Universidade Estadual Paulista – UNESP
Faculdade de Ciências e Tecnologia

Aplicação de Modelos de Machine Learning para Classificação de Tendências e Apoio à Decisão de Compra e Venda de Ações nos Mercados Brasileiro e Americano.

Pedro Henrique Milani Vagula

Presidente Prudente – SP – Brasil
Setembro de 2025

Resumo

A previsão de séries temporais financeiras é um problema complexo e não-linear. Este trabalho investiga e compara o desempenho de quatro modelos de Machine Learning — Random Forest , XGBoost , Support Vector Machines (SVMs) e Redes Neurais Profundas (MLP) — na tarefa de classificação de tendências de ações. O estudo foca em ativos dos mercados brasileiro (Ibovespa) e americano (SeP 500) , classificando-os em 'Compra', 'Venda' ou 'Espera' . A metodologia emprega uma rigorosa rotulagem de 'barreira tripla' para filtrar ruídos do mercado, utilizando indicadores técnicos e macroeconômicos como features . O treinamento respeita a ordem cronológica dos dados , evitando data leakage. A avaliação de desempenho é dupla, combinando métricas estatísticas (F1-Score, Precisão) com um backtest financeiro comparado ao benchmark de Buy and Hold . Espera-se que os modelos não-lineares demonstrem desempenho superior e que a análise de interpretabilidade (SHAP) identifique os preditores de maior relevância.

Contents

1 Introdução

O Machine Learning, também conhecido como Aprendizado de Máquina, é uma área enraizada na Ciência da Computação e na Estatística, focada principalmente no reconhecimento de padrões em conjuntos de dados. Esse foco molda várias metodologias para lidar com esses dados, conhecidas como Modelos, que variam de modelos de classificação a modelos de regressão (GOODFELLOW; BENGIO; COURVILLE, 2016). Esses modelos visam abordar a classificação de classes e a previsão numérica, respectivamente. Dada a sua ampla gama de aplicações, o Aprendizado de Máquina é atualmente predominante em vários campos, incluindo cuidados de saúde para diagnóstico de doenças, comércio para previsões de vendas sazonais, pesquisa académica e, particularmente, na sua aplicação no mercado financeiro (PRADO, 2018).

Tais Modelos frequentemente vem acompanhados de necessidades de tratamento desses mesmos dados, chamados "Features", ou Características. Essas Características possuem diferentes níveis de relevância, variando de problema para problema a ser resolvido, e cabe ao projetista, antes do modelo entrar em estágio de produção, estudar as Características a serem usadas no mesmo, selecionando as mais úteis por meio de testes, parâmetros numéricos e uma dose significativa de senso comum. Esse processo é chamado de Feature Engineering (FE) e é parte fundamental da produção básica de um modelo de AM eficiente (PRADO, 2018), vide que qualquer modelo tem sua utilidade baseada em seus dados, e dados curados e bem estruturados muitas vezes são a diferença entre a possibilidade do modelo se adaptar ao "mundo real", ou não.

A aplicação da FE muitas vezes leva à necessidade do processo de "Labeling" (PRADO, 2018), onde os dados são, automaticamente ou manualmente, atribuídos a diferentes "Labels", ou "Etiquetas". Essas Etiquetas tem como objetivo informar, como verdade fundamental, o que representa aquele dado. Quando todos os elementos de um Modelo tem suas respectivas Etiquetas, essa classe leva o nome de Aprendizado Supervisionado (GOODFELLOW; BENGIO; COURVILLE, 2016), que visa classificar com corretude máxima os dados quanto as suas etiquetas. Quando os dados não possuem Labels, damos o nome de Aprendizado Não-Supervisionado, onde o modelo "deduz" características similares de cada uma das entradas dos dados e assim os agrupa com base nas mesmas.

Outra distinção importante a ser feita é a de modelos Caixa Branca e Caixa Preta e suas diferenças práticas fundamentais (GOODFELLOW; BENGIO; COURVILLE, 2016). Em modelos Caixa Branca (White Box), são caracterizados pela sua transparencia completa, permitindo ao desenvolvedor entender como os dados de entrada são transformados em seus respectivos dados de saída, entendendo claramente a lógica por trás dessas predições. Já nos modelos Caixa Preta (Black Box), a lógica interna é opaca a visão do desenvolvedor, tornando o processo como um todo considerávelmente mais complexo e difícil de interpretar, mas entregando resultados surpreendentes, como Redes Neurais (Neural Networks) e Redes Neurais Profundas (Deep Neural Networks).

Diante desse panorama, a presente proposta de projeto de pesquisa visa aplicar essas e outras classes de modelos de Aprendizado de Máquina, usando como base de dados o ambiente das bolsas americanas e brasileiras, em uma escala reduzida, com menor número de ações a serem exploradas, e assim entender quais modelos melhor performam na classificação dessas ações, com período de Venda, Compra ou Espera, aplicando diferentes parâmetros, características e modelos para tal.

2 Objetivos

O presente projeto tem como meta primária a aplicação de metodologias e diversos modelos de algoritmos no domínio do Aprendizado de Máquina, com ênfase na utilização de modelos direcionados à classificação de dados, com o intuito de analisar o desempenho na classificação de ações das bolsas de valores brasileira e americana, organizando-as nas seguintes categorias: Venda, Compra e On-Hold, com o objetivo de maximizar lucros e minimizar perdas no ambiente. Para atingir essa meta, foram definidos os seguintes objetivos específicos:

1. Através da utilização de diversos métodos de aprendizado de máquina, efetuar o treinamento destes, para que consigam prever com uma precisão considerável qual a ação mais apropriada em um determinado momento: Compra, venda ou "Espera".

2. Utilizando técnicas de Feature Engineering e a análise do domínio, determinar quais são as melhores características a serem empregadas no treinamento desses modelos, com o objetivo de otimizar as taxas de sucesso dos mesmos.

3. Investigar a habilidade dos modelos em realizar uma classificação precisa das ações a serem adotadas, identificando quais deles operam de forma mais eficiente e as razões para tal desempenho.

Assim, o projeto visa contribuir para a pesquisa sobre a aplicação de algoritmos de Aprendizado de Máquina no contexto da compra e venda de ações, maximizando a performance nas bolsas de valores americana e brasileira.

3 Justificativa

Nos últimos anos, o campo do Aprendizado de Máquina tem experimentado avanços notáveis, impulsionados em grande parte pela aplicação e combinação de técnicas de engenharia de atributos e construção de modelos. Destacam-se, neste contexto, uma gama de modelos a serem testados, como o XGBoost, Random Forest, e um foco especial em Redes Neurais, todos introduzidos no domínio do AM pelo proponente desta pesquisa. Essas técnicas, até então demonstraram um potencial significativo para testar os limites do que é possível alcançar em termos de desempenho e precisão dos modelos de AM no âmbito das Bolsas de Valores.

Neste cenário, a proposta de pesquisa apresentada visa aplicar tais modelos e técnicas de modelagem, treiná-los e estudar seus resultados. O projeto se fundamenta na premissa de que a integração desses métodos pode ajudar a entender com um pouco mais de clareza o comportamento das ações e as diferentes aplicações decorrentes das diferenças proporcionadas pelas bolsas Americana e Brasileira.

A pesquisa será estruturada em duas frentes principais:

- **Modelagem e testes:** Serão moldados cada um dos modelos de Aprendizado de Máquina, por diferentes métodos, além da estruturação dos dados necessários para aplicação nos mesmos. Diferentes testes serão feitos e os resultados armazenados.

- **Análise de Resultados:** Com os resultados armazenados será possível então analisá-los, atentando-se aos detalhes e diferenças percptíveis entre cada modelo. Essa análise busca entender suas diferenças e características dos mesmos para com cada uma das duas Bolsas de valores.

Em suma, o desenvolvimento deste projeto busca não apenas comparar o desempenho dos modelos de aprendizado de máquina em diferentes contextos de mercado, mas também compreender de forma mais profunda o comportamento das ações sob distintas condições econômicas. Espera-se que, ao final da pesquisa, seja possível identificar quais modelos apresentam maior robustez e capacidade de generalização, bem como compreender quais características dos dados mais influenciam suas decisões. Além disso, a análise comparativa entre os mercados brasileiro e americano poderá evidenciar diferenças estruturais relevantes, contribuindo tanto para o aprimoramento de estratégias automatizadas de investimento quanto para a consolidação do Aprendizado de Máquina como uma ferramenta prática e confiável no apoio à tomada de decisão financeira.

