Árvores de Decisão

Conceito

A árvore de decisão é um tipo de algoritmo de aprendizagem de máquina supervisionado que se baseia na ideia de divisão dos dados em grupos homogêneos. Este algoritmo é utilizado para tratar problemas de classificação ou regressão.

Em problemas de classificação, este algoritmo realiza a previsão de categorias discretas (verdadeiro ou falso; sim ou não; 1 ou 0). Por outro lado, em problemas de regressão, este algoritmo é utilizado para prever valores numéricos (valor de um determinado produto; o lucro em reais de uma determinada campanha de *marketing*).

O principal objetivo do algoritmo árvore de decisão é encontrar o atributo ou variável independente que melhor realiza a divisão dos dados. Existem diversas técnicas que permitem identificar o melhor atributo que faz esta divisão de dados e gerando a partição mais pura. Algumas destas métricas, por exemplo, são: *Information Gain*, Redução de Variância, $Entropia$, $Gini$ e *Chi-Square*.

Como funciona

Podemos assimilar o funcionamento deste algoritmo como um fluxograma. Isto porque, as árvores de decisão estabelecem nós de decisão que se relacionam entre si dentro de uma hierarquia. Além dos nós de decisão, existem também o nó raiz e os nós folha. O nó raiz é o nó mais importante devido segregar o conjunto de dados encontrando a partição mais pura conforme citado acima.

Os nós folha são os resultados finais gerados pelo conjunto de decisões das ramificações percorridas pelo algoritmo dentro da árvore. No contexto de *machine learning*, o nó raiz e os nós de decisão são atributos da base de dados, sendo o nós raiz o mais importante e os nós de decisão o conjunto de regras que caminha para os nós folha. Por fim, o nó folha é a classe ou valor que será gerado como resposta.

O topo ou início da árvore é determinada pelo nó raiz que inicia a divisão dos dados. Cada divisão subsequente gerada pelos nós de decisão são realizadas a partir de amostras de testes. As divisões dos dados pelos nós de decisão são realizadas até alcançar os nós folha que representam a classe ou variável dependente. Neste nível final, caso estejamos lidando com um problema de classificação teremos resultados como por exemplo, sim ou não, ou um valor médio das observações caso estejamos lidando com problemas de regressão.

Através da Figura 1 – *Decision Tree*, é apresentado uma ilustração do funcionamento das árvores de decisão para classificar se devemos comprar mais produtos ou não baseados em ofertas geradas. Ainda, neste exemplo, podem ser vistos o nó raiz, nós de decisão e nós folha que constituem o algoritmo como um todo.

