.
Inoltre possiamo notare come il metodo B, sia sempre leggermente più accurato del metodo A.

4 Note

Per riprodurre i risultati riportati è necessario riprendere tutto il materiale dalla mia pagina di GitHub, in cui è possibile sfruttare il file ReadMe per capire meglio la struttura del progetto.