4 Referêncial Teórico

Esta seção apresenta o referencial teórico pertinente ao projeto de pesquisa. Para assegurar uma organização adequada, clareza e facilitar a compreensão, esta seção está dividida entre cada um dos modelos a serem abordados neste projeto:

4.1 XGBoost

4.1.1 Definições

Ensemble Learning (Aprendizado de Conjunto): O Aprendizado de Conjunto, ou Ensemble Learning, é uma abordagem em aprendizado de máquina na qual múltiplos modelos, frequentemente chamados de "aprendizes fracos" (weak learners), são estrategicamente combinados para formar um único modelo preditivo mais robusto e preciso, conhecido como "aprendiz forte" (strong learner). A premissa fundamental é que a sabedoria coletiva de diversos modelos pode superar o desempenho de qualquer modelo individual, compensando seus erros e vieses. Ao agregar as "opiniões" de vários algoritmos — seja por meio de votação para tarefas de classificação ou pela média para regressão — os métodos de conjunto visam reduzir a variância (como no Bagging e Random Forest) (BREIMAN, 2001) ou o viés (como no Boosting), resultando em uma maior capacidade de generalização para dados não vistos.

Boosting: Boosting é uma família de algoritmos de aprendizado de conjunto que opera de forma sequencial e iterativa para converter um conjunto de aprendizes fracos em um único aprendiz forte. Diferente de métodos que constroem modelos em paralelo, a abordagem do Boosting é aditiva: cada novo modelo é treinado com o objetivo específico de corrigir os erros cometidos pelo conjunto de modelos anteriores. As instâncias que foram classificadas incorretamente na iteração anterior recebem um peso maior na iteração seguinte, forçando o novo modelo a focar nos casos mais difíceis. Esse processo continua até que um critério de parada seja atingido, resultando em um modelo final ponderado que agrega as previsões de toda a sequência, com maior ênfase nos modelos que demonstraram melhor desempenho (GOODFELLOW; BENGIO; COURVILLE, 2016).

Gradient Boosting: O Gradient Boosting é uma generalização poderosa e flexível do conceito de Boosting, que o eleva a um problema de otimização matemática. Em vez

de simplesmente ajustar os pesos das instâncias mal classificadas, esta técnica constrói os novos modelos sequenciais para prever os "pseudo-resíduos" do modelo anterior, que representam o gradiente (a direção de maior inclinação do erro) de uma função de perda (loss function) pré-definida. Ao ajustar cada novo aprendiz para minimizar o erro na direção apontada pelo gradiente, o algoritmo desce iterativamente em direção ao mínimo da função de perda, de forma análoga ao método de otimização gradient descent. Essa abordagem permite que o Gradient Boosting seja aplicado a uma vasta gama de funções de perda personalizáveis, tornando-o extremamente eficaz para tarefas de regressão, classificação e ranqueamento (CHEN; GUESTRIN, 2016).

4.1.2 Processo de divisão

A construção de cada árvore de decisão dentro do XGBoost se baseia em um processo de divisão binária e recursiva. Partindo do nó raiz, que contém todas as amostras de treinamento, o algoritmo busca determinar qual característica (feature) e qual valor de limiar (threshold) servem como o melhor critério de divisão para separar os dados em dois subconjuntos (os nós filhos). O "melhor" critério é aquele que maximiza a separação dos resíduos, ou seja, que melhor agrupa amostras com erros similares. Para quantificar a qualidade de uma potencial divisão, o XGBoost não utiliza métricas tradicionais como o Índice Gini ou a Entropia, mas sim uma métrica de otimização própria, chamada de Ganho (Gain), que está diretamente ligada à função de perda do modelo (CHEN; GUESTRIN, 2016).

Para calcular o Ganho de uma divisão, é preciso primeiro entender o conceito de **Pontuação de Similaridade (Similarity Score)**. Essa pontuação é calculada para qualquer conjunto de amostras que reside em um nó (seja ele pai ou filho) e mede a "pureza" dos pseudo-resíduos dentro daquele nó. Um valor alto indica que os resíduos são homogêneos, o que facilita a determinação de um valor de saída ótimo para aquela folha. A fórmula para o *Similarity Score* de um nó é definida como:

$$\text{Score}(I) = \frac{1}{2} \cdot \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda}$$

Onde I é o conjunto de amostras no nó, g_i é o gradiente (o pseudo-resíduo da amostra i), h_i é a Hessiana (a segunda derivada da função de perda), e λ (lambda) é o parâmetro de regularização L2, que penaliza a complexidade do modelo para evitar o sobreajuste (*overfitting*).

Cover: A Medida de Peso de um Nó

Dentro da fórmula do *Similarity Score*, o denominador desempenha um papel fundamental e recebe um nome específico: **Cover**. O *Cover* é a soma das Hessianas ($\sum_{i \in I} h_i$) de todas as amostras presentes em um nó. Intuitivamente, a Hessiana representa a curvatura da função de perda e, neste contexto, o *Cover* pode ser interpretado como uma medida do "peso" ou da "confiança" das previsões para aquele nó. Um nó com um *Cover* muito baixo contém poucas amostras ou amostras cujas previsões já são muito assertivas. O XGBoost utiliza o *Cover* como um critério de parada (através do hiperparâmetro `min_child_weight`), prevenindo a criação de divisões baseadas em um número insuficiente de exemplos e tornando o modelo mais robusto.

Gain: Quantificando o Benefício da Divisão

Com o *Similarity Score* definido, o **Ganho (Gain)** de uma divisão pode ser finalmente calculado. O Ganho mede o quanto a qualidade dos nós melhora após a realização de uma divisão, em comparação com a situação anterior (o nó pai). A sua fórmula é simples e elegante:

$$\text{Gain} = \text{Score}(I_L) + \text{Score}(I_R) - \text{Score}(I) - \gamma$$

Em essência, o algoritmo calcula o *Similarity Score* do nó pai e o subtrai da soma dos *Similarity Scores* dos dois nós filhos resultantes da divisão. O XGBoost testa exaustivamente todas as características e todos os possíveis valores de limiar, calculando o *Gain* para cada uma dessas possibilidades. A divisão que apresentar o maior valor de *Gain* será a escolhida, pois ela representa a separação que mais contribui para a redução da função de perda (erro) geral do modelo (CHEN; GUESTRIN, 2016).

4.1.3 As Otimizações do "Extreme" Gradient Boosting

Uma das características que diferenciam o XGBoost de implementações tradicionais de *Gradient Boosting* é a sua abordagem nativa para combater o sobreajuste (*overfitting*). Isso é alcançado através da **regularização**, uma técnica que penaliza a complexidade do modelo diretamente em sua função objetivo. O XGBoost emprega duas formas de regularização:

1. **Regularização L1 (Alpha):** Esta técnica adiciona uma penalidade proporcional ao valor absoluto dos pesos das folhas. Em modelos de árvore, a regularização L1 tem o efeito de encorajar a esparsidade, o que pode levar a um número menor de folhas ativas no modelo final. Essencialmente, ela pode zerar o peso de folhas que contribuem pouco, realizando uma forma implícita de seleção de características e simplificando a árvore.

2. **Regularização L2 (Lambda):** Também conhecida como regularização de Ridge, esta técnica adiciona uma penalidade proporcional ao quadrado dos pesos das folhas. Conforme visto na fórmula do *Similarity Score*, o parâmetro λ aumenta o denominador, o que diminui o valor final dos pesos nas folhas. Isso suaviza a previsão final, prevenindo que o modelo dependa excessivamente de um pequeno número de árvores com pesos muito altos e, consequentemente, melhorando sua capacidade de generalização para novos dados (CHEN; GUESTRIN, 2016).

Pruning (Poda) com o Parâmetro Gamma (γ)

Além da regularização dos pesos, o XGBoost controla a complexidade da estrutura da árvore através de um método de poda inteligente, governado pelo hiperparâmetro **Gamma** (γ). O Gamma atua como um limiar mínimo para o Ganho (*Gain*) necessário para que uma nova divisão em um nó seja realizada. Durante a construção da árvore, após o algoritmo calcular o maior *Gain* possível para um nó, ele realiza a seguinte verificação:

$$\text{Gain} > 0$$

Se o Ganho obtido pela melhor divisão possível não superar o valor de γ, o algoritmo considera que o benefício da divisão é muito baixo para justificar o aumento na complexidade do modelo. Consequentemente, a divisão é descartada e o nó se torna uma folha.

Esse mecanismo é uma forma eficaz de poda pós-crescimento (ou poda "preemptiva", dependendo da perspectiva), pois impede que a árvore continue a crescer em ramos que apenas capturam ruído ou padrões muito específicos dos dados de treinamento, resultando em um modelo final mais simples e robusto (CHEN; GUESTRIN, 2016).