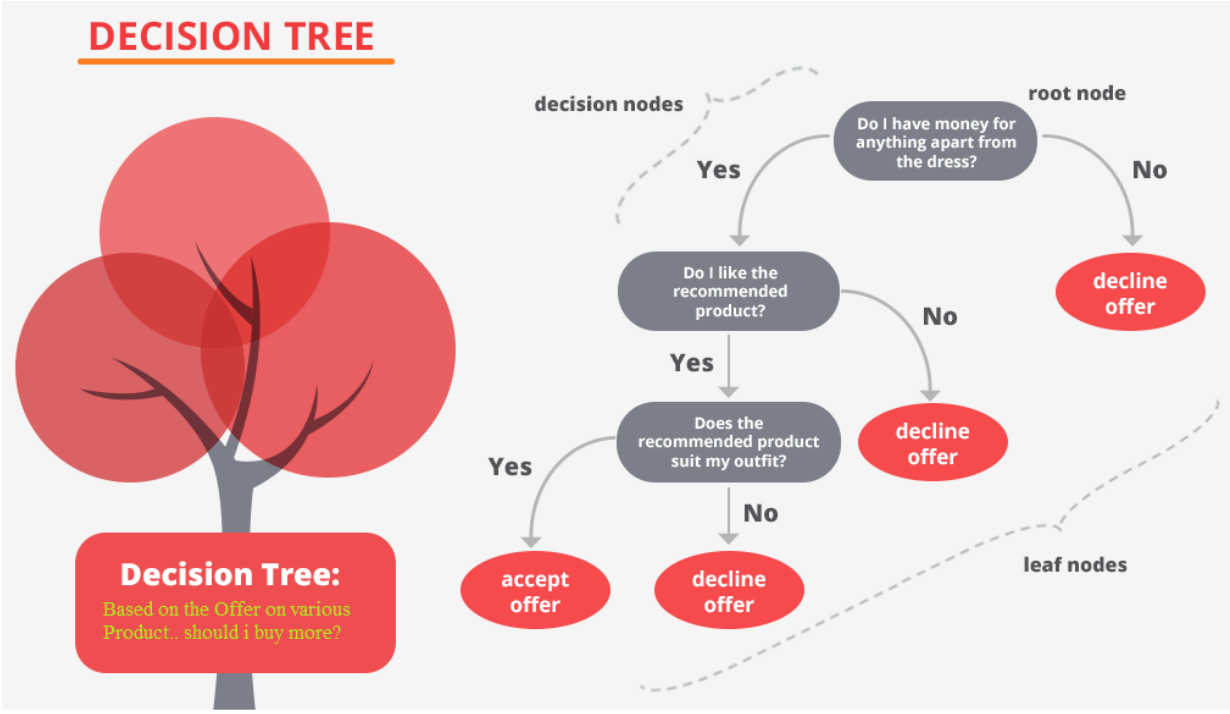

Figura 1 – *Decision Tree*

Em cada ligação entre os nós, o algoritmo pergunta acerca de uma condição ("_if-else_") gerada fundamentada no conhecimento absorvido dos dados pelo algoritmo e realiza a divisão dos dados com base nas regras ou condições. Por exemplo, se estamos lidando com uma base de pessoas, uma condição que poderia ser gerada seria considerar pessoas com idade maior de 18 anos.

Desta maneira, se 70% dos meus dados possuem pessoas com idade maior do que 18 anos, então uma ramificação do nó será constituída com 70% de dados com pessoas acima de 18 anos e a outra ramificação com 30% de dados com pessoas com idade igual ou menor a 18 anos. Esta lógica é reproduzida nos demais atributos através de novas regras até chegar nos nós folha.

Como pode ser visto, o maior trabalho realizado pelas árvores é encontrar os nós que vão compor cada posição da sua estrutura. Para isso, alguns cálculos se tornam importantes para auxiliar na identificação de cada um dos nós de decisão e encontrar o melhor critério de divisão ou a melhor divisão com base em medidas de impureza. Os 3 principais critérios de divisão usados nas árvores de decisão são:

- Impureza ou Índice $Gini$: A impureza $Gini$ é uma medida da impureza de um nó. Esta medida quantifica a quantidade de vezes que um elemento escolhido aleatoriamente do conjunto de dados seria rotulado de maneira incorreta se fosse rotulado aleatoriamente de acordo com a distribuição de rótulos do subconjunto. É a maneira mais popular e fácil de dividir uma árvore de decisão e funciona apenas com alvos categóricos, pois faz apenas divisões binárias. Quanto menor a Impureza $Gini$, maior a homogeneidade do nó. A Impureza $Gini$ de um nó puro (mesma classe) é igual a zero. A fórmula para calcular a Impureza $Gini$ é a:

$$gini(R) = \sum p(c|R)(1 - p(c|R))$$

Onde: $p(c|R)$ é a probabilidade de um ponto da região R pertencer a classe C.

