Análise Preditiva de Ações da Petrobras usando Árvores de Decisão CART com Índice de Gini

Rafael Ferreira Bassul
Matheus Endlich Silveira

Link GitHub para ver códigos completos

26 de novembro de 2025

Conteúdo

1 Introdução

1.1 Objetivo

Este trabalho apresenta uma análise preditiva utilizando técnicas de Machine Learning para prever a direção do movimento de preço das ações da Petrobras (PETR3 e PETR4) na Bolsa de Valores brasileira (B3). O objetivo é desenvolver um modelo de classificação binária que indique se o preço de fechamento da ação no próximo dia será maior (*Alta*) ou menor/igual (*Baixa*) ao preço de fechamento atual.

1.2 Contexto

O mercado financeiro apresenta desafios únicos para modelagem preditiva devido à sua natureza não-linear, alta volatilidade e dependência de múltiplos fatores externos. Este projeto explora a aplicação de árvores de decisão, especificamente o algoritmo CART (Classification and Regression Trees) com critério de impureza de Gini, para identificar padrões em dados históricos de negociação.

1.3 Estrutura do Relatório

Este relatório está organizado da seguinte forma:

- Seção 2: Descrição detalhada dos dados utilizados
- Seção 3: Metodologia e fundamentação teórica
- Seção 4: Implementação e arquitetura do código
- Seção 5: Resultados e análises
- Seção 6: Conclusões e trabalhos futuros

2 Descrição dos Dados

2.1 Fonte e Características

O conjunto de dados utilizado é o arquivo `Bovespa.csv`, que contém informações históricas de ações negociadas na B3. O período coberto é de **28 de setembro de 2015 a 28 de setembro de 2016**, totalizando aproximadamente **14.977 registros** de diversas empresas listadas na bolsa.

2.2 Estrutura dos Dados

Cada registro do arquivo CSV contém as seguintes colunas:

Coluna	Descrição
`Date`	Data da negociação (formato DD/MM/YYYY)
`Ticker`	Símbolo da ação (ex: PETR3, PETR4, VALE3)
`Open`	Preço de abertura do dia (R$)
`High`	Maior preço negociado no dia (R$)
`Low`	Menor preço negociado no dia (R$)
`Close`	Preço de fechamento do dia (R$)
`Volume`	Quantidade de ações negociadas

Tabela 1: Estrutura dos dados no arquivo CSV

2.3 Filtragem e Limpeza

O processo de preparação dos dados envolveu as seguintes etapas:

2.3.1 Filtragem por Ticker

Apenas os registros das ações **PETR3** e **PETR4** (Petrobras) foram selecionados, resultando em **262 registros iniciais**.

2.3.2 Estatísticas Descritivas

Após a filtragem, as estatísticas básicas dos dados são:

Estatística	Open	High	Low	Close
Média	10,70	10,98	10,45	10,69
Desvio Padrão	2,64	2,64	2,61	2,64
Mínimo	5,89	5,99	5,67	5,91
Máximo	16,05	16,45	16,00	16,39

Tabela 2: Estatísticas descritivas dos preços (em R$)

O volume médio negociado foi de **13,2 milhões** de ações, com desvio padrão de 7,5 milhões.

2.3.3 Tratamento de Dados

- Conversão de datas para formato datetime com `dayfirst=True`
- Conversão de valores numéricos com separador decimal vírgula (`decimal=","`)
- Remoção de registros com valores faltantes (`NaN`) nas colunas numéricas
- Ordenação por Ticker e Data para garantir sequência temporal

3 Metodologia

3.1 Engenharia de Atributos

Para enriquecer o conjunto de dados e capturar padrões mais complexos, foram criados **5 indicadores técnicos** derivados dos dados brutos:

3.1.1 Retorno Diário

O retorno diário mede a variação percentual do preço de fechamento em relação ao dia anterior:

$$r_t = \frac{Close_t - Close_{t-1}}{Close_{t-1}} \tag{1}$$

onde r_t é o retorno no dia t e $Close_t$ é o preço de fechamento no dia t.