4.1.4 Determinando o Output da Folha

Diferente de uma árvore de decisão tradicional que armazena a classe majoritária ou um valor médio em suas folhas, uma árvore no XGBoost armazena um valor numérico contínuo, frequentemente chamado de peso ou score da folha. Este valor é calculado para minimizar a função de perda para todas as amostras que terminam naquela folha específica. O peso ótimo (w_j^*) para uma folha j é derivado matematicamente e sua fórmula está diretamente relacionada aos gradientes (g_i) e Hessianas (h_i) das amostras na folha:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

Onde I_j é o conjunto de amostras na folha j e λ é o parâmetro de regularização L2. Essencialmente, cada árvore do ensemble não classifica diretamente uma amostra, mas contribui com um pequeno score que ajuda a empurrar a previsão final na direção correta para minimizar o erro geral (CHEN; GUESTRIN, 2016).

O Processo de Classificação: Da Soma dos Scores à Probabilidade

A previsão final para uma determinada amostra não é o resultado de uma única árvore, mas sim da soma aditiva dos scores de todas as árvores do conjunto. O processo se inicia com uma previsão base (geralmente 0.5 para problemas de classificação binária) e, a cada iteração, o score da nova árvore (ponderado por uma taxa de aprendizado, η) é somado à previsão anterior. A previsão bruta final, portanto, é um valor numérico que representa o logaritmo da chance (*log-odds*).

Para que este valor seja interpretável e utilizável para classificação, ele precisa ser convertido em uma probabilidade. Este processo ocorre em duas etapas conceituais:

1. **Entendendo Odds e Log-Odds:** A **chance (*Odds*)** de um evento é a razão entre a probabilidade de ele ocorrer e a probabilidade de ele não ocorrer ($\frac{P}{1-P}$). O **logaritmo da chance (*Log-Odds*)**, por sua vez, é simplesmente ln(Odds). O score bruto gerado pelo XGBoost é precisamente este *log-odds*. Esta transformação é útil porque, enquanto a probabilidade está contida no intervalo $[0, 1]$, o *log-odds* se estende por toda a reta dos números reais $(-\infty, +\infty)$, o que facilita o processo de otimização do modelo.

2. **A Função Logística (Sigmoid):** Para reverter o processo e converter o *log-odds* de volta em uma probabilidade, utiliza-se a **função logística**, também conhecida como função sigmoide. Ela mapeia qualquer valor real de volta para o intervalo $[0, 1]$. A fórmula é:

$$P(Y = 1|X) = \sigma(\text{score}) = \frac{1}{1 + e^{-\text{score}}}$$

Com a probabilidade final em mãos, a classificação é feita aplicando-se um limiar de decisão (por exemplo, se a probabilidade for > 0.5, a classe é 1; caso contrário, é

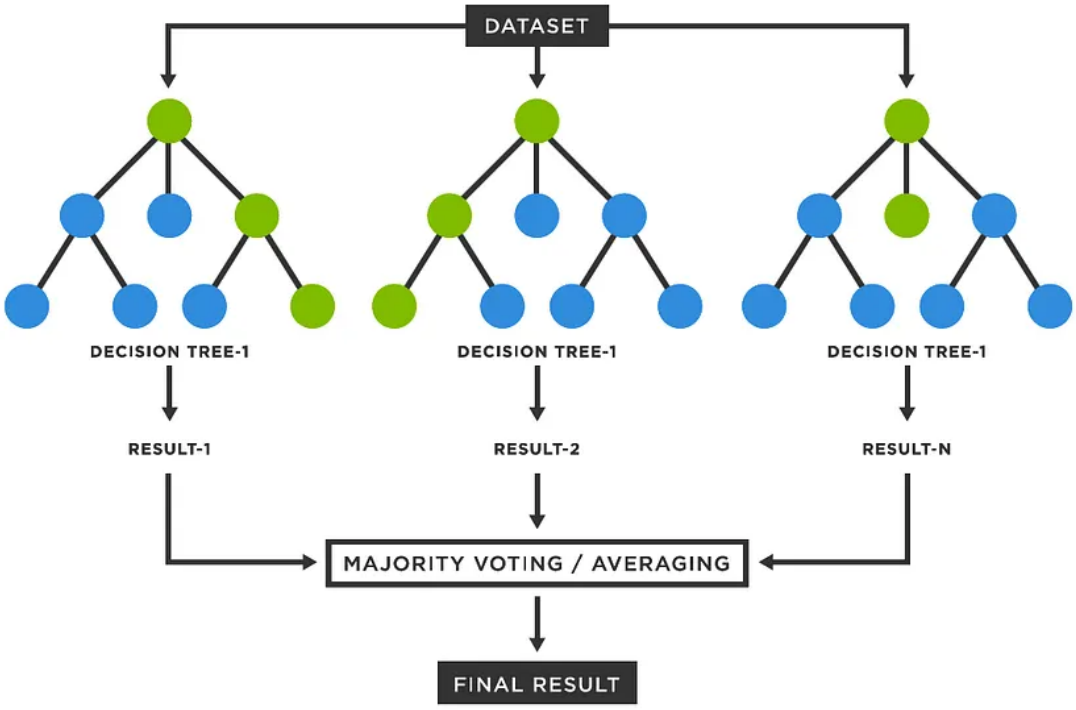
0). No contexto de um problema multiclasse como o deste projeto (“Compra”, “Venda”, “Espera”), uma generalização da função logística chamada **Softmax** é utilizada. A função Softmax recebe os scores brutos para cada uma das classes e os converte em um vetor de probabilidades, onde a soma de todas as probabilidades é igual a 1, permitindo a escolha da classe com a maior probabilidade (GOODFELLOW; BENGIO; COURVILLE, 2016).

4.2 Random Forest

4.2.1 Definições

Bagging (Bootstrap Aggregating): O Bagging, um acrônimo para Bootstrap Aggregating, é uma técnica de aprendizado de conjunto (ensemble learning) projetada primordialmente para reduzir a variância e combater o sobreajuste (overfitting). Sua filosofia operacional difere fundamentalmente do Boosting, pois se baseia em paralelismo em vez de sequenciamento. O processo consiste em criar múltiplos subconjuntos de dados a partir do conjunto de treinamento original através de amostragem bootstrap — um método de amostragem aleatória com reposição. Cada um desses subconjuntos é então utilizado para treinar um modelo de forma completamente independente e paralela. Finalmente, as previsões de todos os modelos individuais são agregadas, seja por votação majoritária (para classificação) ou pela média (para regressão), para formar uma única predição final que é tipicamente mais estável e robusta do que a de qualquer um dos seus componentes individuais (BREIMAN, 2001).

Figure 1: Arquitetura do Algoritmo Random Forest e Votação Majoritária



Gunay (2023).

Árvore de Decisão (Decision Tree): A Árvore de Decisão é o algoritmo de aprendizado supervisionado que serve como bloco de construção fundamental (o "aprendiz fraco") para o Random Forest. Conceitualmente, o modelo aprende uma estrutura hierárquica de regras de decisão simples, análogas a uma série de declarações "se-então-senão", que particionam recursivamente o espaço de características. O objetivo é criar nós terminais, ou folhas, que sejam os mais "puros" possíveis, ou seja, que contenham predominantemente amostras de uma única classe. A interpretabilidade é uma das suas principais características, pois a lógica de decisão pode ser facilmente visualizada e entendida. No entanto, árvores de decisão individuais, especialmente quando profundas, têm uma alta tendência ao sobreajuste, um problema que o método de ensemble do Random Forest visa mitigar (BREIMAN, 2001).

Métricas de Divisão: Índice Gini e Entropia: Para decidir qual característica e qual limiar utilizar para criar a melhor divisão em um nó, a árvore de decisão precisa de uma métrica quantitativa para medir a "pureza" ou "impureza" dos dados. Duas das métricas mais comuns para tarefas de classificação são o Índice Gini e a Entropia. O Índice Gini mede a probabilidade de uma amostra ser classificada incorretamente se fosse rotulada aleatoriamente de acordo com a distribuição das classes no nó. Um valor de Gini igual a 0 representa a pureza perfeita (todas as amostras pertencem à mesma classe). A Entropia, um conceito derivado da teoria da informação, mede o nível de incerteza ou desordem em um nó. O algoritmo então calcula o Ganho de Informação (Information Gain), que é a redução na entropia obtida após a divisão, escolhendo a divisão que maximiza esse ganho. Ambas as métricas servem ao mesmo propósito: encontrar a regra de divisão que resulta nos nós filhos mais homogêneos possíveis (BREIMAN, 2001).

4.2.2 A Dupla Aleatoriedade na Construção das Árvores

O Random Forest baseia seu sucesso em duas camadas de aleatoriedade projetadas para criar árvores de decisão diversificadas e descorrelacionadas. Essa diversidade é essencial para que os erros individuais de cada árvore sejam compensados pela votação do conjunto.

