

Relatório - Compêndio 2

COMP0427 - Inteligência Artificial - 2023.2
Igor Nathan Monteiro Santos - 202100011495
Prof. Dr. Hendrik Teixeira Macedo

1. Introdução

Este documento descreve os algoritmos desenvolvidos no 2º Compêndio da disciplina de Inteligência Artificial.

- Link para o repositório: <https://github.com/Igor-Mont/COMP0427-IA>.
- Link para o Replit: <https://replit.com/join/ixfofhxdcv-igor-mont>

Cada seção de um algoritmo segue a seguinte estrutura:

- Descrição
- Associação (entre o pseudocódigo e a implementação)

OBS: Nesse compêndio consegui fazer apenas 2, como foram grupos diferentes fiz um com o meu grupo do primeiro compêndio (ID3) e o rejection-sampling fiz sozinho.

2. Algoritmo 05: Rejection-Sampling

O algoritmo rejection-sampling, ou amostragem por rejeição, é uma técnica fundamental para gerar amostras aleatórias de uma distribuição de probabilidade específica, quando a geração direta é difícil ou impossível. A ideia principal é usar uma distribuição "candidata" mais simples, da qual é possível gerar amostras facilmente, para gerar amostras da distribuição desejada.

```
function REJECTION-SAMPLING( $X, e, bn, N$ ) returns an estimate of  $P(X | e)$   
  inputs:  $X$ , the query variable  
          $e$ , observed values for variables  $E$   
          $bn$ , a Bayesian network  
          $N$ , the total number of samples to be generated
```

```
  local variables:  $C$ , a vector of counts for each value of  $X$ , initially zero
```

```
  for  $j = 1$  to  $N$  do  
     $x \leftarrow \text{PRIOR-SAMPLE}(bn)$   
    if  $x$  is consistent with  $e$  then  
       $C[j] \leftarrow C[j] + 1$  where  $x_j$  is the value of  $X$  in  $x$   
  return NORMALIZE( $C$ )
```

```

1  function rejection_sampling(
2      X: string,
3      e: Map<string, boolean>,
4      bn: BayesNet,
5      N = 10000
6  ) {
7      const variable_values = bn.variable_values(X);
8      const counts = new Map<(typeof variable_values)[0], number>();
9      variable_values.forEach((variable_value) => {
10         counts.set(variable_value, 0);
11     });
12
13     for (let j: number = 0; j < N; j++) {
14         const sample = prior_sample(bn);
15
16         if (consistent_with(sample, e)) {
17             const key = sample.get(X) as boolean;
18             const current_value_count = counts.get(key) as number;
19             counts.set(key, current_value_count + 1);
20         }
21     }
22
23     return new ProbDist(X, counts).show_approx();
24 }

```

3. Algoritmo 06: Árvore de Decisão

Ele constrói árvores de decisão a partir de um dado conjunto de exemplos, sendo a árvore resultante usada para classificar amostras futuras.

function LEARN-DECISION-TREE(*examples*, *attributes*, *parent_examples*) **returns** a tree

```

if examples is empty then return PLURALITY-VALUE(parent_examples)
else if all examples have the same classification then return the classification
else if attributes is empty then return PLURALITY-VALUE(examples)

```

else

```

    A ← argmaxa ∈ attributes IMPORTANCE(a, examples)
    tree ← a new decision tree with root test A

```

for each value *v* of *A* **do**

```

    exs ← { e : e ∈ examples and e.A = v }
    subtree ← LEARN-DECISION-TREE(exs, attributes − A, examples)
    add a branch to tree with label (A = v) and subtree subtree

```

return *tree*

```
1 buildTree(examples, attrs, parentExamples=[]) {
2   if (examples.length === 0) {
3     return this.pluralityValue(parentExamples);
4   }
5   if (this.allSameClass(examples)) {
6     return new DecisionLeaf(examples[0][this.dataset.target]);
7   }
8   if (attrs.length === 0) {
9     return this.pluralityValue(examples);
10  }
11
12  let A = this.chooseAttribute(attrs, examples);
13  let tree = new DecisionFork(
14    A,
15    this.dataset.attrNames[A],
16    this.pluralityValue(examples)
17  );
18
19  for (const [v, e] of this.splitBy(A, examples)) {
20    let subtree = this.buildTree(e, utils.removeAll(A, attrs), examples);
21    tree.add(v, subtree);
22  }
23  return tree;
24 }
```

```
1  pluralityValue(examples) {
2    let popular = utils.argmax(
3      this.dataset.values[this.dataset.target],
4      (v) => this.count(this.dataset.target, v, examples)
5    );
6    return new DecisionLeaf(popular);
7  }
8
9  informationGain(attr, examples) {
10   const I = (exs) => {
11     let values = [];
12     for (const v of this.dataset.values[this.dataset.target]) {
13       let count = this.count(this.dataset.target, v, exs);
14       values.push(count);
15     }
16     return informationContent(values);
17   };
18
19   let n = examples.length;
20   let entropies = [];
21   for (const [v, e] of this.splitBy(attr, examples)) {
22     let entropy = (e.length / n) * I(e);
23     entropies.push(entropy);
24   }
25   let remainder = utils.sum(entropies);
26
27   return I(examples) - remainder;
28 }
29
30 splitBy(attr, examples) {
31   let vals = this.dataset.values[attr];
32   let result = [];
33   for (const v of vals) {
34     let valExamples = [];
35     for (const e of examples) {
36       if (e[attr] === v) {
37         valExamples.push(e);
38       }
39     }
40
41     result.push([v, valExamples]);
42   }
43   return result;
44 }
```