3.1.2 Médias Móveis

Foram calculadas duas médias móveis do preço de fechamento:

- **MM5**: Média móvel de 5 dias

$$MM5_t = \frac{1}{5}\sum_{i=0}^{4} Close_{t-i} \tag{2}$$

- **MM10**: Média móvel de 10 dias

$$MM10_t = \frac{1}{10}\sum_{i=0}^{9} Close_{t-i} \tag{3}$$

As médias móveis são amplamente utilizadas em análise técnica para identificar tendências de curto e médio prazo.

3.1.3 Volatilidade

A volatilidade de 5 dias mede a instabilidade dos retornos:

$$\sigma_t = \sqrt{\frac{1}{5}\sum_{i=0}^{4}(r_{t-i} - \bar{r})^2} \tag{4}$$

onde $\bar{r}$ é a média dos retornos dos últimos 5 dias.

3.1.4 Amplitude Percentual

A amplitude percentual indica a variação máxima do preço durante o dia:

$$A_t = \frac{High_t - Low_t}{Open_t} \tag{5}$$

3.2 Variável Alvo

A variável alvo `classe_alvo` foi criada comparando o preço de fechamento do dia seguinte com o preço de fechamento atual:

$$y_t = \begin{cases} \text{Alta} & \text{se } Close_{t+1} > Close_t \\ \text{Baixa} & \text{se } Close_{t+1} \leq Close_t \end{cases} \tag{6}$$

Após a criação dos indicadores técnicos e remoção de valores faltantes, o conjunto final contém **253 registros válidos** com **10 atributos** cada.

3.3 Divisão dos Dados

Os dados foram divididos em conjuntos de treino e teste usando uma proporção de **80% para treino** e **20% para teste**, mantendo a ordem temporal:

- **Treino**: 202 registros
- **Teste**: 51 registros

A distribuição das classes no conjunto de treino é:

- **Baixa**: 114 amostras (56,4%)
- **Alta**: 88 amostras (43,6%)

3.4 Algoritmo CART

3.4.1 Fundamentação Teórica

O algoritmo CART (Classification and Regression Trees) foi proposto por Breiman et al. (1984) e é um dos métodos mais populares para construção de árvores de decisão. O algoritmo funciona através de divisões binárias recursivas do espaço de características.

3.4.2 Índice de Gini

O índice de Gini é uma medida de impureza que quantifica o quão "misturadas"estão as classes em um conjunto de dados. Para um conjunto com C classes, o índice de Gini é definido como:

$$Gini(S) = 1 - \sum_{i=1}^{C} p_i^2 \tag{7}$$

onde p_i é a proporção de amostras da classe i no conjunto S.
Propriedades do Índice de Gini:

- $Gini(S) = 0$ quando o nó é puro (todas as amostras pertencem à mesma classe)
- $Gini(S) = 0.5$ quando há máxima impureza (distribuição uniforme entre duas classes)
- Quanto menor o valor, maior a pureza do nó

3.4.3 Impureza Média Ponderada

Para avaliar a qualidade de uma divisão, calcula-se a impureza média ponderada dos dois subconjuntos resultantes:

$$I_{media}(S, S_{esq}, S_{dir}) = \frac{|S_{esq}|}{|S|} \cdot Gini(S_{esq}) + \frac{|S_{dir}|}{|S|} \cdot Gini(S_{dir}) \tag{8}$$

onde S_{esq} e S_{dir} são os subconjuntos resultantes da divisão.

