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

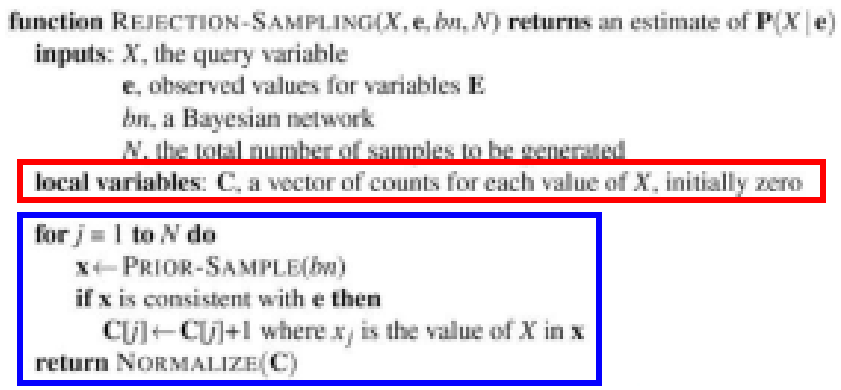
Cada seção de um algoritmo segue a seguinte estrutura:

- 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

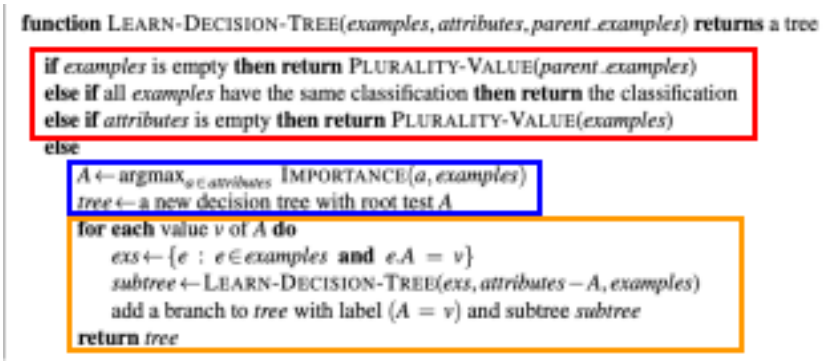
O algoritmo rejection-sampling, ou amostragem por rejeição, é uma técnica fundamental para gerar amostras aleatórias de uma distribuição de probabilidade específica, quando a geração direta é difícil ou impossível. A ideia principal é usar uma distribuição "candidata" mais simples, da qual é possível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



```typescript
function rejection_sampling(
  X: string,
  e: Map<string, boolean>,
  bn: BayesNet,
  N = 10000
) {
  const variable_values = bn.variable_values(X);
  const counts = new Map<(typeof variable_values)[0], number>();
  variable_values.forEach((variable_value) => {
    counts.set(variable_value, 0);
  });

  for (let j: number = 0; j < N; j++) {
    const sample = prior_sample(bn);

    if (consistent_with(sample, e)) {
      const key = sample.get(X) as boolean;
      const current_value_count = counts.get(key) as number;
      counts.set(key, current_value_count + 1);
    }
  }

  return new ProbDist(X, counts).show_approx();
}
```

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.



```
buildTree(examples, attrs, parentExamples=[]) {
  if (examples.length === 0) {
    return this.pluralityValue(parentExamples);
  }
  if (this.allSameClass(examples)) {
    return new DecisionLeaf(examples[0][this.dataset.target]);
  }
  if (attrs.length === 0) {
    return this.pluralityValue(examples);
  }

  let A = this.chooseAttribute(attrs, examples);
  let tree = new DecisionFork(
    A,
    this.dataset.attrNames[A],
    this.pluralityValue(examples)
  );

  for (const [v, e] of this.splitBy(A, examples)) {
    let subtree = this.buildTree(e, utils.removeAll(A, attrs), examples);
    tree.add(v, subtree);
  }
  return tree;
}
```

```
pluralityValue(examples) {
  let popular = utils.argmax(
    this.dataset.values[this.dataset.target],
    (v) => this.count(this.dataset.target, v, examples)
  );
  return new DecisionLeaf(popular);
}

informationGain(attr, examples) {
  const I = (exs) => {
    let values = [];
    for (const v of this.dataset.values[this.dataset.target]) {
      let count = this.count(this.dataset.target, v, exs);
      values.push(count);
    }
    return informationContent(values);
  };

  let n = examples.length;
  let entropies = [];
  for (const [v, e] of this.splitBy(attr, examples)) {
    let entropy = (e.length / n) * I(e);
    entropies.push(entropy);
  }
  let remainder = utils.sum(entropies);

  return I(examples) - remainder;
}

splitBy(attr, examples) {
  let vals = this.dataset.values[attr];
  let result = [];
  for (const v of vals) {
    let valExamples = [];
    for (const e of examples) {
      if (e[attr] === v) {
        valExamples.push(e);
      }
    }

    result.push([v, valExamples]);
  }
  return result;
}
```