1. Amostragem Bootstrap (Aleatoriedade nas Amostras): Cada árvore na floresta não é treinada com o conjunto de dados original completo. Em vez disso, é gerada uma "amostra bootstrap" selecionando aleatoriamente dados do conjunto de treino com reposição. Isso significa que cada árvore aprende com uma versão ligeiramente diferente dos dados, onde algumas amostras podem se repetir e outras podem ser omitidas.

2. Seleção Aleatória de Características (Aleatoriedade nas Features): Esta é a inovação principal do modelo. Ao construir uma árvore, em cada nó que precisa ser dividido, o algoritmo não testa todas as características disponíveis. Ele sorteia um subconjunto aleatório de características e procura a melhor divisão apenas dentro desse subconjunto. Isso impede que características muito dominantes influenciem todas as árvores, forçando o modelo a encontrar diferentes padrões nos dados (BREIMAN, 2001).

4.2.3 Agregação e o Processo de Classificação Final: Votação Majoritária (Majority Voting)

Após a floresta ser construída, o processo de classificação de uma nova amostra é direto e democrático. A amostra é passada por todas as árvores de decisão independentes

que compõem o ensemble. Cada árvore individual "vota" em uma classe (no seu caso, "Compra", "Venda" ou "Espera") com base nas regras que aprendeu. A previsão final do Random Forest é simplesmente a classe que recebeu o maior número de votos do conjunto de árvores, tornando a decisão final uma "sabedoria do coletivo" que mitiga os erros e vieses de árvores individuais (BREIMAN, 2001).

4.2.4 Características Notáveis e Vantagens

Robustez ao Sobreajuste (Overfitting): O Random Forest é inerentemente robusto ao sobreajuste. A combinação da amostragem bootstrap e da seleção aleatória de características garante que as árvores do ensemble sejam descorrelacionadas. A votação majoritária final neutraliza os erros e vieses individuais de cada árvore, resultando em um modelo de baixa variância com alta capacidade de generalização para dados não vistos (BREIMAN, 2001).

Importância das Características (Feature Importance): Uma vantagem significativa do modelo é a capacidade de calcular a importância relativa de cada característica. Isso é tipicamente obtido medindo-se a média da redução de impureza (como o Índice Gini) que cada feature proporciona em todas as árvores da floresta. Essa métrica permite uma análise interpretativa dos resultados, identificando quais indicadores de mercado foram mais influentes para as decisões do modelo (BREIMAN, 2001).

Avaliação Out-of-Bag (OOB): O processo de amostragem bootstrap permite um método de validação interna conhecido como Out-of-Bag. Para cada árvore, as amostras do conjunto de treino que não foram selecionadas (o conjunto OOB) podem ser usadas para testar seu desempenho. Ao agregar as previsões OOB para todas as amostras, obtém-se uma estimativa de erro (o "OOB Score") que serve como um indicador confiável da performance do modelo sem a necessidade de um conjunto de teste separado (BREIMAN, 2001).

4.3 Deep Neural Networks

As Redes Neurais Profundas (Deep Neural Networks, DNNs), mencionadas na introdução deste trabalho como um exemplo de modelo "Caixa Preta", representam uma classe de algoritmos de aprendizado de máquina que se tornaram o estado da arte em diversas tarefas complexas. Elas são construídas a partir de unidades computacionais simples, inspiradas no neurônio biológico (GOODFELLOW; BENGIO; COURVILLE, 2016).

4.3.1 Fundamentos e Evolução do Neurônio Artificial

A unidade fundamental das redes neurais é o neurônio artificial, um modelo matemático cuja complexidade evoluiu para superar limitações iniciais.

1. **Perceptron (Rosenblatt):** Um dos primeiros modelos de neurônio artificial com uma regra de aprendizado. Sua arquitetura consiste em entradas (vetor $\mathbf{x}$), pesos sinápticos (vetor $\mathbf{w}$), um viés (b) e uma Função de Ativação Degrau (step function). O cálculo da saída u é um produto escalar (uma operação de Álgebra Linear) entre os pesos e as entradas: $u = \mathbf{w} \cdot \mathbf{x} + b$. A saída final é 1 se $u > 0$ e 0 caso contrário. A "Regra de Aprendizado do Perceptron" ajustava os pesos com base no erro da saída final (0 ou 1). Sua limitação notória é a incapacidade de resolver problemas não-linearmente separáveis, como o XOR (OU Exclusivo).

2. **ADALINE (Adaptive Linear Neuron):** Uma evolução crucial do Perceptron. O ADALINE calcula o erro usando a saída linear u, antes da função de ativação. Isso permitiu a aplicação da Regra Delta (Regra de Widrow-Hoff), que é a implementação direta do algoritmo de otimização Gradiente Descendente. O objetivo da Regra Delta é minimizar uma Função de Custo contínua (o Erro Quadrático $E = \frac{1}{2}(\text{real} - u)^2$), ajustando os pesos w_i na direção oposta ao gradiente ($\frac{\partial E}{\partial w_i}$) do erro. Esta regra estabeleceu a base matemática para o treinamento de redes mais complexas (GOODFELLOW; BENGIO; COURVILLE, 2016).

4.3.2 Arquitetura do Perceptron Multicamadas (MLP)

A limitação fundamental de modelos lineares como o Perceptron e o ADALINE é a sua incapacidade de resolver problemas não-linearmente separáveis, exemplificada pelo clássico problema do XOR. A solução para esta limitação é o Perceptron Multicamadas (MLP), uma arquitetura de rede neural que introduz uma ou mais camadas ocultas (hidden layers) entre as camadas de entrada e saída.

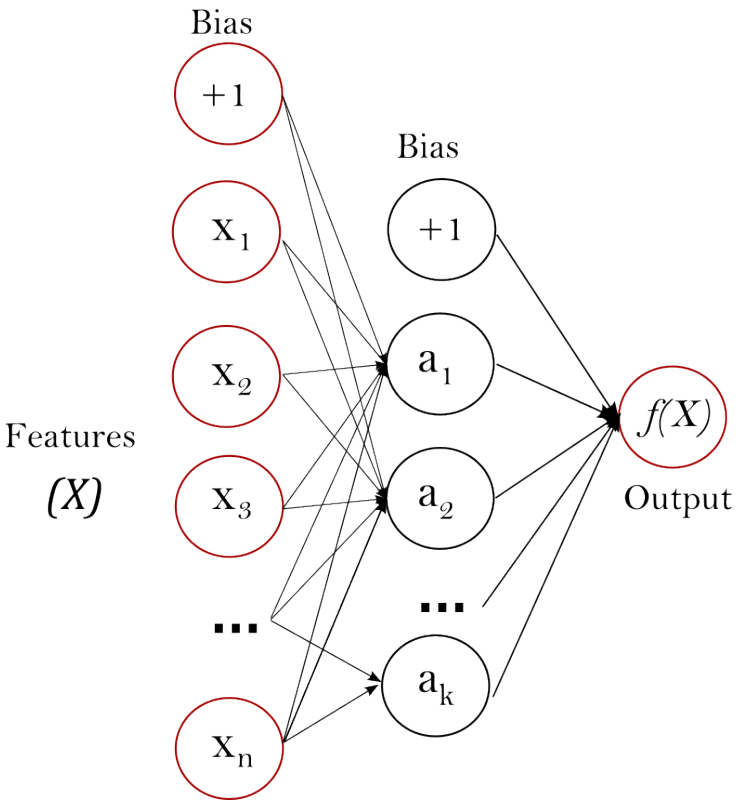
O MLP é a arquitetura canônica de uma rede neural feedforward, onde a informação flui em uma única direção (para a frente), sem ciclos. Sua estrutura é definida por:

1. **Camada de Entrada (Input Layer):** Esta camada não realiza computações. Ela é um vetor que simplesmente recebe e repassa as características de entrada ($\mathbf{x}$) para a primeira camada oculta. O número de neurônios nesta camada é igual à dimensionalidade das features do problema.

2. **Camadas Ocultas (Hidden Layers):** São as camadas intermediárias que dão à rede seu poder de representação. Cada neurônio em uma camada oculta é totalmente conectado a todos os neurônios da camada anterior. É a presença dessas camadas que permite à rede aprender fronteiras de decisão complexas e não-lineares, essencialmente "dobrando" o espaço das características para tornar os dados separáveis.

3. **Camada de Saída (Output Layer):** A camada final que produz o resultado da rede. Para um problema de classificação multiclasse como o deste projeto ("Compra", "Venda", "Espera"), esta camada tipicamente terá um neurônio para cada classe.

