

Árvores de Decisão

Otacílio de Araújo Ramos Neto

IFPB

Novembro 4, 2022

Tópicos

- 1 O que é uma Árvore de Decisão?
- 2 Entropia
- 3 Criando a Árvore
- 4 Montando tudo
- 5 Referências

Árvore de Decisão

Uma árvore de decisão representa, em uma estrutura de árvore, um determinado número de caminhos possíveis de decisão e os resultados de cada um deles.

Exemplo

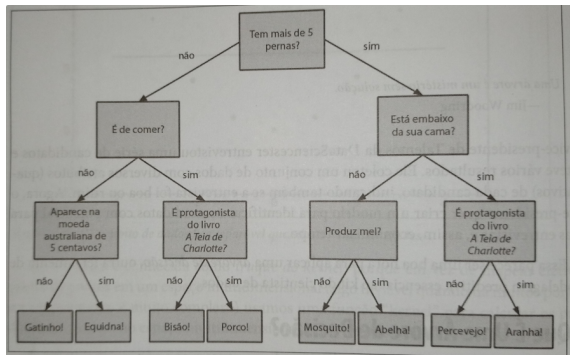


Figura: Árvore de exemplo para adivinhação do animal.

Propriedades pertinentes

- Fáceis de entender e interpretar;
- Lidam facilmente com diversos atributos numéricos e categóricos;
- Construir um árvore de decisão “ideal” para um conjunto de dados de treinamento é um problema computacional complexo;
- Árvores práticas são “razoáveis”, não “ideais”;
- Deve-se tomar cuidado para não construir árvores com “sobreajuste” a um conjunto de dados.

Tópicos

- 1 O que é uma Árvore de Decisão?
- 2 Entropia
- 3 Criando a Árvore
- 4 Montando tudo
- 5 Referências

Entropia

Conceitos pertinentes

- Para construir a árvore de decisão, temos que definir as perguntas que faremos e sua sequência;
- Cada pergunta corresponde a uma propriedade. Exemplo: "Qual a sua experiência?";
- As respostas das perguntas estão nas propriedades do objeto sob análise;
- Boas perguntas fornecem muitas informações sobre o assunto em questão. Isto implica dizer que a propriedade forma um conjunto com respostas variadas;
- Perguntas ruins não fornecem muitas informações novas. Por exemplo, perguntar se um profissional é humano quando se está tentando descobrir a profissão de uma pessoa qualquer;
- Para medir a quantidade de informação vamos usar a entropia, ela será a nossa medida de incerteza relacionada aos dados.

Entropia

Cálculo da entropia

- Considere um conjunto S em que cada membro pertence a uma classe de um conjunto finito de classes C_1, C_2, \dots, C_n conforme uma de suas propriedades;
- Se todos os membros pertencem a uma mesma classe então não existe incerteza real, ou seja, a entropia é baixa;
- Se os membros estão distribuídos homogeneamente em todas as classes então a incerteza é alta, logo, a entropia é alta.

Entropia

Cálculo da entropia

- Matematicamente, seja p_i a proporção dos dados rotulados como pertencentes a classe C_i de uma determinada propriedade. Dessa forma definimos a entropia do conjunto S com relação a propriedade em questão como sendo

$$H(S) = \sum_{i=1}^n -p_i \log_2(p_i)$$

com a convenção de que $0 \log_2(0) = 0$;

- Note que $-p_i \log_2(p_i)$ é não negativo e está próximo de zero quando p_i está próximo de 0 ou 1.