- Entropia: A Entropia representa a falta de uniformidade ou uma medida de aleatoriedade nos dados. Quanto mais alta a entropia, mais caótico e misturados estão os dados e quanto menor a entropia, mais uniforme e homogênea está o conjunto de dados. A fórmula para se calcular a entropia é:

$$entropia(R) = \sum p(c|R)\log(p(c|R))$$

Onde: A probabilidade é estimada pela razão entre quantidade de pontos da classe c e o total de pontos em R

- Ganho de informação: O ganho de informação é uma propriedade estatística que mede quão bem um determinado atributo separa os exemplos de treinamento de acordo com sua classificação alvo ou rótulo. Em outras palavras, o ganho de informação representa a informação aprendida sobre os rótulos quando dividimos uma região do espaço em duas sub-regiões de acordo com um critério de divisão como a entropia ou impureza $gini$, citadas acima. A fórmula que define o ganho de informação é a:

$$InfoGain(R, R_e, R_d) = H(R) - (|R_e| * H(R_e) + |R_d| * H(R_d))/|R|$$

Onde:
H é a impureza da região R é a região atual R_e é sub-região da esquerda R_d é sub-região da direita |.| é quantidade de exemplos na dada região

Para apresentar na prática, vamos considerar o gráfico abaixo da Figura 2 - Distribuição de Produtos, onde é apresentada a distribuição de produtos de acordo com suas medidas em termos de largura e altura. Vamos calcular o ganho de informação para o ponto de corte representado na Figura 2.

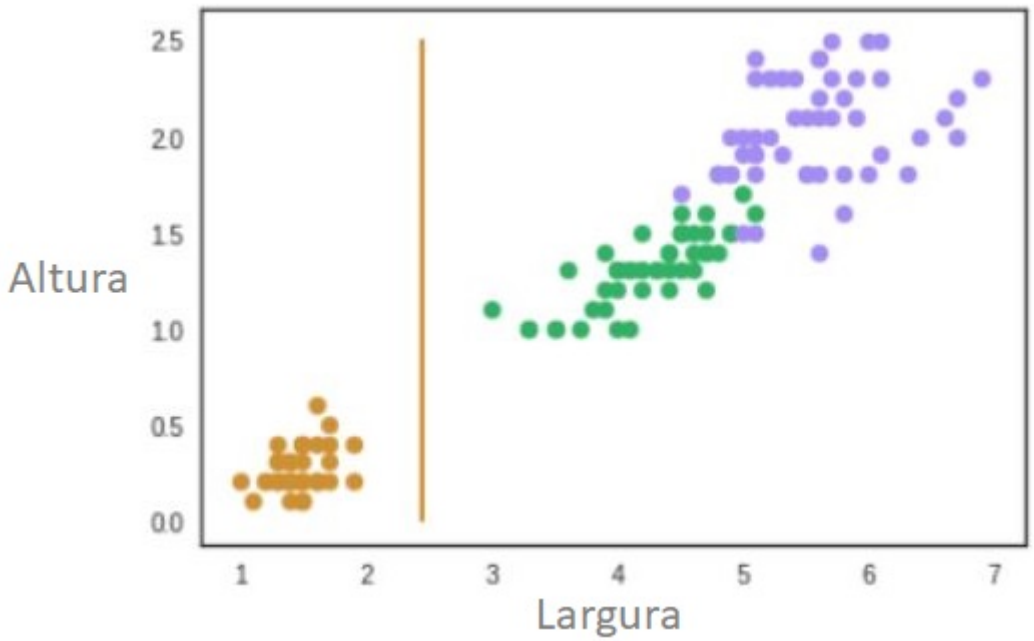

Figura 2 – Distribuição de Produtos

Vamos iniciar calculando $p(c|R)$ para cada região e produto.

Sabendo que temos 50 exemplos de cada produto $(Produto_A, Produto_B e Produto_C)$, a $p(c|R)$ para todos os produtos é $50/150$ 0.33.

Calculando para a sub-região da esquerda, temos que $p(c = Produto_A|R_e) = 1.0$ (só temos Produto_A nessa região) e $p(c = Produto_B|R_e) = p(c = Produto_C|R_e) = 0.0$.