Figure 2: Arquitetura de um Perceptron Multicamadas (MLP)



Scikit-Learn Developers (2025a).

Pesos Sinápticos (W) e Biases (b)

Cada conexão entre um neurônio i na camada $(l-1)$ e um neurônio j na camada (l) possui um peso sináptico associado ($w_{ij}^{(l)}$). O conjunto de todos os pesos entre duas camadas pode ser representado de forma eficiente em uma matriz de pesos $\mathbf{W}^{(l)}$. Da mesma forma, cada neurônio j na camada (l) (exceto os da camada de entrada) possui um termo de bias ($b_j^{(l)}$), que pode ser representado como um vetor de biases $\mathbf{b}^{(l)}$.

Do ponto de vista da Álgebra Linear, a passagem de informação de uma camada $(l-1)$ para uma camada (l) é uma transformação afim: o vetor de ativações da camada anterior $\mathbf{a}^{(l-1)}$ é multiplicado pela matriz de pesos $\mathbf{W}^{(l)}$, e o vetor de bias $\mathbf{b}^{(l)}$ é somado, antes de passar pela função de ativação. Esses pesos e biases são os parâmetros $(\mathbf{W}, \mathbf{b})$ que a rede deve "aprender" durante o processo de treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016).

4.3.3 Funções de Ativação (O Papel da Não-Linearidade)

A capacidade de um MLP de modelar relações complexas não-lineares, como o problema do XOR, não advém do simples empilhamento de camadas. Se uma rede multicamadas utilizasse apenas transformações afins (operações de Álgebra Linear $\mathbf{Wx} + \mathbf{b}$) em suas camadas, o modelo inteiro seria matematicamente redutível a uma única transformação afim, possuindo o mesmo poder expressivo de um modelo linear simples, como o ADALINE.

A "mágica" do MLP reside na aplicação de uma Função de Ativação não-linear ($g(z)$) em cada neurônio, após a soma ponderada u. Esta função atua como um "portão" não-linear que determina a saída (ou "ativação") do neurônio, $a = g(u)$.

Para que o processo de aprendizado baseado em Gradiente Descendente (que será detalhado na seção 4.3.5) seja viável, a função de ativação deve ser diferenciável. Esta é a razão pela qual a função degrau (step function) do Perceptron original não é utilizada: sua derivada é zero em todos os pontos, exceto em um único ponto de descontinuidade, não fornecendo, assim, nenhum sinal de gradiente para o aprendizado. A evolução das funções de ativação é central para o sucesso do Deep Learning:

1. **Função Sigmoide:** Historicamente, foi a função mais utilizada. É definida como:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

   Ela comprime qualquer entrada real z para o intervalo [0, 1], o que era útil para interpretar a saída do neurônio como uma probabilidade. No entanto, sua derivada satura (tende a zero) em valores de entrada muito positivos ou muito negativos, levando ao problema do Desaparecimento do Gradiente (Vanishing Gradient), que será discutido na seção 4.3.6.

2. **Função Tangente Hiperbólica (Tanh):** Uma versão reescalada e centrada em zero da Sigmoide, definida como:

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

   Ela comprime as entradas para o intervalo [-1, 1]. Por ser centrada em zero, muitas vezes leva a uma convergência mais rápida que a Sigmoide, mas ainda sofre do mesmo problema de saturação e desaparecimento do gradiente.

3. **ReLU (Rectified Linear Unit):** A Unidade Linear Retificada é a função de ativação padrão na vasta maioria das redes neurais profundas modernas, devido à sua simplicidade e eficácia. Sua definição é:

$$\text{ReLU}(z) = \max(0, z)$$

Matematicamente, ela é uma função linear por partes: retorna z se z for positivo, e 0 caso contrário. Suas vantagens são cruciais:

- **Eficiência Computacional:** É uma operação de limiar trivial, muito mais rápida que os cálculos exponenciais da Sigmoide ou Tanh.
- **Mitigação do Desaparecimento do Gradiente**: Sua derivada é constante (igual a 1) para qualquer entrada positiva. Isso significa que, durante o backpropagation, o gradiente não é atenuado (não "desaparece") à medida que flui para trás através de múltiplas camadas, permitindo o treinamento de redes muito mais profundas.
- **Esparsidade:** Ao zerar as ativações negativas, a ReLU induz esparsidade nas ativações da rede, o que pode tornar a representação de dados mais eficiente (GOODFELLOW; BENGIO; COURVILLE, 2016).

4.3.4 O Processo de Predição (Forward Propagation)

O processo pode ser generalizado para qualquer camada (l) da rede: Cálculo da Soma Ponderada (Transformação Afim): A primeira etapa é calcular a entrada linear líquida, $\mathbf{u}^{(l)}$, para todos os neurônios na camada (l). Esta operação é um produto matriz-vetor entre a matriz de pesos $\mathbf{W}^{(l)}$ (que armazena todos os pesos que conectam a camada $l-1$ à camada l) e o vetor de ativações $\mathbf{a}^{(l-1)}$ da camada anterior, seguido pela adição do vetor de bias $\mathbf{b}^{(l)}$.

$$\mathbf{u}^{(l)} = \mathbf{W}^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

O material de aula ilustra este cálculo de forma escalar para um neurônio h_1 na camada oculta: $u_{h1} = x_1 w_1 + x_2 w_2 + b_1$ 2. A equação vetorial acima é a generalização que calcula $u_{h1}, u_{h2}, ...$ simultaneamente.Aplicação da Ativação Não-Linear: A entrada linear $\mathbf{u}^{(l)}$ é então passada, elemento por elemento, pela função de ativação g (como Sigmoide, Tanh ou ReLU) para produzir o vetor de ativação $\mathbf{a}^{(l)}$ da camada.

$$\mathbf{a}^{(l)} = g(\mathbf{u}^{(l)})$$

Este vetor de ativação $\mathbf{a}^{(l)}$ serve, por sua vez, como entrada $\mathbf{a}^{(l-1)}$ para a próxima camada, $(l+1)$. Este processo de duas etapas (transformação afim e ativação não-linear) é repetido para todas as camadas ocultas até que a camada final de saída seja alcançada. O vetor de ativação final $\mathbf{a}^{(L)}$ (onde L é a última camada) representa a predição bruta da rede, $\hat{\mathbf{y}}$. Este valor é então utilizado na Função de Custo para quantificar o erro da predição, o que inicia o processo de aprendizado descrito a seguir (GOODFELLOW; BENGIO; COURVILLE, 2016).

4.3.5 O Processo de Aprendizado (Otimização)

Após o Forward Propagation (seção 4.3.4) gerar uma predição ($\hat{\mathbf{y}}$), o processo de aprendizado se inicia. Este processo é um problema de otimização matemática cujo objetivo é ajustar iterativamente os parâmetros da rede ($\mathbf{W}$ e $\mathbf{b}$) para minimizar a discrepância entre as predições ($\hat{\mathbf{y}}$) e os valores reais ($\mathbf{y}$).

- **Função de Custo (Loss Function)**: O primeiro passo é quantificar essa discrepância através de uma Função de Custo (ou Loss Function). Esta função E agrega os erros de todas as saídas em um único valor escalar. Para tarefas de regressão, ou mesmo classificação, uma função comum é o Erro Quadrático Total (ou Sum of Squared Errors, SSE):

$$E_{total} = \sum_k \frac{1}{2}(y_k - \hat{y}_k)^2$$

Onde y_k é o valor real (desejado) e $\hat{y}_k$ é a predição da rede. O objetivo do treinamento é encontrar o conjunto de parâmetros $(\mathbf{W}, \mathbf{b})$ que minimiza o valor de E_{total}.

- **Otimização por Gradiente Descendente (GD)**: O algoritmo de otimização padrão para redes neurais é o Gradiente Descendente. Este é um método iterativo que "desce" a superfície de erro (definida pela função de custo) na direção de maior inclinação negativa.

O conceito central do Gradiente Descendente é o Gradiente (∇E), um vetor (ou matriz) que contém todas as derivadas parciais da função de custo E em relação a cada peso w_{ij} e bias b_j na rede ($\frac{\partial E}{\partial w_{ij}}$, $\frac{\partial E}{\partial b_j}$). Este vetor aponta para a direção de aumento mais íngreme do erro.

A regra de atualização dos pesos é, portanto, mover-se na direção oposta ao gradiente:

$$w_{novo} = w_{antigo} - \eta \frac{\partial E}{\partial w}$$

Onde η (eta) é a taxa de aprendizado (learning rate), um hiperparâmetro escalar que controla o tamanho do passo em cada iteração. Na prática, existem duas variações principais deste algoritmo:

1. **Batch Gradient Descent:** Calcula o gradiente ∇E usando a média de erro de todo o conjunto de dados de treinamento. É um cálculo preciso, mas computacionalmente muito caro e lento.