Entropia

Gráfico

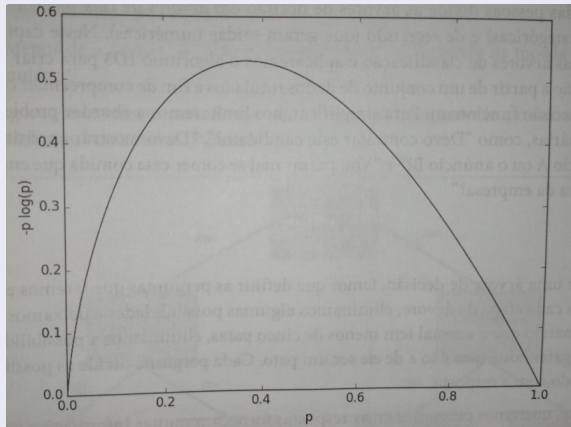


Figura: Gráfico de $-p \log_2(p)$.

Entropia

Implementação da entropia

```
from typing import List
from math import log

def entropy(class_probabilities: List[float]) -> float:
    """Calcula a entropia para a lista de probabilidades
    de classe informada"""

    return sum(-p*log(p, 2)
               for p in class_probabilities
               if p > 0)

assert entropy([1.0]) == 0
assert entropy([0.5, 0.5]) == 1
assert 0.81 < entropy([0.25, 0.75]) < 0.82
```

Entropia

Implementação da entropia dos dados

```
from typing import List, Any
from collections import Counter
from entropia import *

def class_probabilities(labels: List[Any]) -> List[float]:
    total_count = len(labels)
    return [count / total_count
            for count in Counter(labels).values()]

def data_entropy(labels: List[Any]) -> float:
    return entropy(class_probabilities(labels))

assert data_entropy(['a']) == 0
assert data_entropy([True, False]) == 1
assert data_entropy([3, 4, 4, 4]) == entropy([0.25, 0.75])
```

Entropia

Entropia de uma partição

- Cada nó interno da árvore de decisão traz uma pergunta cuja resposta particiona os dados em um ou mais subconjuntos;
- Dessa forma, é útil definir a entropia decorrente do particionamento resultante de uma pergunta;
- Sejam os conjuntos S_1, S_2, \dots, S_m resultantes do particionamento dos dados de um conjunto S . A entropia da partição é dada pela soma da entropia de cada conjunto (correspondente a outra propriedade) ponderada pela proporção do conjunto na propriedade.

$$H = q_1 H(S_1) + \dots + q_m H(S_m) = \sum_{i=1}^m q_i H(S_i)$$

Entropia

Implementação da entropia da partição

```
from typing import List, Any
from entropia_dados import *

def partition_entropy(subsets: List[List[Any]]) -> float:
    """Retorna a entropia da partição
    dos dados em subconjuntos"""
    total_count = sum(len(subset) for subset in subsets)

    return sum(data_entropy(subset) * len(subset) / total_count
               for subset in subsets)
```

Tópicos

- 1 O que é uma Árvore de Decisão?
- 2 Entropia
- 3 Criando a Árvore
- 4 Montando tudo
- 5 Referências

Base de dados para a criação da árvore

```
from typing import NamedTuple, Optional
class Candidate(NamedTuple):
    level: str
    lang: str
    tweets: bool
    phd: bool
    did_well: Optional[bool] = None # Permite dados não rotulados
#           level      lang      tweets phd      did_well
inputs = [ Candidate('Senior', 'Java', False, False, False),
             Candidate('Senior', 'Java', False, True, False),
             Candidate('Mid', 'Python', False, False, True),
             Candidate('Junior', 'Python', False, False, True),
             Candidate('Junior', 'R', True, False, True),
             Candidate('Junior', 'R', True, True, False),
             Candidate('Mid', 'R', True, True, True),
             Candidate('Senior', 'Python', False, False, False),
             Candidate('Senior', 'R', True, False, True),
             Candidate('Junior', 'Python', True, False, True),
             Candidate('Senior', 'Python', True, True, True),
             Candidate('Mid', 'Python', False, True, True),
             Candidate('Mid', 'Java', True, False, True),
             Candidate('Junior', 'Python', False, True, False)
```

]

Algoritmo ID3

Passos do ID3

- 1 Se todos os dados tiverem o mesmo rótulo, crie um nó folha para prever este rótulo e pare;
- 2 Se a lista de atributos estiver vazia (ou seja, caso não haja mais perguntas possíveis para fazer), crie um nó folha para prever o rótulo mais comum e pare;
- 3 Caso contrário, particione os dados com base nos atributos;
- 4 Escolha a partição com a entropia mais baixa;
- 5 Adicione um nó de decisão com base no atributo escolhido;
- 6 Repita o procedimento em cada subconjunto particionado usando os demais atributos.