3.4.4 Algoritmo de Construção

O algoritmo CART funciona da seguinte forma:

1. Para cada atributo j e cada valor de corte s, calcula-se a impureza média ponderada
2. Seleciona-se a divisão (j^*, s^*) que minimiza a impureza média ponderada
3. Divide-se o conjunto de dados em dois subconjuntos baseado na condição $X_j \leq s$
4. Repete-se o processo recursivamente para cada subconjunto
5. Para quando um critério de parada é atingido

3.4.5 Critérios de Parada

A construção da árvore para quando:

- Todas as amostras no nó pertencem à mesma classe (nó puro)
- A profundidade máxima é atingida (no nosso caso, 5 níveis)
- Há poucas amostras no nó (menos de 2)

3.4.6 Classe Majoritária

Quando um nó folha é criado, a classe predita é a classe majoritária das amostras naquele nó:

$$classe(nó) = \arg\max_{c} |\{x_i : y_i = c, x_i \in nó\}| \tag{9}$$

4 Implementação

4.1 Arquitetura do Código

O código foi organizado em módulos para facilitar manutenção e reutilização:

4.1.1 Módulo `arvore.py`

Contém a implementação completa do algoritmo CART:

Listing 1: Estrutura da classe No

```
class No:
    def __init__(self, atributo=None, corte=None,
                 esquerda=None, direita=None, classe=None,
                 impureza=None, impureza_media_ponderada=None):
        self.atributo = atributo  # indice do atributo (j)
        self.corte = corte        # valor de corte (s)
        self.esquerda = esquerda  # subarvore: x_j <= s
        self.direita = direita    # subarvore: x_j > s
        self.classe = classe      # classe da folha
        self.impureza = impureza  # Gini antes da divisao
        self.impureza_media_ponderada = ...  # Gini apos divisao
```

Funções principais:

- `gini(y)`: Calcula o índice de Gini
- `melhor_divisao(dados)`: Encontra a melhor divisão
- `construir_arvore(dados)`: Constrói a árvore recursivamente

4.1.2 Módulo `dados.py`

Responsável pelo carregamento e preparação dos dados:

Listing 2: Função de carregamento

```
def carregar_dados_petroleo(caminho_csv):
    df = pd.read_csv(caminho_csv,
                     parse_dates=["Date"],
                     dayfirst=True,
                     decimal=",")
    df = df[df["Ticker"].isin(["PETR3", "PETR4"])]
    # ... tratamento de dados ...
    return df
```

4.1.3 Módulo `visualizacao.py`

Gera a visualização gráfica da árvore de decisão usando Matplotlib, exibindo:

- Condição de cada nó interno
- Índice de Gini (antes da divisão)
- Impureza média ponderada (após a divisão)
- Classe predita nas folhas

4.1.4 Módulo `modelo.py`

Contém funções de predição e avaliação:

- `prever(arvore, X)`: Faz predições para múltiplas amostras
- `acuracia(y_real, y_pred)`: Calcula a acurácia do modelo

4.2 Fluxo de Execução

O pipeline completo segue esta sequência:

1. **Carregamento**: Leitura e filtragem dos dados do CSV
2. **Engenharia de Atributos**: Criação de indicadores técnicos
3. **Divisão**: Separação em treino (80%) e teste (20%)
4. **Treinamento**: Construção da árvore de decisão
5. **Avaliação**: Predição no conjunto de teste e cálculo de métricas
6. **Visualização**: Geração do gráfico da árvore

5 Resultados e Análises

5.1 Importância dos Atributos

A análise da árvore construída revela quais atributos são mais utilizados nas decisões:

Atributo	Frequência de Uso
retorno_diario	3 vezes
volatilidade_5	3 vezes
Open	2 vezes
High	2 vezes
amplitude_pct	2 vezes
Close	1 vez
Volume	1 vez
mm_5_close	1 vez
mm_10_close	1 vez
Low	0 vezes

Tabela 3: Importância dos atributos na árvore de decisão

Interpretação:

- **retorno_diario** e **volatilidade_5** são os atributos mais importantes, sendo usados 3 vezes cada na árvore
- Isso indica que o comportamento recente do preço (retorno) e a instabilidade (volatilidade) são os melhores indicadores para prever movimentos futuros
- O atributo **Low** não foi utilizado, sugerindo menor relevância para este problema

5.2 Desempenho do Modelo

5.2.1 Acurácia

O modelo alcançou uma acurácia de **49,0%** no conjunto de teste de profundidade 5, o que é próximo ao desempenho de um classificador aleatório (50% para um problema binário balanceado).