Para direita,
$p(c = Produto_A|R_d) = 0.0ep(c = Produto_B|R_d) = p(c = Produto_C|R_d) = 0.5.$

Com esses valores em mãos, podemos calcular a entropia e a impureza $gini$ de cada região e, por consequência, obter o ganho de informação.

$$entropia(R) = -\sum p(c|R)log(p(c|R)) = -3 * (0.33log(0.33))\ 0.48$$

$$entropia(R_e) = -(1.0log(1.0) + 0.0log(0.0) + 0.0log(0.0)*) = 0$$

$$entropia(R_d) = -(0.0log(0.0) + 0.5log(0.5) + 0.5log(0.5)*)\ 0.30$$

Portanto, o ganho de informação usando entropia como critério de impureza é: InfoGain = 0.48 - (50*0 + 100*0.30) / 150 = 0.28

Calculando o $gini$ de maneira análoga,

$$gini(R) = \sum p(c|R)(1 - p(c|R)) = 3 * (0.33 * (1 - 0.33))\ 0.66$$

$$gini(R_e) = (1.0 * (1.0 - 1.0) + 0.0 * (1 - 0.0) + 0.0 * (1 - 0.0)) = 0$$

$$gini(R_d) = (0.0 * (1 - 0.0) + 0.5 * (1 - 0.5) + 0.5 * (1 - 0.5))\ 0.5$$

Portanto, o ganho de informação usando $gini$ como critério de impureza é: InfoGain = 0.66 - (50*0 + 100*0.50) / 150 = 0.16

Desta maneira, o ganho de informação considerando o critério de impureza entropia foi maior do que considerando o critério de impureza $gini$. Assim, o ponto de corte que retornou o maior ganho de informação e o melhor a ser considerado no exemplo acima seria a entropia, porque torna os ramos da árvore mais homogêneos.

Terminologias:

- O nó raiz é o atributo dentre os atributos existentes no conjunto de dados que melhor segrega os dados.
- Os nós de decisão são os responsáveis por definir os caminhos ou ramificações da árvore através das condições lógicas (*if-else*).
- Os nós folha estão sempre no final de cada ramificação da árvore e representam os resultados da classificação ou regressão, respectivamente, com uma classe ou um valor contínuo.

Floresta de Árvores Aleatórias

Podemos lidar com uma árvore de decisão simples com todas as características vistas acima ou podemos utilizar um conjunto (*ensemble*) de árvores. O algoritmo que aborda o conceito de métodos *ensemble* que veremos a seguir é um conjunto de árvores de decisão aleatórias conhecido como Random Forest.

O algoritmo floresta aleatória (*random forest*) cria diversas árvores de decisão de maneira aleatória, gerando o que podemos chamar de floresta. Ao final, cada uma das árvores que compõem a floresta é utilizada para determinar e escolher o melhor resultado.

Métodos *ensemble*

Para fornecer um melhor entendimento do algoritmo *random forest*, é necessário apresentar o conceito por trás de métodos *ensemble*, dos quais ele faz parte. Métodos *ensemble* tem como objetivo combinar diferentes modelos a fim de se obter um único resultado. Essa característica torna os algoritmos mais complexos e robustos, fazendo com que seja elevado o custo computacional, porém que costuma ser acompanhado de melhores resultados.

O que acontece é que ao invés de apenas escolhermos o algoritmo com melhor desempenho, podemos testar diferentes configurações do algoritmo escolhido, gerando diferentes modelos. Desta maneira, a geração de diversos modelos a partir de um algoritmo permite que métodos *ensemble* façam uso de todas a variações encontradas para compor o resultado.

Por exemplo, se criarmos 50 modelos, teremos 50 resultados que serão agregados em apenas um único resultado. Em abordagens de classificação, o resultado que mais se repete poderá ser o escolhido. Por outro lado, se estivermos lidando com um problema de regressão, a média dos valores gerados de cada uma das variações do algoritmo ou modelos produzidos permitirá a obtenção do resultado.

Existem casos em que o resultado de um modelo é utilizado para criação de um próximo, fazendo com que exista um encadeamento entre os modelos e levando a um único resultado gerado de vários resultados intermediários.