Algoritmo ID3

O passo 3 demanda o particionamento dos dados com base nos atributos. Uma rotina é necessária para isto.

```
from typing import Dict, TypeVar, List, Any
from collections import defaultdict

T = TypeVar('T') # tipo genérico para entrada de dados

def partition_by(inputs: List[T], attribute: str) -> Dict[Any, List[T]]:
    """Particione as entradas em listas com base no atributo especificado"""
    partitions: Dict[Any, List[T]] = defaultdict(list)
    for input in inputs:
        key = getattr(input, attribute) # Valor do atributo especificado
        partitions[key].append(input)   # Adiciona a entrada na partição

    return partitions
```

Algoritmo ID3

Também é necessária uma rotina para calcular a entropia correspondente a uma partição.

```
from partition_by import *
from partition_entropy import *
def partition_entropy_by(inputs: List[Any],
                        attribute: str,
                        label_attribute: str) -> float:
    """Calcula a entropia correspondente à partição especificada"""
    # as partições contém as entradas
    partitions = partition_by(inputs, attribute)

    # mas o partition_entropy só precisa dos rótulos das classes
    labels = [[getattr(input, label_attribute) for input in partition]
               for partition in partitions.values()]

    return partition_entropy(labels)
```

Algoritmo ID3

Agora que os algoritmos de particionamento e cálculo da entropia para o particionamento foram definidos, é necessário encontrar o particionamento com menor entropia.

```
from partition_entropy_by import *
from dados import *

for key in ['level', 'lang', 'tweets', 'phd']:
    print(key, partition_entropy_by(inputs, key, 'did_well'))

assert 0.69 < partition_entropy_by(inputs, 'level', 'did_well') < 0.70
assert 0.86 < partition_entropy_by(inputs, 'lang', 'did_well') < 0.87
assert 0.78 < partition_entropy_by(inputs, 'tweets', 'did_well') < 0.79
assert 0.89 < partition_entropy_by(inputs, 'phd', 'did_well') < 0.90
```

Algoritmo ID3

A menor entropia está na divisão por *level*. Neste ponto é criado um ramo para cada valor possível de *level* (figura 3).

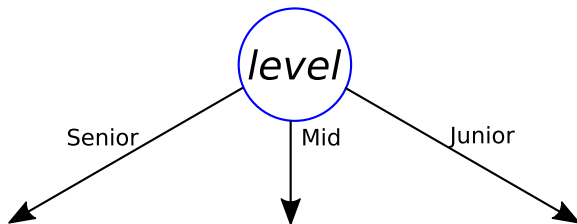


Figura: Árvore de decisão após a descoberta que *level* tem a menor entropia.

Algoritmo ID3

Os rótulos para os dados registram que sempre que o candidato tem experiência **Mid** ele é contratado. Dessa forma, é adicionado uma folha na árvore com a decisão correspondente.

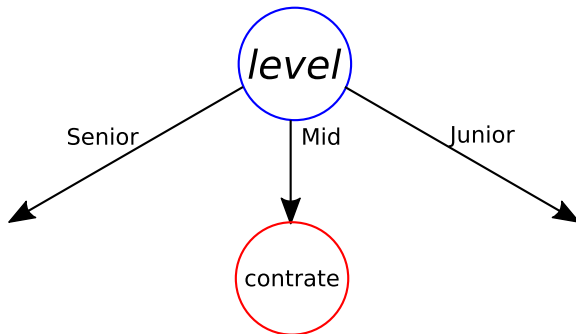


Figura: Árvore de decisão com a folha de contratação para a experiência **Mid**.