2. **Stochastic Gradient Descent (SGD):** Calcula o gradiente usando o erro de um único exemplo de treinamento (ou um pequeno "mini-batch"). O caminho é "ruidoso", mas é muito mais rápido, eficiente em termos de memória e ajuda a escapar de mínimos locais.

Backpropagation (Retropropagação)

O Gradiente Descendente dita o que fazer (atualizar os pesos usando o gradiente), mas não explica como calcular eficientemente o gradiente ∇E em uma rede profunda com milhões de parâmetros. O Backpropagation (Retropropagação) é o algoritmo que torna esse cálculo viável.

Em sua essência, o Backpropagation é uma aplicação computacionalmente eficiente da Regra da Cadeia (Chain Rule) do Cálculo. A função de custo E é uma função composta aninhada: o erro E depende da ativação da saída $\mathbf{a}^{(L)}$, que depende da soma ponderada $\mathbf{u}^{(L)}$, que depende da ativação $\mathbf{a}^{(L-1)}$, e assim por diante.

Para encontrar a influência de um peso $w^{(l)}$ em uma camada interna sobre o erro final E, a Regra da Cadeia é usada para decompor a derivada:

$$\frac{\partial E}{\partial w^{(l)}} = \frac{\partial E}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial u^{(L)}} \cdot \frac{\partial u^{(L)}}{\partial a^{(L-1)}} \cdot \frac{\partial a^{(L-1)}}{\partial u^{(L-1)}} \cdot \ldots \cdot \frac{\partial a^{(l)}}{\partial u^{(l)}} \cdot \frac{\partial u^{(l)}}{\partial w^{(l)}}$$

O algoritmo Backpropagation implementa isso de forma elegante, trabalhando "de trás para a frente":

1. **Cálculo do Erro na Saída:** O algoritmo primeiro calcula o "termo de erro" (delta, $\delta^{(L)}$) para a camada de saída L. Este termo é o produto da derivada da função de custo em relação à ativação ($\frac{\partial E}{\partial \mathbf{a}^{(L)}}$) e a derivada da função de ativação em relação à sua entrada linear ($\frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{u}^{(L)}}$).

2. **Retropropagação do Erro:** O termo de erro $\delta^{(L)}$ é então propagado para a camada anterior $(L-1)$ através das matrizes de peso. O erro da camada oculta $\delta^{(l)}$ é calculado como o erro da camada seguinte $\delta^{(l+1)}$ multiplicado (usando produto de matriz) pela transposta da matriz de pesos $\mathbf{W}^{(l+1)T}$, e então multiplicado (elemento a elemento) pela derivada da função de ativação da camada l.

3. **Cálculo do Gradiente:** Uma vez que o termo de erro $\delta^{(l)}$ é conhecido para uma camada, o gradiente $\frac{\partial E}{\partial \mathbf{W}^{(l)}}$ para os pesos daquela camada é simplesmente o produto vetorial (produto externo) entre o termo de erro $\delta^{(l)}$ e o vetor de ativação $\mathbf{a}^{(l-1)}$ da camada anterior.

Ao final deste processo, o Backpropagation calculou eficientemente o gradiente ∇E completo, que é então fornecido ao otimizador (SGD) para atualizar todos os pesos e biases da rede, completando uma iteração de treinamento (GOODFELLOW; BENGIO; COURVILLE, 2016).

4.4 Support Vector Machines (SVMs)

O Support Vector Machine (SVM) é um modelo de aprendizado supervisionado capaz de realizar classificação linear e não-linear. Sua abordagem teórica é baseada em princípios de geometria e otimização de margens, o que o torna eficaz em espaços de alta dimensão e robusto contra o sobreajuste.

4.4.1 Conceitos Básicos

O funcionamento e a aplicação prática do SVM se baseiam em três conceitos principais:

- Hiperplano de Margem Máxima (Maximal Margin): Em sua forma linear, o SVM busca encontrar o hiperplano de decisão (definido por $\mathbf{w} \cdot \mathbf{x} + b = 0$) que melhor separa as classes. O hiperplano ótimo é aquele que maximiza a margem, ou seja, a distância entre ele e os pontos de dados mais próximos de cada classe. Esses pontos específicos que delimitam a margem são chamados de Vetores de Suporte (Support Vectors), pois são os únicos que definem a solução do modelo.

- Margem Suave (Soft Margin): Para lidar com dados não-linearmente separáveis ou ruidosos, o SVM utiliza o conceito de margem suave. Ele introduz variáveis de folga ($\xi_i \geq 0$), que permitem que alguns pontos de dados violem a margem ou sejam classificados incorretamente, mediante uma penalidade. O hiperparâmetro de regularização C controla o equilíbrio (trade-off) entre maximizar a largura da margem e minimizar a soma dessas penalidades ($\sum \xi_i$).

- Validação Cruzada (Cross-Validation): Esta é uma técnica essencial para a avaliação e sintonização de hiperparâmetros do SVM (como C). O método divide os dados de treinamento em k subconjuntos (folds), treinando o modelo k vezes, onde cada fold é usado uma vez como conjunto de validação. O resultado é uma estimativa de performance mais robusta, usada para selecionar os parâmetros que oferecem a melhor capacidade de generalização (BERGSTRA; BENGIO, 2012).

4.4.2 Classificador Vetorial de Suporte (Support Vector Classifier)

O Classificador Vetorial de Suporte (Support Vector Classifier ou SVC) é a formalização do modelo SVM que utiliza os conceitos de margem máxima e margem suave. Ele define o problema de classificação como um problema de otimização quadrática convexa.

O objetivo do SVC é encontrar os parâmetros $\mathbf{w}$ (vetor de pesos) e b (bias) que definem o hiperplano de margem máxima, ao mesmo tempo em que gerencia as violações de margem (erros) através das variáveis de folga ξ_i e do parâmetro de regularização C.

O problema de otimização formal, que combina a maximização da margem com a penalização dos erros da margem suave, é expresso matematicamente como:

$$\min_{\mathbf{w},b,\xi} \left(\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i \right)$$

Este objetivo é sujeito a duas restrições para cada ponto de dado i:

1. $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$
2. $\xi_i \geq 0$

Onde m é o número de amostras, y_i é o rótulo da classe (definido como -1 ou +1), $\mathbf{x}_i$ é o vetor de características, e C é o hiperparâmetro de margem suave. O termo $\frac{1}{2}\|\mathbf{w}\|^2$ é o termo de regularização que, ao ser minimizado, maximiza a margem. O termo $C\sum \xi_i$ é a penalidade total para os erros de classificação (CORTES; VAPNIK, 1995).

4.4.3 Kernel Polinomial e o "Truque do Kernel"

O Classificador Vetorial de Suporte (SVC), conforme descrito na seção 4.4.2, é um modelo inerentemente linear. Ele falha em problemas onde a fronteira de decisão entre as classes é não-linear, como o clássico problema do XOR (também um desafio para os neurônios lineares da seção 4.3.1).

A solução do SVM para este problema é uma técnica matemática elegante e computacionalmente eficiente conhecida como "Truque do Kernel" (Kernel Trick).

- O Papel do Produto Escalar: A formulação matemática do problema de otimização do SVM (conhecida como sua forma "dual") revela que o algoritmo depende apenas do produto escalar (dot product) entre os vetores de características ($\mathbf{x}_i \cdot \mathbf{x}_j$), e não das coordenadas individuais dos vetores $\mathbf{x}$.
- A Relação de Alta Dimensão: O "truque" consiste em substituir este produto escalar simples, $\mathbf{x}_i \cdot \mathbf{x}_j$, por uma Função de Kernel $K(\mathbf{x}_i, \mathbf{x}_j)$. Esta função K calcula o produto escalar entre os vetores como se eles tivessem sido primeiro mapeados para um espaço de características de dimensão muito mais alta (ou até infinita) por uma função de mapeamento $\phi(\mathbf{x})$. O SVM pode, assim, encontrar um hiperplano linear neste espaço de alta dimensão, que corresponde a uma fronteira de decisão complexa e não-linear no espaço original de baixa dimensão. A vantagem computacional é que a função $\phi(\mathbf{x})$ (o mapeamento explícito) nunca precisa ser calculada.
- O Kernel Polinomial: Este é um exemplo de função de kernel que mapeia os dados para um espaço de características polinomiais. Sua fórmula é:

  $$K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i \cdot \mathbf{x}_j + r)^d$$