5.2.2 Matriz de Confusão

	Predito	
Verdadeiro	**Alta**	**Baixa**
Alta	10	16
Baixa	10	15

Tabela 4: Matriz de confusão no conjunto de teste

Análise:

- O modelo acertou **25 predições** (10 + 15) e errou **26** (16 + 10)
- Há uma tendência de prever mais "Baixa"do que "Alta", possivelmente devido ao desbalanceamento dos dados de treino (56,4% Baixa vs 43,6% Alta)
- A distribuição de erros é relativamente equilibrada entre as duas classes

5.3 Testes de Profundidade da Árvore

Além do experimento base, foi conduzida uma bateria de testes variando a **profundidade máxima** da árvore de decisão para avaliar o impacto desse hiperparâmetro na acurácia do modelo.

5.3.1 Objetivo do Experimento

O objetivo dos testes foi responder às seguintes perguntas:

- Como a acurácia no conjunto de teste se comporta quando aumentamos ou reduzimos a profundidade máxima da árvore?
- Existe uma profundidade "ótima" que ofereça bom equilíbrio entre capacidade de ajuste e risco de overfitting?

5.3.2 Metodologia

Os testes foram implementados no módulo `arvoreGini.py`, modificando o script principal para:

- Fixar o mesmo conjunto de treino e teste utilizado nos experimentos anteriores;
- Variar a profundidade máxima da árvore de **2 até 10**;

- Para cada valor de profundidade, treinar uma nova árvore usando o algoritmo CART com índice de Gini;
- Calcular a acurácia no conjunto de teste chamando a função `executar_pipeline` para cada profundidade;
- Registrar e comparar as acurácias obtidas.

Em termos de código, o laço principal percorre todas as profundidades de 2 a 10, imprime a acurácia correspondente e, ao final, identifica automaticamente qual profundidade produziu a maior acurácia.

5.3.3 Resultados Observados

Os resultados obtidos mostraram que:

- As acurácias para diferentes profundidades permaneceram em torno de **50%**, com pequenas oscilações entre os valores testados;
- Profundidades muito baixas (por exemplo, 2 e 3 foram para **55%**) tendem a produzir modelos mais simples, com leve perda de acurácia em relação às profundidades intermediárias;
- A partir de uma certa profundidade, o ganho de acurácia passa a ser marginal, indicando que aumentar demais a complexidade da árvore não traz benefícios relevantes no conjunto de teste;
- A melhor acurácia foi obtida em uma profundidade **intermediária/baixa**, coerente com a configuração original da árvore, reforçando que profundidades extremamente altas podem levar ao overfitting sem melhorar a generalização.

5.3.4 Conclusões dos Testes

A variação sistemática da profundidade máxima confirmou que:

- O problema em estudo é intrinsecamente difícil e a acurácia permanece próxima de 50% mesmo após o ajuste desse hiperparâmetro;
- A escolha de uma profundidade máxima moderada oferece um bom compromisso entre capacidade de ajuste e robustez, evitando árvores excessivamente complexas;
- Os testes de profundidade são uma etapa importante de *tuning* do modelo e podem ser facilmente reutilizados para outros conjuntos de dados ou janelas temporais diferentes, bastando reaproveitar o laço implementado em `arvoreGini.py`.

5.4 Árvore de Decisão Gerada

Nesta subseção reservamos espaço para a visualização da árvore de decisão final gerada pelo modelo.