Algoritmo ID3

Considerando que os candidatos *Senior* possuem rótulo de “contrate” e “não contrate” é preciso calcular novamente a entropia para ramificar a árvore novamente.

```
from partition_entropy_by import *  
from dados import *
```

```
senior_inputs = [input for input in inputs if input.level == 'Senior']
```

```
assert 0.4 == partition_entropy_by(senior_inputs, 'lang', 'did_well')
```

```
assert 0.0 == partition_entropy_by(senior_inputs, 'tweets', 'did_well')
```

```
assert 0.95 < partition_entropy_by(senior_inputs, 'phd', 'did_well') < 0.96
```

Algoritmo ID3

Dado que para o *level* “Senior” a menor entropia é para o caso de ter ou não twitter a próxima ramificação usa as respostas para esta pergunta. Para candidatos “Senior” ter twitter sempre resulta em contratação, e os que não tem não são contratados.

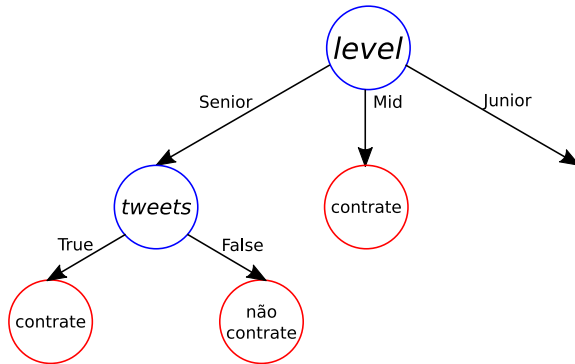


Figura: Árvore de decisão com a folha de contratação para candidatos “Senior” e que tem “twitter”.

Algoritmo ID3

Repetindo o processo do cálculo da entropia para os candidatos com *level* “Junior” encontramos que a menor entropia é para o particionamento por *phd*. Os dados estão rotulados e indicam que para o *level* “Junior” não ter PhD retorna sempre em **contrate** enquanto ter PhD retorna sempre *não contrate*.

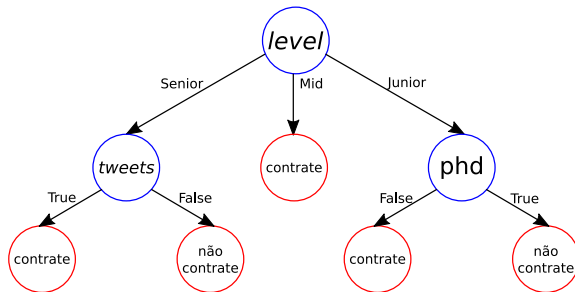


Figura: Árvore de decisão com a folha de contratação para candidatos “Junior” e que tem ou não “PhD”.

Tópicos

- 1 O que é uma Árvore de Decisão?
- 2 Entropia
- 3 Criando a Árvore
- 4 Montando tudo**
- 5 Referências

Modelando os vértices

Modelando os vértices

- Modelando o *Leaf* para conter um só valor;
- Modelando o *Split* com um atributo para orientar a divisão, dicionários para as subárvores e um valor padrão para valores desconhecidos.