Muitos métodos *ensemble* dependem do conceito de árvores de decisão, sendo de grande valia o conhecimento deste conceito no aprendizado dos métodos.

Campos de Aplicação

Existem diversas aplicações práticas no uso de árvores de decisão. Abaixo compartilho alguns exemplos onde as árvores de decisão podem ser utilizadas:

- **Previsão de saída de funcionários:** Comumente utilizado no setor de RH das empresas, o algoritmo de árvore de decisão é utilizado para realizar uma predição se um funcionário vai sair ou não da empresa (*Churn*). O conjunto de regras utilizado pela árvore para determinar o resultado, pode direcionar quais são os atributos mais sensíveis para determinação dos resultados e ajudar a equipe de RH a trabalhar nos mesmos para reter e manter bons funcionários na empresa.

- **Análise de sentimentos:** No campo de procedimento de linguagem natural dentro da área de análise de sentimentos, o algoritmo de árvore de é utilizado para prever se um determinado texto deve ser classificado como positivo ou negativo.

- **Diagnóstico de doenças:** No diagnóstico de doenças, as árvores de decisão aprendem com os dados dos pacientes, entende suas relações e realiza os cálculos para entender quais são os nós mais importantes para determinar se um determinado nódulo (câncer) é benigno ou maligno. Desta maneira, o médico pode direcionar o paciente para o adequado tratamento.

- **Previsão de empréstimo:** Não menos importante do que nas demais áreas citadas acima, as árvores de decisão são utilizadas em sistemas financeiros para oferecer uma previsão de valor de empréstimo que pode ser aplicado a um determinado cliente. Este é um exemplo de regressão aplicada através de árvores de decisão. Ainda, o algoritmo pode classificar o cliente como bom ou ruim pagador diante da análise de dados históricos fazendo com que este cliente seja direcionado para áreas que possam ofertar novos serviços ou efetuar cobranças devidas.

Principais Vantagens e Desvantagens

Árvores de decisões são conceitualmente simples, porém poderosas. Sua popularidade é, principalmente, devido a suas características singulares ou vantagens:

- Fácil explicação e interpretação, já que podemos facilmente visualizá-las (quando não são muito profundas);
- Requerem pouco esforço na preparação dos dados, métodos baseados em árvores normalmente não requerem normalização dos dados. Além disso, conseguem lidar com valores faltantes, categóricos e numéricos;
- São capazes de lidar com problemas com múltiplos rótulos quando usada no contexto de classificação;
- São simples e eficientes;
- Não são muito sensitivas a *outliers*;
- Fornecem pontuação de probabilidade para observações.

Porém, como qualquer outro algoritmo, mesmo com as vantagens apresentadas acima, este algoritmo lida com alguns problemas ou desvantagens, tais como:

- Propenso a sofrer *overfitting* (sobreajuste). Caso os dados de treino sejam muito ajustados podemos não ter um bom desempenho com os dados de teste. Na prática o que acontece é que árvores com ramificações grandes e desenvolvidas até sua profundidade máxima podem decorar o conjunto de treino, gerando o *overfitting*. Isto produz a degradação do desempenho quando aplicado a novos dados. A solução de contorno para este problema ou diminuição do seu impacto é podar a árvore de decisão atribuindo uma profundidade máxima ou uma quantidade máxima de folhas.
- São modelos instáveis (alta variância), pequenas variações nos dados de treino podem resultar em árvores completamente distintas. Isso pode ser evitado ao treinarmos várias árvores distintas e agregar suas predições, como por exemplo, o uso do *Random Forest*.
- O algoritmo de construção da árvore de decisão não garante a construção da melhor estrutura para os dados de treino em questão.
- Não lida bem com muitas características/variáveis categóricas.
- Requer transformação de recursos não lineares.

Aplicação Prática

Através do link abaixo, pode ser feito o download do(s) script(s) python criado(s) para exemplificar uma abordagem do uso prático de árvores de decisão como também o(s) respectivo(s) conjunto(s) de dados utilizados.