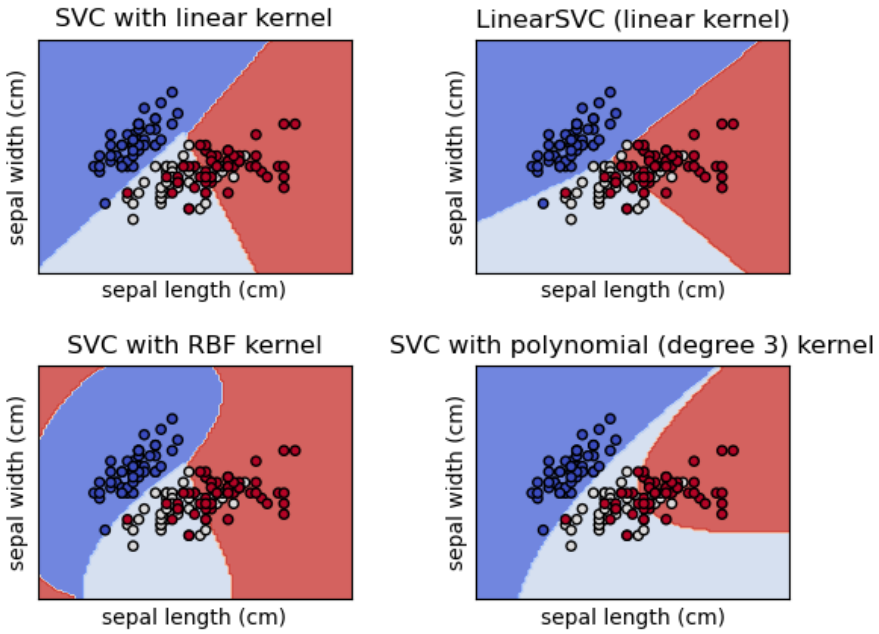
  Onde d é o grau do polinômio, γ (gama) é um coeficiente de escala e r é um termo de coeficiente independente. Um kernel com $d = 2$, por exemplo, implicitamente cria um espaço de características com todas as combinações quadráticas dos dados (ex: x_1^2, x_1x_2, x_2^2), permitindo ao SVM encontrar fronteiras de decisão quadráticas.

- A Função de Decisão Kernelizada: Quando um kernel é utilizado, a função de decisão para classificar um novo ponto $\mathbf{x}$ não é mais calculada usando um vetor $\mathbf{w}$ explícito. Em vez disso, a predição é determinada pela relação (kernel) do novo ponto com os Vetores de Suporte ($\mathbf{x}_i \in SV$) que foram identificados durante o treinamento:

$$f(\mathbf{x}) = \text{sinal}\left(\sum_{i \in SV} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

Onde α_i são os "multiplicadores de Lagrange" (essencialmente, os novos pesos) aprendidos pelo modelo, e y_i é o rótulo de classe (+1 ou -1) do vetor de suporte i (CORTES; VAPNIK, 1995).

Figure 3: Comparação de fronteiras de decisão SVM com diferentes Kernels



Fonte: Scikit-Learn Developers (2025b).

Nota: A imagem ilustra como diferentes kernels afetam a fronteira de decisão. Na linha superior, kernels lineares criam fronteiras retas. Na linha inferior, kernels não-lineares (RBF e Polinomial) criam fronteiras curvas complexas para separar as classes.

4.4.4 O Kernel Radial (Radial Kernel)

Além do Kernel Polinomial, a função de kernel não-linear mais amplamente utilizada e, em geral, mais poderosa, é o Kernel de Função de Base Radial (RBF), por vezes chamado de Kernel Gaussiano.

Definição Matemática: O Kernel RBF é definido pela seguinte equação:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$$

Onde:$\|\mathbf{x}_i - \mathbf{x}_j\|^2$: É o quadrado da distância Euclidiana entre os dois vetores de características. Esta é a operação de Álgebra Linear fundamental para este kernel.γ (gamma): É um hiperparâmetro que define o "alcance" da influência de um único vetor de suporte. Um valor de γ baixo significa que a influência é ampla (levando a uma fronteira de decisão mais suave), enquanto um γ alto significa que a influência é local e restrita (levando a uma fronteira mais complexa e ajustada aos dados).

Ao contrário do Kernel Polinomial, que é um modelo "global", o Kernel RBF é um modelo "local". Seu valor pode ser interpretado como uma medida de similaridade: ele é igual a 1 quando os pontos são idênticos ($\mathbf{x}_i = \mathbf{x}_j$) e decai exponencialmente em direção a 0 à medida que a distância entre eles aumenta. O "truque" aqui é que esta função de similaridade local mapeia implicitamente os dados de entrada para um espaço de características de dimensão infinita.

Devido a esse mapeamento para um espaço de dimensão infinita, o Kernel RBF é universalmente capaz de aproximar qualquer fronteira de decisão contínua. Isso o torna uma escolha padrão e extremamente flexível, capaz de resolver problemas de classificação muito complexos. A sua eficácia é controlada pelo ajuste fino dos hiperparâmetros C (da margem suave) e γ (do próprio kernel) (CORTES; VAPNIK, 1995; GOODFELLOW; BENGIO; COURVILLE, 2016).

5 Metodologia

5.1 Coleta e Definição dos Dados

A fundação do estudo consiste na coleta de dados diários OHLCV (Open, High, Low, Close, Volume) , uma abordagem padrão cuja premissa de análise é extensivamente validada na literatura de análise técnica (MURPHY, 1999).

O universo de ativos será composto por 5 a 10 ações de alta liquidez do Ibovespa (ex: PETR4, VALE3) e do SeP 500 (ex: AAPL, MSFT), garantindo representatividade.

A janela temporal (ex: 2015-2024) será definida para cobrir distintas fases de mercado, alinhando-se aos princípios de modelagem de séries temporais (HYNDMAN; ATHANASOPOULOS, 2018). As fontes de dados incluem o Yahoo Finance para dados de mercado e as respectivas autoridades monetárias (Banco Central do Brasil, Federal Reserve) para as taxas de juros.

5.2 Formulação do Problema e Rotulagem

O problema de regressão (previsão de preço) é transformado em uma tarefa de classificação supervisionada (Compra, Venda, Espera). A definição dos rótulos será baseada no "Método da Barreira Tripla", uma técnica avançada de rotulagem financeira (PRADO, 2018).

Este método utiliza um horizonte de previsão k (ex: 10 dias) e um limiar de retorno τ (ex: $\pm 2,5\%$). O retorno futuro $R_{t,t+k}$ é calculado e classificado conforme as barreiras:

- Se $R_{t,t+k} > +\tau \rightarrow$ Compra (1)
- Se $R_{t,t+k} < -\tau \rightarrow$ Venda (-1)
- Se $-\tau \leq R_{t,t+k} \leq +\tau \rightarrow$ Espera (0)

A justificativa central para esta abordagem é a filtragem de ruído: ela evita que pequenas variações de preço, que não representam sinais operacionais significativos, sejam classificadas como tendências, um princípio fundamental em finanças quantitativas (PRADO, 2018).

5.3 Engenharia de Atributos

Os dados brutos são convertidos em indicadores (features). Serão utilizados indicadores técnicos clássicos, como Médias Móveis, RSI, MACD e Bandas de Bollinger, cuja validade como preditores de tendência e momentum é estabelecida na literatura (MURPHY, 1999). A estes, serão somados os indicadores macroeconômicos (Taxa Selic, Fed Funds Rate).

Para os modelos sensíveis à escala (SVM e MLP), os atributos passarão por padronização (média 0, desvio padrão 1) através da transformação $z = \frac{x-\mu}{\sigma}$. Este pré-processamento é essencial para a correta convergência de otimizadores baseados em gradiente (GOODFELLOW; BENGIO; COURVILLE, 2016) e para a eficácia de modelos baseados em distância (CORTES; VAPNIK, 1995).

5.4 Modelagem e Treinamento

O desenho experimental deste trabalho segue precedentes da literatura que comparam o desempenho de Redes Neurais Profundas, Gradient-Boosted Trees e Random Forests na previsão de retornos de ações (KRAUSS; DO; HUCK, 2017; FISCHER; KRAUSS, 2018). Os quatro modelos do referencial teórico (Random Forest (BREIMAN, 2001), XGBoost (CHEN; GUESTRIN, 2016), SVM (CORTES; VAPNIK, 1995) e MLP (GOODFELLOW; BENGIO; COURVILLE, 2016)) serão treinados e comparados.

A divisão dos dados seguirá uma ordem cronológica estrita (ex: Treino 2015-2022, Validação 2023, Teste 2024). Esta abordagem é crucial para evitar o data leakage e o overfitting de backtest, um risco metodológico central na pesquisa financeira (BAILEY et al., 2014; PRADO, 2018).