Modelando os vértices

```
from typing import NamedTuple, Union, Any

class Leaf(NamedTuple):
    value: Any

class Split(NamedTuple):
    attribute: str
    subtrees: dict
    default_value: Any = None

DecisionTree = Union[Leaf, Split]
```

Modelando a árvore

```
from leafsplit import *
hiring_tree = Split('level', {# Primeiro vértice é level
    'Junior': Split('phd', { # Se level tem resposta Junior vai no vértice phd
        False: Leaf(True), # Se phd tem valor False vai no vértice True
        True: Leaf(False)  # Se phd tem valor True vai no vértice False
    }),
    'Mid': Leaf(True), # Se level tem resposta Mid vai no vértice True
    'Senior': Split('tweets', { # Se level tem resposta Senior vai em tweets
        False: Leaf(False), # Se tweets tem resposta False vai no vértice False
        True: Leaf(True) # Se tweets tem resposta True vai no vértice True
    })
})
```

Rotina de classificação

```
from decisiontree import *  
def classify(tree: DecisionTree, input: Any) -> Any:  
    """classifique a entrada usando a árvore de decisão indicada"""  
  
    # Se for um nó folha, retorne o seu valor  
    if(isinstance(tree, Leaf)):  
        return tree.value  
  
    # Caso contrário, a árvore consiste em um atributo de divisão  
    # e um dicionário cujas chaves são valores deste atributo  
    # e cujos valores são subárvores que serão consideradas em seguida  
    subtree_key = getattr(input, tree.attribute)  
  
    if subtree_key not in tree.subtrees: # Se não houver subárvore para a chave  
        return tree.default_value # retorne o valor padrão  
  
    subtree = tree.subtrees[subtree_key] # Escolha a subárvore adequada  
    return classify(subtree, input) # e use-a para classificar a entrada
```

Construindo a representação

```
from typing import List
from partition_entropy_by import *
from classify import *

def build_tree_id3(inputs: List[Any],
                   split_attributes: List[str],
                   target_attribute: str) -> DecisionTree:
    # Conte os rótulos especificados
    label_counts = Counter(getattr(input, target_attribute)
                           for input in inputs)
    most_common_label = label_counts.most_common(1)[0][0]

    # Se houver só um rótulo, preveja esse rótulo
    if len(label_counts) == 1:
        return Leaf(most_common_label)

    # Se não restar nenhum atributo de divisão, retorne o rótulo majoritário
    if not split_attributes:
        return Leaf(most_common_label)

    """Continua..."""
```

Construindo a representação

```
# Caso contrário, divida pelo melhor resultado
def split_entropy(attribute: str) -> float:
    """A função auxiliar para encontrar o melhor atributo"""
    return partition_entropy_by(inputs, attribute, target_attribute)

best_attribute = min(split_attributes, key=split_entropy)

partitions = partition_by(inputs, best_attribute)
new_attributes = [a for a in split_attributes if a != best_attribute]

# Construa recursivamente as subárvores
subtrees = {attribute_value : build_tree_id3(subset,
                                              new_attributes,
                                              target_attribute)
            for attribute_value, subset in partitions.items()}

return Split(best_attribute, subtrees, default_value=most_common_label)
```


Instanciando e fazendo testes

A árvore faz precisões perfeitas no treinamento, mas também deve ser capaz de generalizar, fazendo previsões para dados fora do conjunto de treinamento.

```
tree = build_tree_id3(inputs,
                      ['level', 'lang', 'tweets', 'phd'],
                      'did_well')

# Deve prever True
assert classify(tree, Candidate("Junior", "Java", True, False))

# Deve prever False
assert not classify(tree, Candidate("Junior", "Java", True, True))


# Aplicando a valores inesperados
assert classify(tree, Candidate("Intern", "java", True, True))
```

Tópicos

- 1 O que é uma Árvore de Decisão?
- 2 Entropia
- 3 Criando a Árvore
- 4 Montando tudo
- 5 Referências**

Livro

Todo o material apresentado nesta apresentação foi obtido do livro
“Data Science do Zero” (GRUS, 2021).

 GRUS, J. *Data Science do Zero*. [s.n.], 2021. ISBN 9788550803876. Disponível em: [⟨https://books.google.com.br/books?id=2LZwDwAAQBAJ⟩](https://books.google.com.br/books?id=2LZwDwAAQBAJ).