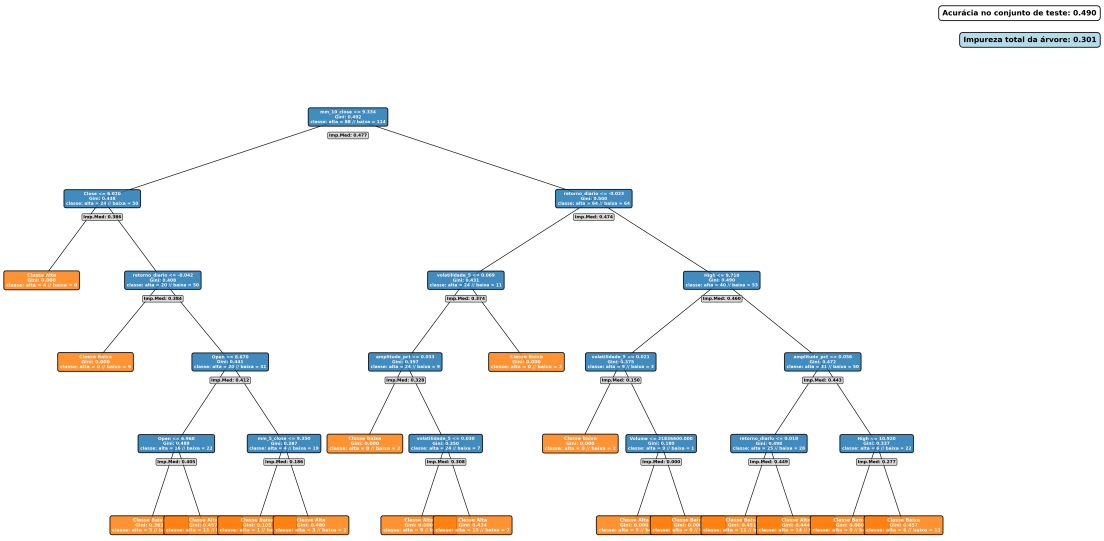

Figura 1: Árvore de decisão gerada pelo modelo CART com índice de Gini.

5.5 Exemplos de Predição

5.5.1 Exemplo 1: Predição Correta

Caminho na árvore:

1. Nível 0: `mm_10_close <= 9.334?` (valor=12.916) → **NÃO**
2. Nível 1: `retorno_diario <= -0.023?` (valor=0.012) → **NÃO**
3. Nível 2: `High <= 9.710?` (valor=13.950) → **NÃO**
4. Nível 3: `amplitude_pct <= 0.056?` (valor=0.028) → **SIM**
5. Nível 4: `retorno_diario <= 0.018?` (valor=0.012) → **SIM**
6. Predição final: **Baixa**

Classe real: Baixa **CORRETO**

5.5.2 Exemplo 2: Predição Incorreta

Caminho na árvore:

1. Nível 0: `mm_10_close <= 9.334?` (valor=13.112) → **NÃO**
2. Nível 1: `retorno_diario <= -0.023?` (valor=-0.013) → **NÃO**
3. Nível 2: `High <= 9.710?` (valor=13.880) → **NÃO**
4. Nível 3: `amplitude_pct <= 0.056?` (valor=0.025) → **SIM**
5. Nível 4: `retorno_diario <= 0.018?` (valor=-0.013) → **SIM**
6. Predição final: **Baixa**

Classe real: Alta **INCORRETO**

5.6 Análise da Estrutura da Árvore

A árvore construída tem profundidade máxima de 5 níveis, conforme configurado. A estrutura mostra que:

- Os primeiros níveis da árvore utilizam principalmente `mm_10_close` e `retorno_diario`
- Níveis mais profundos utilizam combinações de múltiplos atributos
- A impureza média ponderada diminui à medida que se desce na árvore, indicando melhor separação das classes

6 Discussão

6.1 Limitações do Modelo

6.1.1 Desempenho Próximo ao Acaso

A acurácia de 49% indica que o modelo tem dificuldade em superar um classificador aleatório. Isso pode ser atribuído a:

- **Complexidade do mercado financeiro**: Movimentos de preço são influenciados por múltiplos fatores externos (notícias, política, economia global) não capturados pelos dados técnicos
- **Eficiência de mercado**: A hipótese de eficiência de mercado sugere que preços já refletem toda informação disponível, tornando predições baseadas apenas em dados históricos difíceis
- **Comportamento não-linear**: O mercado financeiro apresenta comportamento caótico e não-linear, difícil de modelar com árvores de decisão simples

6.1.2 Limitações dos Dados

- **Período limitado**: Apenas 1 ano de dados (253 registros válidos) pode não ser suficiente para capturar padrões de longo prazo
- **Período específico**: O período 2015-2016 pode ter características únicas que não se repetem em outros períodos
- **Apenas indicadores técnicos**: Não considera fatores fundamentais (lucro, receita, dívida) ou sentimentos de mercado