O ajuste de hiperparâmetros (ex: C e γ do SVM, profundidade das árvores) será feito usando validação cruzada específica para séries temporais (ex: TimeSeriesSplit) no conjunto de validação. A busca pelos parâmetros seguirá o método de Random Search, que demonstrou ser mais eficiente que o Grid Search (BERGSTRA; BENGIO, 2012).

5.5 Métricas de Avaliação de Desempenho

A avaliação será dupla: estatística e financeira.

1. **Métricas de Classificação**: Serão utilizadas Precisão, Revocação (Recall) e F1-Score (cujas fórmulas estão detalhadas em), além da Acurácia global . Este conjunto

de métricas é necessário para uma análise sistemática de desempenho em classificação, superando os vieses da Acurácia em cenários potencialmente desbalanceados (SOKOLOVA; LAPALME, 2009; POWERS, 2011).

2. **Métricas Financeiras (Backtest)**: O desempenho prático será medido simulando uma estratégia de investimento no conjunto de teste . O retorno acumulado de cada modelo será comparado a uma estratégia de referência passiva (benchmark) de Buy and Hold. Uma estratégia algorítmica só possui valor prático se superar financeiramente este benchmark (PRADO, 2018).

3. **Análise de Interpretação:** Para entender quais atributos mais influenciaram as previsões, serão usadas técnicas de Feature Importance (para modelos de árvore) e SHAP values , conforme a metodologia unificada para interpretação de modelos (LUNDBERG; LEE, 2017).

6 Trabalhos Relacionados

A aplicação de *Machine Learning* (ML) para a previsão de séries temporais financeiras tem sido objeto de extensa investigação acadêmica. Esta seção revisa os estudos seminais que fundamentam a escolha dos algoritmos, a metodologia de validação e as técnicas de interpretação adotadas neste projeto. A literatura selecionada divide-se em três eixos principais: comparação de desempenho entre famílias de algoritmos, metodologias robustas de rotulagem e validação, e interpretabilidade de modelos complexos.

6.1 Comparação de Modelos em Mercados Financeiros

A superioridade de modelos não-lineares sobre abordagens estatísticas tradicionais e lineares tem sido demonstrada empiricamente em diversos estudos recentes.

Krauss, Do e Huck (2017) realizaram um estudo comparativo influente aplicando Redes Neurais Profundas (DNN), *Gradient-Boosted Trees* (GBT) e *Random Forests* (RF) para prever retornos no índice S&P 500 entre 1992 e 2015. Os autores implementaram uma estratégia de *trading* estatístico e encontraram que, após os custos de transação, os modelos de *Deep Learning* e *Gradient Boosting* superaram consistentemente o desempenho do *Random Forest* e do mercado em geral. Este trabalho serve como a base primária para o desenho experimental da presente pesquisa, justificando a seleção das mesmas famílias de algoritmos (Árvores e Redes Neurais) para uma nova avaliação comparativa.

Expandindo essa linha de investigação, Fischer e Krauss (2018) focaram no uso de redes neurais recorrentes (LSTM) para o mercado financeiro. Ao compararem LSTMs com *Random Forests* e Redes Neurais Profundas padrão (MLP), os autores demonstraram que a capacidade das LSTMs de capturar dependências temporais resultou em uma acurácia preditiva superior e retornos financeiros mais robustos. Embora o presente trabalho foque em arquiteturas MLP e *ensemble*, os achados de Fischer e Krauss corroboram a hipótese de que modelos com maior capacidade de representação de não-linearidades tendem a capturar melhor os padrões complexos de mercado.

6.2 Rigor Metodológico e Prevenção de Overfitting

Um desafio crítico na literatura de ML financeiro é a validação de estratégias. Métodos tradicionais de validação cruzada (k-fold com embaralhamento) frequentemente falham

em séries temporais, levando a resultados excessivamente otimistas.

Bailey et al. (2014) e López de Prado (2018) destacam os riscos severos de *overfitting* em *backtests* financeiros (o "falso positivo" estratégico). López de Prado (2018), em particular, argumenta que a rotulagem de dados baseada apenas em retornos de tempo fixo introduz ruído excessivo, dificultando o aprendizado do modelo. Como solução, o autor propõe o "Método da Barreira Tripla" (*Triple-Barrier Method*), que define o sucesso de uma operação baseada em limites de lucro (*take-profit*) e perda (*stop-loss*) dinâmicos, além do tempo. Esta metodologia de rotulagem e a exigência de divisão cronológica estrita dos dados são adotadas centralmente neste projeto para mitigar os vieses metodológicos apontados pelos autores.

6.3 Interpretabilidade de Modelos

O uso de modelos complexos ("Caixa-Preta"), como Redes Neurais e *Boosting*, cria uma barreira de confiança para a tomada de decisão financeira. A capacidade de explicar por que uma decisão de compra ou venda foi tomada é tão relevante quanto a acurácia da previsão.

Neste contexto, Lundberg e Lee (2017) unificaram diversos métodos de interpretação de modelos ao introduzirem os valores SHAP (*SHapley Additive exPlanations*). Baseado na teoria dos jogos, o SHAP atribui a cada *feature* um valor de importância para uma predição específica, garantindo consistência e precisão local. A inclusão desta técnica no presente trabalho visa responder a uma lacuna comum em estudos de previsão financeira, permitindo não apenas classificar a tendência, mas também identificar quais indicadores técnicos ou macroeconômicos foram determinantes para a decisão do algoritmo.

7 Resultados Esperados

Com base nas informações e técnicas definidas até o momento, esperam-se os seguintes resultados ao final deste trabalho:

1. Espera-se que modelos mais complexos apresentem um desempenho de classificação (medido pelo F1-Score) superior a modelos mais simples, observando a possível superioridde destes de captar padrões não lineares complexos e as interações inerentes aos mercados financeiros.

2. Desempenho financeiro superior ao benchmark de 'Buy and Hold' usando as estratégias de backtest simuladas, visando provar que o uso do limiar faz com que ruídos sejam mais facilmente detectados e possibilitando identificar sinais operacionais com valor prático.

3. Identificar preditores relevantes, como indicadores macroeconômicos, esperando que esses fatores externos apareçam com significativa importância durante o projeto, ao lado de indicadores técnicos classicos de momentum (como RSI) e tendência (como Médias Móveis).

4. Identificar possíveis diferenças na previsibilidade entre os mercados Brasileiro e Americano.

References

BAILEY, D. H. et al. The probability of backtest overfitting. *The Journal of Portfolio Management*, v. 40, n. 5, p. 60–79, 2014.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, v. 13, p. 281–305, 2012.

BREIMAN, L. Random forests. *Machine Learning*, v. 45, n. 1, p. 5–32, 2001.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. [S.l.: s.n.], 2016. p. 785–794.

CORTES, C.; VAPNIK, V. Support-vector networks. *Machine Learning*, v. 20, n. 3, p. 273–297, 1995.

FISCHER, T.; KRAUSS, C. *Deep learning with long short-term memory networks for financial market predictions*. [S.l.: s.n.], 2018. v. 270. 654–669 p.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016.

GUNAY, D. *Random Forest*. 2023. Medium. Disponível em: <https://medium.com/@denizgunay/random-forest-af5bde5d7e1e>. Acesso em: 20 nov. 2025.

HYNDMAN, R. J.; ATHANASOPOULOS, G. *Forecasting: Principles and Practice*. Melbourne, Australia: OTexts, 2018.

KRAUSS, C.; DO, X. A.; HUCK, N. Deep neural networks, gradient-boosted trees, random forests: Stock return prediction on the s&p 500. *European Journal of Operational Research*, v. 259, n. 2, p. 689–702, 2017.

LUNDBERG, S. M.; LEE, S.-I. A unified approach to interpreting model predictions. In: *Advances in Neural Information Processing Systems (NIPS)*. [S.l.: s.n.], 2017. p. 4765–4774.

MURPHY, J. J. *Technical Analysis of the Financial Markets*. [S.l.]: New York Institute of Finance, 1999.

POWERS, D. M. W. Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, v. 2, n. 1, p. 37–63, 2011.

PRADO, M. L. de. *Advances in Financial Machine Learning*. Hoboken, NJ: Wiley, 2018.

Scikit-Learn Developers. *Neural network models (supervised)*. 2025. Scikit-Learn Documentation. Disponível em: <https://scikit-learn.org/stable/modules/neural_networks_supervised.html>. Acesso em: 20 nov. 2025.

Scikit-Learn Developers. *Support Vector Machines*. 2025. Scikit-Learn Documentation. Disponível em: <https://scikit-learn.org/stable/modules/svm.html>. Acesso em: 20 nov. 2025.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, v. 45, n. 4, p. 427–437, 2009.