Script Python
Dataset 1

```
### Árvores de Decisão

# Carregando as bibliotecas
import pandas as pd
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.tree import DecisionTreeClassifier

# Carregando o dataset
df_credito = pd.read_csv('credit.csv')
df_credito.shape

# Apresentando as 5 primeiras linhas do dataset
df_credito.head()

# Segregando as variáveis previsoras e classe
previsores = df_credito.iloc[:,0:20].values
classe = df_credito.iloc[:,20].values

# Realizando a conversão de atributos categóricos para numéricos de acordo
com o respectivo indice
labelencoder = LabelEncoder()
previsores[:,0] = labelencoder.fit_transform(previsores[:,0])
previsores[:,2] = labelencoder.fit_transform(previsores[:,2])
previsores[:, 3] = labelencoder.fit_transform(previsores[:, 3])
previsores[:, 5] = labelencoder.fit_transform(previsores[:, 5])
previsores[:, 6] = labelencoder.fit_transform(previsores[:, 6])
previsores[:, 8] = labelencoder.fit_transform(previsores[:, 8])
previsores[:, 9] = labelencoder.fit_transform(previsores[:, 9])
previsores[:, 11] = labelencoder.fit_transform(previsores[:, 11])
previsores[:, 13] = labelencoder.fit_transform(previsores[:, 13])
previsores[:, 14] = labelencoder.fit_transform(previsores[:, 14])
previsores[:, 16] = labelencoder.fit_transform(previsores[:, 16])
previsores[:, 18] = labelencoder.fit_transform(previsores[:, 18])
previsores[:, 19] = labelencoder.fit_transform(previsores[:, 19])

# Segregando os dados entre treinamento e teste
\\(X_treinamento\\), \\(X_teste, y_treinamento\\), \\(y_teste =
train_test_split(previsores,
                                                        classe,
                                                        test_size =
0.3,

random_state = 0)\\)

# Criação do modelo
arvore = DecisionTreeClassifier()

# Treinamento do modelo
```

```
arvore.fit(X_treinamento, y_treinamento)

# Obtendo as previsões
previsoes = arvore.predict(X_teste)
previsoes

# Confusion Matrix
confusao = confusion_matrix(y_teste, previsoes)
confusao

# Calculando a taxa de acerto
taxa_acerto = accuracy_score(y_teste, previsoes)
taxa_acerto

# Calculando a taxa de erro
taxa_erro = 1 - taxa_acerto
taxa_erro
```

Materiais complementares

- Let's Code channel at Youtube: O que é classificação e regressão em machine learning?
- SIMÕES, Adriana Carla Araújo. Mineração de Dados baseada em Árvores de Decisão para Análise do Perfil de Contribuintes. 2008. Dissertação de Mestrado. Universidade Federal de Pernambuco.
- OYA, Juliano Kazuki Matsuzaki. Utilização de árvores de decisão para aprimorar a classificação de fragmentos. 2016.
- C MARA, Renan; PAES, Aline; DE OLIVEIRA, Daniel. Aplicação de Árvores de Decisão para Recomendação de Parâmetros em Workflows Científicos. In: Anais do IX Brazilian e-Science Workshop. SBC, 2015. p. 11-20.

Referências

- SIMÕES, Adriana Carla Araújo. Mineração de Dados baseada em Árvores de Decisão para Análise do Perfil de Contribuintes. 2008. Dissertação de Mestrado. Universidade Federal de Pernambuco.
- OYA, Juliano Kazuki Matsuzaki. Utilização de árvores de decisão para aprimorar a classificação de fragmentos. 2016.
- C MARA, Renan; PAES, Aline; DE OLIVEIRA, Daniel. Aplicação de Árvores de Decisão para Recomendação de Parâmetros em Workflows Científicos. In: Anais do IX Brazilian e-Science Workshop. SBC, 2015. p. 11-20.

Tópico anterior

Próximo Tópico