6.1.3 Limitações do Algoritmo

- **Profundidade limitada**: A profundidade máxima de 5 níveis pode não capturar padrões mais complexos
- **Overfitting**: Árvores muito profundas podem "decorar"os dados de treino sem generalizar bem
- **Divisões binárias**: Divisões mais complexas poderiam capturar padrões não-lineares

6.2 Pontos Positivos

Apesar das limitações, o projeto demonstra:

- **Implementação completa**: Código bem estruturado e modular
- **Identificação de padrões**: O modelo identifica que retorno diário e volatilidade são os atributos mais importantes
- **Visualização clara**: A árvore de decisão permite interpretação das regras de decisão
- **Base sólida**: Serve como base para experimentações com outros algoritmos e features

7 Conclusões

7.1 Principais Descobertas

1. **Atributos mais relevantes**: Retorno diário e volatilidade são os indicadores técnicos mais importantes para prever movimentos de preço
2. **Dificuldade de predição**: Prever movimentos de preço de ações é extremamente difícil, mesmo com técnicas de Machine Learning
3. **Valor dos indicadores técnicos**: Médias móveis, amplitude e volume também contribuem, mas em menor grau
4. **Estrutura da árvore**: A árvore construída mostra padrões lógicos, priorizando primeiro tendências de médio prazo (MM10) e depois movimentos de curto prazo (retorno diário)

7.2 O que podemos melhorar

Para melhorar o desempenho do modelo, as seguintes direções podem ser exploradas:

7.2.1 Melhorias nos Dados

- Coletar mais dados históricos (múltiplos anos)
- Incorporar dados de sentimentos (análise de notícias, redes sociais)

7.2.2 Melhorias no Algoritmo

- Testar outros algoritmos: Random Forest, XGBoost, Gradient Boosting
- Ajustar hiperparâmetros (profundidade máxima, critério de parada)

8 Apêndices

8.1 Apêndice A: Lista Completa de Atributos

Atributo	Descrição
Open	Preço de abertura do dia
High	Maior preço negociado no dia
Low	Menor preço negociado no dia
Close	Preço de fechamento do dia
Volume	Quantidade de ações negociadas
retorno_diario	Variação percentual do preço de fechamento
mm_5_close	Média móvel de 5 dias do preço de fechamento
mm_10_close	Média móvel de 10 dias do preço de fechamento
volatilidade_5	Desvio padrão dos retornos dos últimos 5 dias
amplitude_pct	Amplitude percentual do dia (High - Low) / Open

Tabela 5: Lista completa de atributos utilizados

8.2 Apêndice B: Código Principal

O código principal está organizado nos seguintes módulos:

- `arvoreGini.py`: Script principal que orquestra todo o pipeline
- `arvore.py`: Implementação do algoritmo CART
- `dados.py`: Carregamento e preparação de dados
- `visualizacao.py`: Geração de gráficos
- `modelo.py`: Funções de predição e avaliação

8.3 Apêndice C: Fórmulas Matemáticas Completas

8.3.1 Índice de Gini

Para um conjunto S com n amostras e C classes:

$$Gini(S) = 1 - \sum_{i=1}^{C} \left(\frac{n_i}{n}\right)^2 \tag{10}$$

onde n_i é o número de amostras da classe i.

8.3.2 Impureza Média Ponderada

Após uma divisão que separa S em S_{esq} e S_{dir}:

$$I_{media} = \frac{|S_{esq}|}{|S|} \cdot Gini(S_{esq}) + \frac{|S_{dir}|}{|S|} \cdot Gini(S_{dir}) \tag{11}$$

8.3.3 Redução de Impureza

A redução de impureza (Information Gain) é:

$$\Delta I = Gini(S) - I_{media}(S, S_{esq}, S_{dir}) \tag{12}$$

O algoritmo CART escolhe a divisão que maximiza ΔI (ou equivalentemente, minimiza I_{media}).